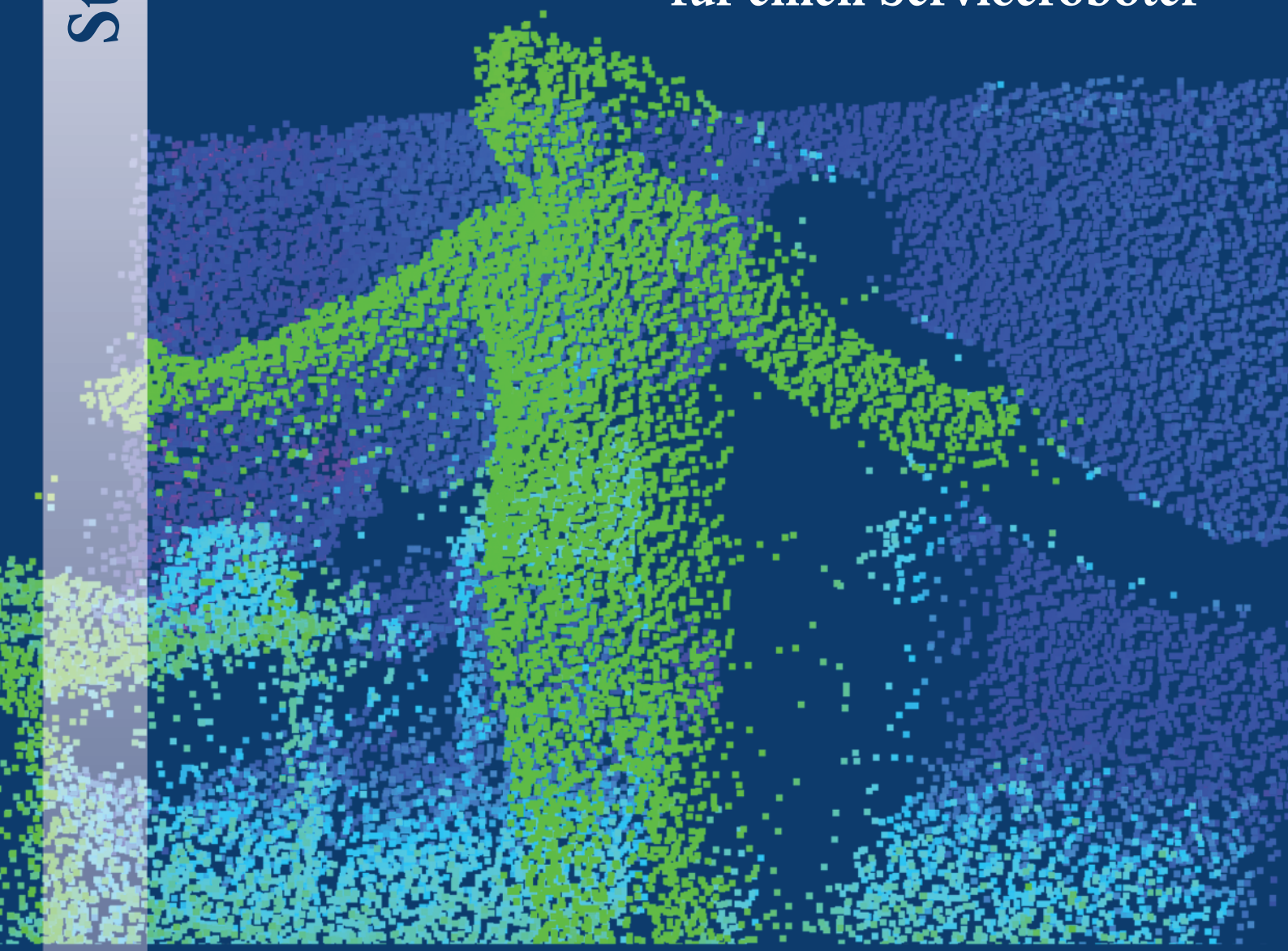
Steffen A. Knoop

Interaktive Erstellung und Ausführung von Handlungswissen

für einen Serviceroboter



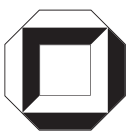
universitätsverlag karlsruhe

Steffen A. Knoop

Interaktive Erstellung und Ausführung von Handlungswissen für einen Serviceroboter

Interaktive Erstellung und Ausführung von Handlungswissen für einen Serviceroboter

von
Steffen A. Knoop



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Informatik, 2007
Referenten: Prof. Dr.-Ing. Rüdiger Dillmann, Prof. Dr.-Ing. Gerhard Sagerer
Tag der mündlichen Prüfung: 19.07.2007

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2007
Print on Demand

ISBN: 978-3-86644-189-7

Interaktive Erstellung und Ausführung von Handlungswissen für einen Serviceroboter

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik

der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation
von

Steffen A. Knoop

aus Heidelberg

Tag der mündlichen Prüfung:	:	19.7.2007
Erster Gutachter	:	Prof. Dr.-Ing. Rüdiger Dillmann
Zweiter Gutachter	:	Prof. Dr.-Ing. Gerhard Sagerer

Danksagung

Die vorliegende Doktorarbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter in den Jahren 2003 bis 2007 am Lehrstuhl meines Doktorvaters Prof. Dr.-Ing. Rüdiger Dillmann. Bei ihm möchte ich mich für die Unterstützung, das in mich gesetzte Vertrauen und die gewährte wissenschaftliche Freiheit bedanken; ich hoffe, dass diese Freiheit auch künftigen Doktorandengenerationen so gewährt werden kann, wie ich sie erfahren durfte.
Für die freundliche Übernahme des Korreferats möchte ich mich aufs Herzlichste bei Herrn Prof. Dr.-Ing. Gerhard Sagerer bedanken. Die kubanische Volleyball-Frauen-Nationalmannschaft werde ich nie vergessen! Des Weiteren möchte ich Herrn Prof. Dr.-Ing. Uwe D. Hanebeck und Herrn Prof. Dr. rer. nat. Ralf Reussner herzlich dafür danken, dass sie als Prüfer zu meiner Doktorprüfung beigetragen haben.

Mein größter Dank geht an die "alte Garde" des Instituts: Markus Ehrenmann, Oliver Rogalla und Raoul Zöllner führten mich bereits zu Studienzeiten in die bunte Welt der Robotik ein; erst als Hiwi, später während der Studienarbeit. Danke, ihr habt mir immer Mut gemacht! Ich erinnere mich auch gerne an Fahrradausflüge, Bunkerpartys, Autoreparaturabende und Whiskyrunden.
Dazu gehören auch Tobias Salb, Medizinkollege mit Robotikneigungen, Björn Giesler, *bgtools*-Vater und (un-)geduldiger Verfechter der hohen Ideale des Softwareentwurfs, Peter Steinhaus und Regine Becher, SFB-Veteranen der ersten Stunde, sowie Sascha Seifert, Leidensgenosse bis zum Schluss. Von ihnen allen habe ich sowohl in meiner Zeit als Student als auch in meiner Anfangszeit als Doktorand viel gelernt und profitiert. Liebe Alt-Kollegen: Ihr habt mächtig große Fußstapfen hinterlassen!

Ein großes Dankeschön geht an meine Büronachbarn Stefan Vacek, Michael Pardowitz, und (schon wieder) Raoul. Das waren noch Zeiten, als sich ein ganzes Büro auf die gleiche Musik einigen konnte! Auch wenn Raoul sich ungleich weniger einigte als der Rest...
Viel Erfolg wünsche ich der jungen Kopfhörergeneration: Martin Lösch und Sven Schmidt-Rohr übernehmen 310 mit viel Elan und Ideen. Ich wünsche ihnen, dass sie Gelegenheit haben, diese Ideen auch in Ruhe umzusetzen!

Ein riesiges Dankeschön gilt dem Sekretariat. Nela Redzovic, Christine Brand und Isabelle Wappler bilden das Herz des Lehrstuhls und sorgen für das Wohlergehen im Allgemeinen und Speziellen. Vielen Dank für die Organisation rund um meine Promotionsprüfung, die offenen Ohren, die man bei euch immer findet, die Vermittlerrolle, die euch oft genug zufiel, unzählige amüsante Diskussionen beim Mittagessen, Heidelbeergetzen und – nicht zuletzt – die wundervollen Küchendienstpläne!

Großer Dank gebührt allen Studenten, die im Rahmen von Studien- und Diplomarbeiten oder als Hiwis zu dieser Arbeit beitrugen. Besonders möchte ich Sven Schmidt-Rohr und Christian Au hervorheben; ihr Einsatz insbesondere bei der Umsetzung und Integration auf den Robotern sowie der Vorbereitung verschiedener Demos hat viele Erfolge erst möglich gemacht.

Ein großer Dank geht weiterhin an Hertlingshausen, Unterselighof 18. Sowohl die ersten als auch die letzten Seiten dieser Ausarbeitung sind dort, in der Ruhe des Pfälzer Waldes, entstanden. Besser als die Ruhe dort ist nur noch der Wein!

Großen Dank schulde ich außerdem den Korrekturlesern meiner Ausarbeitung. (Schon wieder) Markus Ehrenmann, Heike Klamm-Ewald und meine Schwester Hilke Knoop ertrugen Spaghettisätze, Tippfehler und konsequente Inkonsequenz (*"Warum ist das hier kursiv??"*) auf über 200 Seiten und gaben mir unzählige Korrekturen und Hinweise, von denen diese Arbeit nur profitieren konnte.

Mein größter Dank geht an meine Eltern Hella und Bernd Knoop, die mich nicht nur während der gesamten Ausbildung immer unterstützt und in meinen Zielen bestärkt haben. Danke für die Liebe, den Rückhalt und die Anstöße zur richtigen Zeit. Das gleiche gilt auch meiner Schwester Hilke, ohne die diese Arbeit nicht einmal von außen so aussehen würde, wie sie nun ist.

Der größte Dank geht an meine Frau Simone. Ihr Beitrag zum Gelingen dieser Arbeit ist unermesslich. Deshalb widme ich ihr diese Arbeit.

28. August 2007 Steffen Knoop

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Erschaffung intelligenter Maschinen ist schon seit langem ein Menschheitstraum: In Büchern und Filmen übernahmen menschenähnliche Roboter schon zu Zeiten Hauptrollen, in denen ihre technische Realisierbarkeit noch in unerreichbarer Ferne stand. Ob Freund, Arbeitskraft oder Helfer, ob unkontrollierbare Maschine oder gar Monster – Roboter üben auf den Menschen offensichtlich eine unwiderstehliche Anziehungskraft aus. Und vielleicht ist gerade diese Faszination Teil des Antriebs für intensive wissenschaftliche Forschungen, durch die die technische Realisierbarkeit von Service- und Assistenzrobotern mittlerweile in greifbare Nähe gerückt ist.
Mögen sie noch so verschieden sein, spiegeln doch alle Fiktionen und Visionen solcher Maschinen ähnliche Erwartungen an das Wesen "Roboter" wider – Anforderungen, die sich auch in der Forschung als Leitlinien wiederfinden. Ganz oben stehen die Forderungen nach Intelligenz und Flexibilität. Aus diesen primären Eigenschaften ergeben sich weitere: Autonomes, selbständiges Handeln, Einfühlsamkeit, ja selbst eigene Gefühle werden den Maschinen zugeschrieben. Gleichzeitig zeigen Filme wie "Terminator" oder "The Marix" die Kehrseite der Medaille: Das Verhältnis Mensch-Roboter ist zutiefst gespalten durch die Faszination der technischen Möglichkeiten auf der einen Seite und der Urangst vor einer sich eigenständig entwickelnden, unkontrollierbaren Spezies auf der anderen. Schon 1942 stellte Isaac Asimov deshalb in seinen Science-Fiction-Erzählungen in *I, Robot* [Asimov 42] die Grundregeln auf, an die sich Roboter im Umgang mit Menschen zu halten haben: Diese sollen sicherstellen, dass kein Roboter einem Menschen Schaden zufügt.

Eine Zielsetzung der Robotik ist letzten Endes, den Menschen, oder zumindest Teile seiner Fähigkeiten, mit einem künstlichen System zu kopieren. Dies stellt zwar ein äußerst ambitioniertes Ziel dar, das viel Raum und viele Freiheiten in der Entwicklung lässt. Doch muss sich die recht junge Disziplin der Robotik auch an diesem Ziel messen und messen lassen. Gerade der Bau von menschenähnlichen Robotern – derzeit vor allem in Japan und den USA praktiziert – weckt die damit verbundenen Erwartungen auch an das Verhalten, die Flexibilität, die Intelligenz und den Umgang mit Menschen.
Vergleicht man diesen Anspruch mit dem bisher Erreichten, so zeigt sich, dass der Weg bis zu einem tatsächlich flexiblen, intelligenten, menschenähnlichen System noch weit ist; die heutige Realität in der Entwicklung von flexiblen Servicerobotern sieht wesentlich ernüchternder aus. Noch lange nicht haben diese Maschinen ein Stadium erreicht, das sich mit den Ideen und Vorstellungen der Science-Fiction-Autoren und Filmemacher messen lassen kann.

Das gilt sowohl für die mechatronischen Komponenten, als auch für die Regelung sowie die Perzeptions-, Entscheidungs- und Interaktionsmechanismen. Bei der Mechatronik fehlen Entwicklungen sowohl in der Materialforschung als auch bei Sensorik und Aktorik; bei der Verarbeitung fehlen durchgängige und konsistente Repräsentationen, holistische Verarbeitungskonzepte, und nicht zuletzt die Methoden und Verfahren, vorhandenes sensorisches und aktorisches Wissen in allen Facetten zu nutzen.

Erste Schritte wurden bereits im Bereich der Unterhaltungsindustrie gemacht: Roboter wie der künstliche Hund *Aibo* zielen in diese Richtung, und in den letzten Jahren drängen immer mehr auch humanoide Spielzeugroboter auf den Markt. Genau besehen sind die Fähigkeiten dieser Systeme jedoch sehr gering, und beschränken sich insbesondere bei den sensorischen wie auch bei den Entscheidungskomponenten auf einfachste Eigenschaften.

Gerade diese Geräte zeigen deshalb einen generellen Trend der aktuellen Robotikforschung: Oft wird wesentlich mehr Wert auf die äußere Form und auf die Ähnlichkeit in Erscheinung und Bewegung mit dem Menschen gelegt als auf die Perzeptions- und Entscheidungsfähigkeiten. Dies ist zum Teil darin begründet, dass der mechatronische Aufbau eines menschenähnlich aussehenden Systems technisch lösbar ist und sich nicht wesentlich vom Bau anderer mechatronischer Komponenten unterscheidet; ein gutes Beispiel ist hier die japanische Automobilindustrie, die den Bau humanoider oder teilhumanoider Roboter stark forciert. Die Funktionsweise der Perzeption und Entscheidung beim Menschen, oft unter dem Begriff der *kognitiven Fähigkeiten* zusammengefasst, ist dagegen noch nicht umfassend erklärbar und reproduzierbar. Diese Fähigkeiten können heutzutage nur in Ausschnitten und auf bestimmte Domänen beschränkt nachgebildet werden.

Das Vorbild menschlicher Eigenschaften und Fähigkeiten bezieht sich, mit wechselnden Schwerpunkten, auf alle Aspekte der entwickelten Roboter: Die äußere Form, die Kinematik sowie die Dynamik des Menschen wird mit humanoiden Robotern nachgebildet; hier wird besonders auf die Form sowie die Aktorik Wert gelegt. Perzeptive Fähigkeiten des Menschen werden genauso nachempfunden: Verschiedene Sinne des Menschen werden in der und für die Servicerobotik zum Vorbild genommen. Den Schwerpunkt bildet dabei das *Maschinelle Sehen*, was sicherlich durch die herausragende Bedeutung des Sehsinns für den Menschen zu erklären ist. Aber auch der Tastsinn und das Gehör werden auf Robotersysteme übertragen, um die Interaktion zwischen dem Roboter und seiner Umwelt sowie den Menschen reicher zu gestalten. Genauso wird die Fähigkeit des Menschen zum autonomen Handeln und rationalen Entscheiden imitiert. Verstärkte Aufmerksamkeit erhält in den letzten Jahren dazu die Entwicklung multimodaler Interaktionsmechanismen, ebenfalls mit menschlichen Interaktionsformen als Vorbild. Dazu zählen Sprache und Gestik genauso wie Mimik, Körpersprache und physische Interaktion.

Bei den Versuchen, menschliche Fähigkeiten der Perzeption, Ausführung und Entscheidung zu modellieren und nachzubilden, haben sich in der Forschung zwei unterschiedliche Strömungen herausgebildet. Oftmals wird versucht, die Prozesse beim Menschen nachzubilden, indem die Struktur und die Verarbeitungsmechanismen des Menschen imitiert werden; die Mächtigkeit und die Fähigkeiten sollen über die Nachbildung der Struktur und der Mechanismen entstehen. Dem entgegen steht die Philosophie, einen Roboter als ein technisches System zu behandeln, dessen Aufbau grundsätzlich von dem des Menschen abweicht und entsprechend auch eine andere interne Struktur verlangt.

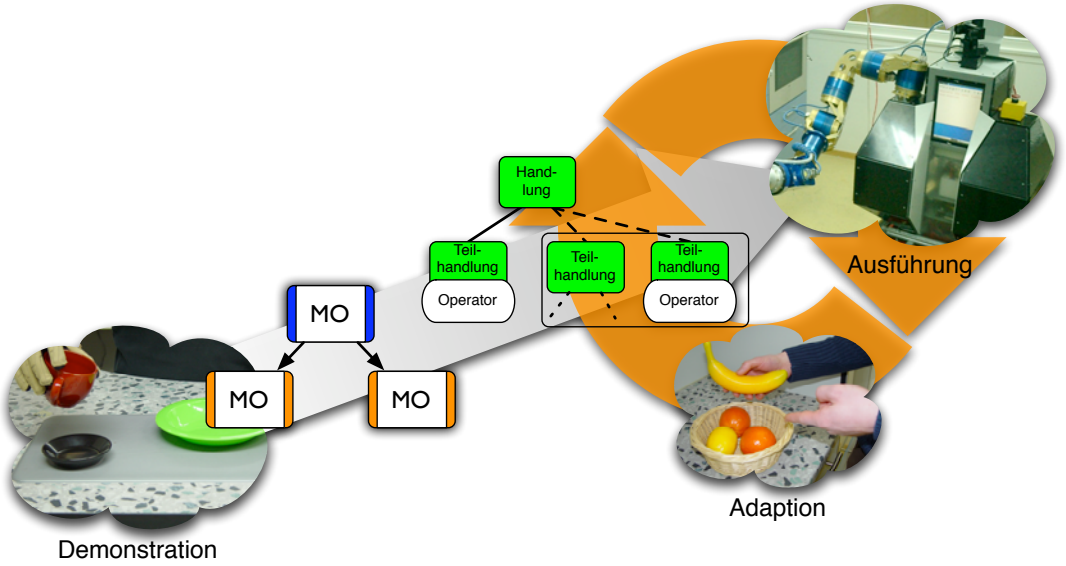


Abbildung 1.1: Genese und Adaption von Handlungswissen in enger Kooperation zwischen Mensch und Roboter. Aus der Vorführung wird ein Makrooperator extrahiert, der in ein ausführbares Roboterprogramm überführt werden kann. Dieses kann von Mensch und Roboter verändert und parametriert werden.

Diese Arbeit will im Sinne dieser zweiten Interpretation einen Beitrag dazu leisten, die Fähigkeiten der Entscheidung und Ausführung sowie der Perzeption existierender Systeme zu erweitern. Der Schwerpunkt liegt dabei nicht auf der menschenähnlichen Erscheinung des Roboters, seiner Bewegungen oder seiner Entscheidung; dieser soll vielmehr als das technische System wahrgenommen werden, das er tatsächlich darstellt. Es wird der Versuch unternommen, Komponenten der Handlungssteuerung, der Ausführung und der Interaktion zu entwickeln, die ein autonomes und flexibles Handeln eines technischen Systems sowohl gegenüber seiner Umwelt als auch gegenüber dem Menschen ermöglichen und vereinfachen.

1.1 Einordnung und Problemstellung

Servicerobotersysteme unterscheiden sich von klassischen Industrierobotern zum einen durch die notwendige Flexibilität in teilweise unbekannten und dynamischen, menschzentrierten Umgebungen. Die erforderte Flexibilität beinhaltet dabei insbesondere die Möglichkeit, Handlungswissen zu lernen, zu adaptieren und zu parametrieren. Zum anderen sollen die angestrebten Robotersysteme in der Lage sein, mit Menschen zu interagieren und zu kommunizieren, die nicht notwendigerweise dediziertes Wissen über die Funktionsweise des Roboters besitzen. Das System muss also in der Lage sein, Äußerungen, Gesten und Körpersprache eines Menschen zu verstehen und zu interpretieren. Abbildung 1.1 verdeutlicht dies.
Die vorliegende Arbeit beschäftigt sich insbesondere mit diesen zwei Aspekten: Der Repräsentation, Akquisition und Ausführung von Handlungswissen für ein dediziertes Robotersystem, sowie der Beobachtung und Interpretation von Bewegungen des Menschen bei der Ausführung kooperierender oder proaktiver Handlungen.

1.1.1 Akquisition und Genese von Handlungswissen

Um das vorhandene Handlungswissen erweiterbar und parametrierbar zu halten, muss es also möglich sein, dem Roboter neues Wissen einerseits über Objekte und Sachverhalte, andererseits aber vor allem über Strategien zur Problemlösung zugänglich zu machen. Einen ambitionierten Ansatz bietet hierzu das *Programmieren durch Vormachen (PdV)* [Rogalla 02, Friedrich 98].
Hierbei wird die Demonstration einer Aufgabe durch den Menschen vorgenommen und vom Robotersystem beobachtet. Dazu wird verschiedene Sensorik eingesetzt, um die Handlung, bestehend aus Manipulationen, Griffen und weiteren Aktionen, möglichst genau zu erfassen. Aus den beobachteten Daten werden relevante Aktionen extrahiert, anhand der Umweltanalyse die Effekte der Vorführung bestimmt und modellbasiert eine Sequenz von Aktionen aufgestellt, die der Demonstration entspricht. Diese Handlungshypothese kann vom Benutzer in einem Bearbeitungsschritt betrachtet und verfeinert werden.
Nach einem Generalisierungsschritt entsteht so eine abstrakte Handlungsbeschreibung, die hierarchisch aus einzelnen Elementaroperationen aufgebaut ist. Mit diesem Verfahren wird roboterunabhängiges Handlungswissen erzeugt, das relevante Abschnitte der Vorführung symbolisch kodiert enthält. Diese Handlungsbeschreibung wird in einer Wissensbasis abgelegt und soll zur Ansteuerung eines Robotersystems genutzt werden.
Im Gegensatz dazu muss ein ausführbares Roboterprogramm Aktionen wie Fehlerbehandlung, Umwelterfassung, Ausgaben für den Benutzer usw. beinhalten. Zusätzlich sind je nach Ausführungsrahmen roboterinterne Aktionen und Kontrollmechanismen zu beachten, um eine fehlerfreie Ausführung zu gewährleisten. Besonders die Umwelterfassung und die Fehlerbehandlung sind aus den beobachteten Sequenzen nicht zu extrahieren; aufgrund der Annahme der *geschlossenen Welt* sowie der Schwierigkeit der Erfassung während der Demonstration sind in der beobachteten Sequenz keine Aktionen zur Umwelterfassung enthalten. Verwendete Vorführungen sind zudem grundsätzlich korrekt im Sinne des Ziels der Aufgabe; die Vorführungen decken also keine Fehlerfälle ab. Darüber hinaus ist es nur mit großem Aufwand möglich, sämtliche Fehlerfälle und die Lösungswege mit den Demonstrationen zu beschreiben.
Dieses Handlungswissen muss also in der Handlungsbeschreibung des Roboters enthalten sein, kann aber nicht direkt aus der Vorführung abgeleitet werden.

1.1.2 Beobachtung des Menschen zur intuitiven Interaktion

Während die eigentliche Vorführung im Rahmen des Programmierens durch Vormachen in einem so genannten *Training Center* stattfindet, um mit dedizierter Sensorik möglichst viel Information aus einer Vorführung extrahieren zu können [Ehrenmann 02], muss eine nachfolgende Adaption und Parametrierung der entstandenen Roboterprogramme notwendigerweise in direkter Interaktion zwischen Mensch und Roboter geschehen. Dies bedeutet, dass die spezielle Sensorik des Training Centers zur Handlungserfassung nicht zur Verfügung steht. Des Weiteren ist die Umgebung der Ausführung nicht a priori bekannt.
Zum Beispiel kann die Vorführung einer Handlung *Tischdecken* im Training Center mit Datenhandschuhen, magnetfeldbasierter Verfolgung der Hände und Szenenerkennung durch mehrere externe Kameras erfolgt sein. Ergeben sich bei der Ausführung nun Probleme oder Konflikte, die der Roboter nicht selbst lösen kann, so muss der Mensch eingreifen, indem er

erneut Teile der Handlung (nun auf eine andere Weise) vorführt. Dies geschieht nun allerdings im Kontext der Ausführung, also in Interaktion zwischen Mensch und Roboter in natürlicher Umgebung.
Um auch im Ausführungskontext die Handlungen des Menschen erkennen und interpretieren zu können, muss der Roboter also mit Sensorik und Methoden ausgerüstet sein, die es ihm erlauben, menschliche Handlungen zu erkennen. Sowohl das begrenzte Sichtfeld als auch die verfügbare Rechenkapazität des Roboters bedeuten hierbei eine erhebliche Einschränkung.
Es müssen deswegen Sensoren und Mechanismen gefunden werden, die auch unter den gegebenen Einschränkungen die Beobachtung menschlicher Handlungen erlauben.
Die Beobachtung menschlicher Bewegungen zur Handlungsanalyse kann zusätzlich auch zur Unterstützung einer natürlichen Interaktion dienen. Die Beobachtung und das Verständnis menschlicher Gesten, Körperhaltung und anderer Aktivitäten durch das Robotersystem als zusätzlicher Kommunikationskanal können sowohl für den Dialog mehrdeutige Situationen auflösen als auch zur direkten Instruierung und Kommandierung dienen. Dazu müssen die beobachteten menschlichen Bewegungen entsprechend interpretiert und klassifiziert werden.

1.2 Zielsetzung und Beitrag der Arbeit

Die Zielsetzung der vorliegenden Arbeit ist es, gelerntes, abstraktes Handlungswissen auf einen Roboter zu übertragen, es dort zu parametrieren und auszuführen. Dabei spielt die Beobachtung menschlicher Bewegungen und deren Klassifikation für die Interaktion eine entscheidende Rolle im Ausführungskontext.

Es werden die folgenden Aspekte betrachtet:

1. Für den Roboter muss eine Handlungsbeschreibung gefunden werden, die es ermöglicht, flexibel und adaptiv sowohl vordefinierte als auch gelernte Handlungsfolgen (Problemlösungswissen) auszuführen. Diese Beschreibung sollte zusätzlich die Möglichkeit bieten, leicht verständlich und veränderbar zu sein, um einen einfachen Entwurf bzw. eine einfache Adaption zu ermöglichen.

2. Die durch *Programmieren durch Vormachen* gelernten Makrooperatoren sollen in diese ausführbare Beschreibung überführt werden können. Dazu müssen die Einschränkungen des jeweiligen Zielsystems sowie die Definition und die Eigenschaften der Zielbeschreibung in Betracht gezogen werden.

3. Eine so entstandene Aufgabenbeschreibung für das Zielsystem soll in Interaktion zwischen Roboter und Mensch nachträglich adaptierbar sein. Dafür sollen Mechanismen zur effizienten Beobachtung des Menschen allein mit Robotersensorik im Ausführungskontext geschaffen werden.

4. Die Information aus der Beobachtung des Menschen soll auch als zusätzlicher Interaktionskanal zur Kommandierung verwendet werden. Dazu ist es notwendig, die beobachteten Bewegungen des Menschen in verschiedene Aktivitäten und Gesten zu klassifizieren.

Die vorliegende Arbeit gliedert sich anhand dieser Aspekte. Kapitel 1.3 gibt einen kurzen Überblick über den Aufbau.

1.3 Aufbau der Arbeit

Das folgende Kapitel 2 gibt zunächst einen Überblick über grundlegende und aktuelle Arbeiten auf den behandelten Gebieten. Hier werden sowohl die Randbedingungen sowie die Vor- und Nachteile der jeweiligen Systeme diskutiert und gegenübergestellt. Schließlich werden daraus die Anforderungen und Voraussetzungen für die vorgestellte Arbeit abgeleitet. Im Einzelnen werden die Themen der verwendeten Architektur sowie der Handlungsbeschreibung diskutiert. Bestehende Ansätze zur Abbildung abstrakten Handlungswissens auf dedizierte Robotersysteme werden vorgestellt, sowie existierende Methoden zur Beobachtung und Interpretation menschlicher Bewegungen. Der aus diesen Betrachtungen identifizierte Handlungsbedarf wird zusammengefasst und dient zur weiteren Motivation dieser Arbeit.
Kapitel 3 stellt den grundlegenden Ansatz dieser Arbeit vor. Dabei wird auf die in Kapitel 2 getroffenen Feststellungen zurückgegriffen. Die Grundlagen und der Prozess des *Programmierens durch Vormachen* sowie Inhalte und Aufbau der resultierenden Handlungsbeschreibung werden vorgestellt. Des Weiteren wird das zur Ausführung genutzte Robotersystem Albert II beschrieben.
Die Möglichkeiten und Einschränkungen eines Systems ergeben sich zum einen durch die verwendeten Verfahren, vor allem aber durch die zu Grunde gelegten Modelle. Aus diesem Grund wird dieses Modellwissen gesondert beschrieben und dargestellt. Kapitel 4 stellt die verwendete Wissensrepräsentation sowie die Modelle zur Darstellung und Ausführung von Handlungswissen vor. Die zur Beobachtung und Interpretation des Menschen verwendete geometrische Modellierung des Körpers sowie das zur Klassifikation verwendete Modell menschlicher Aktivitäten wird hier erläutert.
Um den Prozess von der Vorführung bis zur Ausführung von Handlungen zu schließen, wird eine Abbildung abstrakten Handlungswissens auf ausführbare Roboterprogramme definiert. Kapitel 5 erläutert diesen Prozess der Abbildung von Makrooperatoren auf Roboterprogramme. Hierbei wird insbesondere das benötigte Hintergrundwissen über das ausführende System untersucht und explizit definiert.
Die Mechanismen des Ausführungskontexts werden in Kapitel 6 dargestellt. Hier wird vor allem die Ausführung von Roboterprogrammen detailliert betrachtet. Einen wichtigen Aspekt bildet dabei die Auswahl von Handlungen zur Ausführungszeit; diese Entscheidungskomponente basiert auf einer Modellierung des Kontexts und wählt regelbasiert zu aktivierende Roboterprogramme aus.
Kapitel 7 stellt verwendete Komponenten für die Interaktion mit dem Benutzer dar. Dies ist zum einen die Darstellung des Roboterzustands für den Benutzer, um eine Ausgabe und Rückkopplung über aktuelle Zustände und Aktionen zu geben. Zum anderen die Beobachtungskomponente zur Verfolgung der Körperpose des Menschen und zur Interpretation der so beobachteten Bewegungen zur Kommandierung und Parametrierung.
Eine ausführliche Evaluation der vorgestellten Verfahren wird in Kapitel 8 gegeben. Hier werden Experimente sowohl aus der Erzeugung und Ausführung von Handlungswissen als auch für die Beobachtungskomponenten vorgestellt und bewertet.
Eine Zusammenfassung und einen Ausblick auf mögliche zukünftige Erweiterungen gibt schließlich Kapitel 9.

Kapitel 2

Stand der Forschung

Die Robotik, und insbesondere die Servicerobotik, ist eine junge Wissenschaft, die in den vergangenen Jahrzehnten enorme Fortschritte gemacht hat. Durch diese hohe Dynamik und durch die Tatsache, dass zur Robotik viele verschiedene Forschungsrichtungen und Wissenschaften beitragen, ergibt sich eine große Zahl von unterschiedlichen Ansätzen und Paradigmen in der aktuellen Forschung.
Im folgenden Kapitel werden die für diese Arbeit relevanten existierenden Ansätze untersucht, klassifiziert und hinsichtlich ihrer Verwendbarkeit für den vorliegenden Kontext bewertet.

2.1 Fokus und Rahmen

Die vorliegende Arbeit untersucht den Ausführungskontext komplexer Programme im Rahmen der Servicerobotik. Schwerpunkte sind dabei die Handlungsbeschreibung des Roboters, deren Ausführung und Parametrierung sowie die Beobachtung des Menschen hierzu.
Im Folgenden werden die entsprechenden Bereiche der aktuellen Forschung vorgestellt. Dabei werden für den Ausführungskontext sowohl verschiedene Architekturen untersucht, als auch existierende Repräsentationen für Handlungswissen beleuchtet und bewertet. Die explizite Abbildung von abstraktem Handlungswissen auf dedizierte Robotersysteme hat dabei bisher wenig Aufmerksamkeit erhalten. Dieser Teilbereich wurde bisher hauptsächlich im Rahmen des *Lernen durch Imitation* (engl.: *Imitation learning*) behandelt; dies wird in Kapitel 2.4 näher untersucht. Die Akquisition von abstraktem Handlungswissen soll dabei als gegeben vorausgesetzt werden, für eine ausführliche Diskussion sei hier auf die Arbeiten von Friedrich [Friedrich 98], Rogalla [Rogalla 02], Ehrenmann [Ehrenmann 02] und Zöllner [Zöllner 05b] im Rahmen des *Programmieren durch Vormachen* verwiesen.
Existierende Arbeiten auf dem Gebiet der Beobachtung und Interpretation des Menschen und seiner Bewegungen durch den Roboter zum Zwecke der Interaktion werden in Kapitel 2.5 vorgestellt und hinsichtlich ihrer Anwendbarkeit bewertet.

2.2 Architekturen für Robotersysteme

Die Software-Architektur bildet gewissermaßen das Rückgrat für die Informationsverarbeitung eines Robotersystems. Die richtige Wahl der Systemarchitektur kann Entwurf und Auf-

bau des Systems unterstützen und fördern; umgekehrt kann eine falsche Wahl entsprechend den Systementwurf unnötig verkomplizieren und verschiedene Konstrukte sogar unmöglich machen [Coste-Maniere 00].
Eine Architektur hat die Aufgabe, das Zusammenwirken verschiedener Komponenten zu definieren und zu regeln. Die größte Herausforderung der Robotik ist genau diese Integration verschiedener Disziplinen: Wie kein anderes Thema bringt die Robotik Mensch-Maschine-Interaktion, Mechatronik, Elektronik, Bildverarbeitung, Regelung, Modellierung, Künstliche Intelligenz und andere Gebiete zusammen. Die einem System zu Grunde liegende Architektur hat also fundamentale Bedeutung für die Wirkungsweise, Effektivität und Erweiterbarkeit.
Deshalb hat die Frage nach einer Architektur für flexible Robotersysteme in den vergangenen 20 Jahren in der Forschung zunehmend an Bedeutung gewonnen. Wurde die Architektur anfangs noch als "Verbindung" oder "Kleber" zwischen einzelnen Komponenten betrachtet, hat sich inzwischen daraus eine eigene Forschungsrichtung entwickelt, die sich ausschließlich mit der Suche nach einer optimalen Architektur beschäftigt. Eng verknüpft ist diese Frage mit der Frage nach den verwendeten Modellen und Repräsentationen des Roboters, da die verwendete Architektur diese kodieren und den Zugriff auf das vorhandene Wissen erlauben muss. Kapitel 2.3 geht dem nach.
Existierende Architekturen für Servicerobotersysteme können nach ihrem inneren Aufbau und ihrer Motivation klassifiziert werden. Die gängigste Unterteilung, die auch hier Verwendung finden soll, teilt Architekturen ein in *deliberative*, *verhaltensbasierte* und *hybride* Architekturen. Diese Ansätze werden im Folgenden erläutert und mit Beispielen belegt. Weiterhin können *monolithische* und *Multi-Agenten-Systeme* unterschieden werden. Diese Differenzierung zielt vor allem auf die Erweiterbarkeit des Systems und die Ersetzbarkeit von Komponenten:

- Monolithische Systeme besitzen keine Trennung in einzelne Komponenten. Die Teile des Gesamtsystems sind stark miteinander verwoben und untereinander vernetzt, so dass wenig Einschränkungen bei der Kommunikation und der wechselseitigen Zugriff der Teilfunktionen gemacht werden.

- Multi-Agenten-Systeme dagegen bestehen aus Teilkomponenten mit klar definierten Schnittstellen, Kommunikationswegen und Hierarchien der Einzelkomponenten. Ein solcher Entwurf fördert die Erweiterbarkeit des Systems sowie die Ersetzbarkeit von Teilsystemen.

Seit Mitte der 90er Jahre wurden zusätzlich Ideen der Kognitionswissenschaften aufgegriffen. Der Entwurf von Architekturen, die hierdurch inspiriert sind, führte zu dem Begriff der *kognitiven Architektur*. Diese werden anschließend in Kapitel 2.2.4 vorgestellt und diskutiert. Um bestehende Ansätze bewerten zu können, sollen zunächst die Anforderungen definiert werden.

2.2.1 Anforderungen

Durch die hohe Komplexität heutiger Robotersysteme gibt es eine Reihe an Anforderungen an den Roboter, die Steuerung und damit auch an die Architektur, die teilweise sogar gegensätzlich sind [Coste-Maniere 00].

Zielgerichtetheit: Die Architektur muss eine Repräsentation und Verfolgung von expliziten Zielen ermöglichen. Diese Ziele können abstrakt und aus Teilzielen zusammengesetzt sein.

Interaktion mit der Umgebung: Der Roboter muss in der Lage sein, mit seiner Umwelt zu interagieren. Dies beinhaltet sowohl Interaktion mit Gegenständen als auch mit Menschen. Die Herausforderung ist dabei, die Systemdynamik mit der Interaktion und den Sicherheitsaspekten zu vereinen.

Teilweise unbekannte Umwelt: Das System soll in der Lage sein, mit unvollständigem Wissen über die Umwelt trotzdem optimal zu entscheiden und zu agieren. Dies betrifft sowohl die symbolische als auch die subsymbolische Ebene.

Reaktivität: Aufgrund der teilweise unbekannten Umwelt und deren stochastischen Verhaltens müssen Roboter in der Lage sein, auf unerwartete Änderungen sofort und konsistent zu reagieren.

Sicherheit: Viele Roboter gehören zu den kritischen Systemen. Dies bedeutet, dass Fehler während der Ausführung entscheidende Konsequenzen haben können und die System- und Ausfallsicherheit sowie die Lesbarkeit und Transparenz eine große Rolle spielen.

Selbst diese allgemein gehaltenen Anforderungen sind teilweise gegensätzlicher Natur. So fordert eine hohe Reaktivität kleine und lokale Regelungs- und Entscheidungsschleifen; dies widerspricht der Forderung nach Zielgerichtetheit, die langfristige Ziele global zu verfolgen sucht.
Die Schwerpunkte verschieben sich deshalb abhängig von der konkreten Anwendung, wie im Folgenden zu sehen sein wird.

2.2.2 Deliberative Architekturen

Deliberative oder auch wissensbasierte Architekturen folgen dem klassischen *Erfassen–Planen–Agieren* (engl.: *Sense–Plan–Act*, *SPA*) Schema [Nilsson 80]. Dieses schon lange bekannte Konzept definiert einen unidirektionalen Informationsfluss von den Erfassungskomponenten in ein Weltmodell. Dieses wird von einem Planungsmodell verwendet, um Aktionen zu planen. Dieser Plan wird von einer Ausführungseinheit umgesetzt.
Die Architektur wird dabei oft in diese Komponenten unterteilt [Albus 88]. Charakteristisch ist dabei, dass die Kommunikation zwischen verschiedenen Abstraktionsebenen nur vertikal verläuft. Es wird also die Kommunikation zwischen verschiedenen Elementen gleicher Abstraktion unterbunden und damit die Bedeutung der Planungskomponente hervorgehoben. Diese Art der Robotersteuerung ähnelt stark einem Expertensystem.

2.2.3 Verhaltensbasierte Architekturen

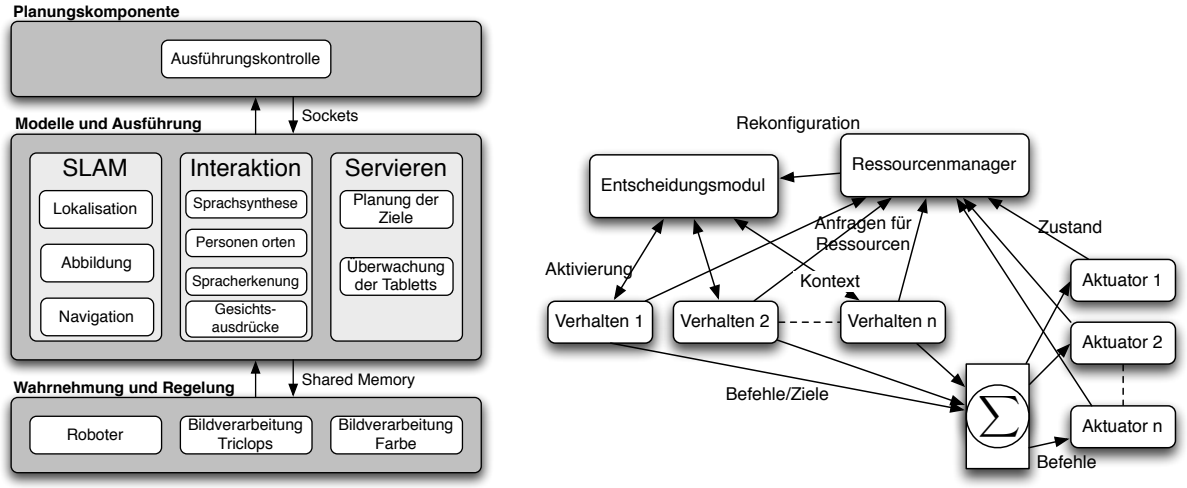
Im Gegensatz zum deliberativen Ansatz steht der verhaltensbasierte Ansatz. Der ursprünglich von Brooks [Brooks 86] vorgeschlagene *Subsumptions-Ansatz* ist der bekannteste Vertreter.
Der rein verhaltensbasierte Ansatz besitzt keine explizite Repräsentation von Plänen symbolischer Natur; genau genommen gibt es keine Symbole (siehe [Brooks 90]).

Die Architektur wird dabei nicht aus Blöcken aufgebaut, die hierarchisch geordnet sind; vielmehr ist sie zusammengesetzt aus einzelnen, kleinen, autonomen Modulen, die über Eingangs- und Ausgangssignale verbunden sind. Diese werden zusammengefasst zu komplexen *Verhalten*, deren Intelligenz nicht als expliziter Plan, sondern als Verschaltung von Einzelkomponenten repräsentiert ist. Die einzige Möglichkeit, auf diese Verhalten Einfluss zu nehmen, um übergeordnete Pläne zu verfolgen, besteht in der Möglichkeit, Signale an Ein- oder Ausgängen von Modulen zu überschreiben.
Mit dem verhaltensbasierten Ansatz sind beachtliche Erfolge erzielt worden. So zeigt Brooks z.B. mit dem humanoiden Roboter *Cog* verschiedene komplexe Verhalten [Brooks 99]. Albiez et al. zeigen in [Albiez 02] verschiedene Experimente mit einem erweiterten Verhaltensnetzwerk zur Regelung des Laufens einer vierbeinigen Laufmaschine.
Allerdings zeigen diese Beispiele auch die Grenzen der verhaltensbasierten Architekturen auf. Je größer Verhaltensnetzwerke werden, umso fehleranfälliger werden diese. Konnten verhaltensbasierte Architekturen für kollisionfreie Navigation noch sehr gut verwendet werden, so ist z.B. von dem Roboter *Herbert*, einem späteren Projekt von R. A. Brooks [Connell 89], keine vollständige, fehlerfreie Ausführung der Gesamthandlung bekannt (siehe [Gat 97b]). Dieser Roboter sollte Getränkedosen auffinden und einsammeln. Ein weiterer Nachteil ist, dass ein Verhaltensnetzwerk keine klare modulare Struktur aufweist und empfindlich auf Änderungen reagiert [Hartley 91]. Schon kleine Änderungen an einzelnen Verhalten oder am Roboter selbst bedeuten, dass das gesamte Netzwerk neu aufgebaut werden muss. Im rein verhaltensbasierten Ansatz fehlt also die Möglichkeit der Modularisierung und der Abstraktion.
Um verhaltensbasierte Architekturen auch auf komplexere Probleme mit temporalen Nebenbedingungen und Sequenzen sowie hierarchischen Aufgabenbeschreibungen verwenden zu können, wurden von Matarić et al. sogenannte *Abstrakte Verhalten* eingeführt (siehe [Nicolescu 02]). Abstrakte Verhalten beinhalten wiederum Teile der klassischen Künstlichen Intelligenz (KI) und sind somit in der Lage, auch komplexe temporale Sequenzen darzustellen. Dieser Ansatz stellt also keine klassische verhaltensbasierte Architektur mehr dar, sondern verbindet die Vorteile verhaltensbasierter Steuerung mit denen eines deliberativen Ansatzes, und gehört somit eigentlich schon zu den hybriden Architekturen.

2.2.4 Hybride Architekturen

Um den in Kapitel 2.2.1 genannten Anforderungen nach Reaktivität auf der einen und Deliberation auf der anderen Seite entsprechen zu können, wurden die hybriden Architekturen entwickelt. Diese kombinieren die Idee der Verhaltensnetzwerke – kleinen lokalen Regelschleifen – mit den Vorteilen der deliberativen Architektur. Auf diese Weise entstehen Architekturen, die sowohl horizontal (Verhalten) als auch vertikal (Deliberation) stark vernetzt sind. In den letzten 10 bis 15 Jahren hat sich dieser Ansatz bewährt, und die meisten Architekturen sind inzwischen hybrid aufgebaut.
Ähnlich zu [Nicolescu 02] (siehe Kapitel 2.2.3) liegt bei [Elinas 02] und [Simmons 02] der Fokus auf einer verhaltensbasierten Architektur, die um eine deliberative Planungsschicht erweitert wurde (siehe Abbildung 2.1(a)). Die untere Schicht besteht aus den notwendigen Funktionen zur Ansteuerung der Aktorik und Sensorik. Die mittlere Schicht stellt eine klassische verhaltensbasierte Struktur dar, mit unabhängigen Prozessen für unterschiedliche Aufgaben. Lokalisiert sind hier Verhalten für Navigation, das Auffinden von Menschen, oder

die Überwachung von Zuständen während der Ausführung. Die obere Schicht beinhaltet die Aktionsplanungskomponente (engl.: *planning*) sowie die symbolische Ausführungskomponente (engl.: *supervisor*). Dieses Modul löst in Abhängigkeit von der geplanten Aktionssequenz verschiedene Verhalten mit den entsprechenden Parametern aus.



(a) (b)

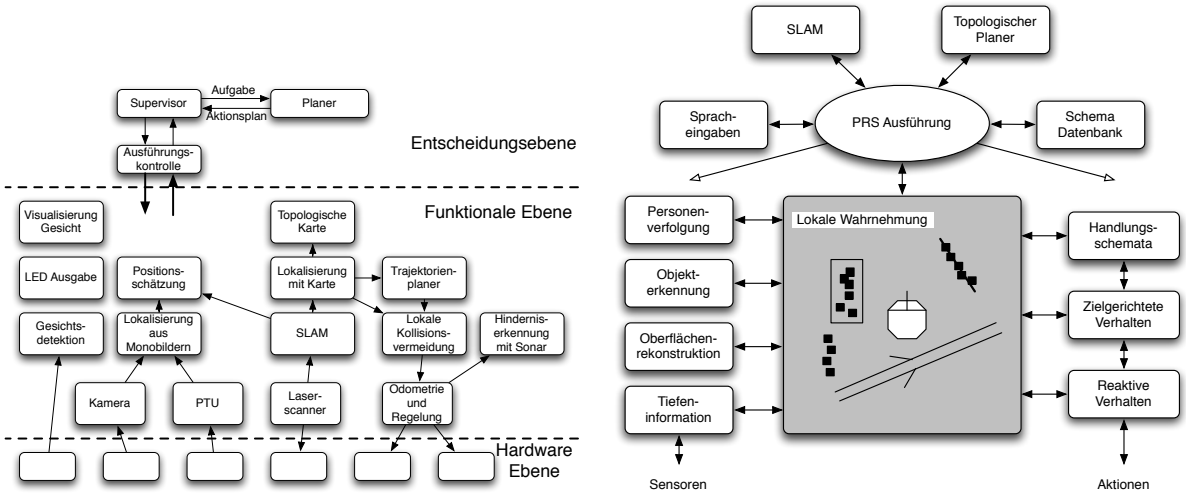
Abbildung 2.1: (a) Architektur des Roboters *Jose*, nach [Elinas 02]. (b) Steuerungsarchitektur *OSCAR*, nach [Blum 03]

Ein ähnlicher Ansatz wird auch in [Blum 03] beschrieben (siehe Abbildung 2.1(b)). Hier wird ein modularer Ansatz verfolgt, in dem die Komponenten über ein Netzwerkprotokoll verbunden sind. Dies stellt ein Multi-Agenten-System dar.

Die *LAAS*-Architektur [Alami 00, Alami 98] ist in drei Schichten aufgebaut (siehe Abbildung 2.2(a)). Die untere Schicht besteht aus Verhaltensnetzwerken, die mit dem Werkzeug GenoM (siehe [Mallet 02, Fleury 97]) aus Modulen aufgebaut werden. Diese werden aktiviert, parametriert und gesteuert aus der mittleren Schicht, die die Ausführung überwacht (engl.: *Supervisor*). Die Sequenz der Aktionen und deren Parametrierung wird von der oberen Schicht zur Verfügung gestellt (engl.: *Planner*).

Einen ähnlichen Aufbau hat die *Saphira*-Architektur [Konolige 97] (Abbildung 2.2(b)). In der unteren Schicht sind die Verhalten dabei hierarchisch geordnet, woraus sich eine Struktur ähnlich zu NASREM (siehe [Albus 89]) ergibt. Die mittlere Schicht hat die gleiche Funktion wie in der LAAS-Architektur, und die obere Schicht besteht aus einem topologischen Planer.

Ein ebenfalls dreischichtiger Ansatz wurde in der *3T*-Architektur verwendet [Bonasso 97]. Die drei Ebenen werden hier von unten nach oben als *Fähigkeiten*, *Sequenzierung* und *Planung* (engl.: *Skills, Sequencing, Deliberation/Planning*) bezeichnet (siehe Abbildung 2.3). Anwendungen umfassen hier mobile Roboter mit Erkennungs- und Manipulationsfähigkeiten, sowie Navigation durch größere Gebäude.



(a) (b)

Abbildung 2.2: Die hybriden Architekturen *LAAS* (a) und *Saphira* (b). Besonders bei der der LAAS-Architektur sind die drei Schichten der Abstraktion gut zu erkennen: Ein Planer erzeugt Aktionssequenzen, die vom *Supervisor* ausgeführt werden. Dieser steuert dazu die Regelungsschicht an. Saphira (b) besitzt den gleichen Aufbau: PRS bildet den Supervisor, und die Pläne werden von höheren Instanzen geliefert.

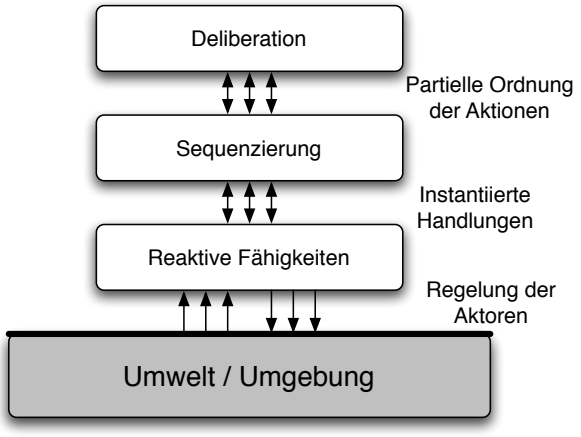


Abbildung 2.3: Der Aufbau der Architektur *3T*. Die drei klassischen Schichten der hybriden Architekturen sind zu erkennen.

2.2.5 Kognitive Architekturen

Im Folgenden werden zunächst Kriterien betrachtet, nach denen eine Architektur als *kognitiv* klassifiziert werden kann.

Vorbetrachtungen

Der Begriff *Kognition* leitet sich ab von lateinisch *cognoscere* (erkennen). Im Allgemeinen versteht man unter Kognition die "Prozesse und Strukturen, die mit dem Wahrnehmen und Erkennen zusammenhängen (Denken, Erinnerung, Vorstellen, Lernen, Planen und anderes)" [Brockhaus 90]. *Kognitiv* bedeutet "die Erkenntnis betreffend".
Der Ausdruck *kognitive Architektur* lässt verschiedene Interpretationen zu, die hier zunächst diskutiert werden müssen.

1. Der *Entwurf* der Architektur ist kognitiv. Damit werden keine Aussagen über die Architektur selbst gemacht, sondern lediglich über den Entwurfsprozess.

2. Die *Struktur* der Architektur gleicht der eines kognitiven Systems (beispielsweise des menschlichen Gehirns). Damit wird eine Aussage über den Aufbau getroffen, nicht über die Fähigkeiten des Systems.

3. Das *System* selbst, das durch die Architektur beschrieben wird, besitzt kognitive Fähigkeiten. In diesem Fall kann das resultierende System als "Blackbox" mit erwiesenermaßen kognitiven Systemen (z.B. Mensch) verglichen werden.

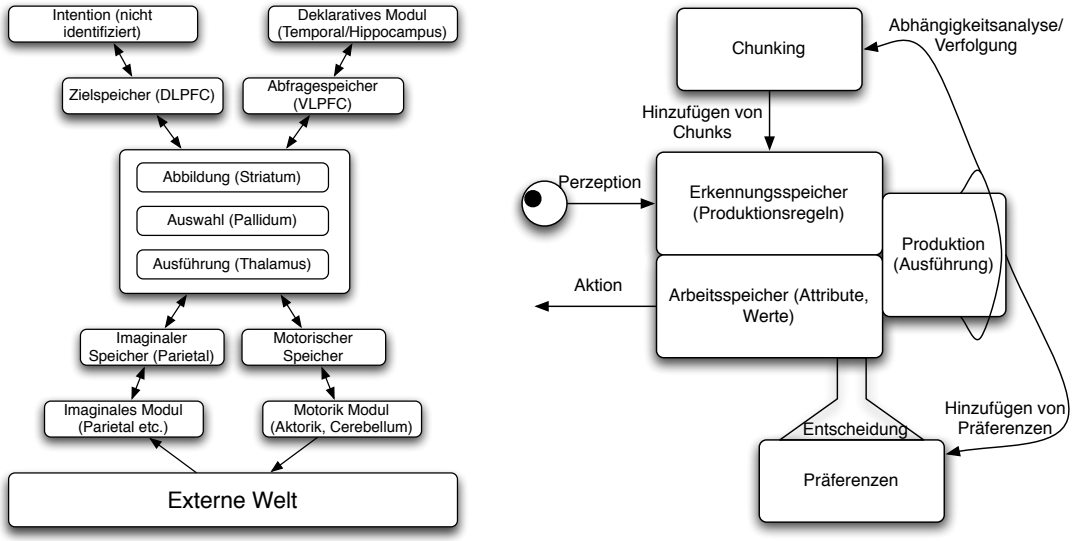
In der Literatur findet sich keine einheitliche Definition für kognitive Architekturen. Ausgehend von der Menge der so bezeichneten Systeme kann man allerdings feststellen: Die meisten Autoren verwenden eine Definition nach Definition 2 (siehe [Wurm 03, Lewis 99b, Anderson 04]). Es werden demnach diejenigen Ansätze als kognitiv bezeichnet, deren Struktur von der Struktur z.B. des menschlichen Gehirns inspiriert ist. Allerdings finden sich auch Autoren, die nach den resultierenden Fähigkeiten eines Systems werten (Definition 3). Dies findet sich beispielsweise in [Langley 05, Burghart 05].
In [Langley 06] wird argumentiert, dass das Ziel eine Abbildung kognitiver Fähigkeiten in das System ist, und deshalb oft die Struktur nach dem einzigen uns bekannten solchen System, dem menschlichen Gehirn, gewählt wird. Dieser Definition folgt auch die vorliegende Arbeit; Ziel ist es, kognitive Fähigkeiten in ein System zu integrieren (Definition 3). Die Adoption einer bestimmten Struktur (nach Definition 2) ist unter Umständen sinnvoll, aber nicht hinreichend.
Es lässt sich also feststellen, dass keine eindeutige Definition existiert. Auch sind kognitive Architekturen nicht im Unterschied zu den vorangegangenen Klassen von Architekturen zu betrachten. Eine kognitive Architektur kann entsprechend deliberativ, verhaltensbasiert oder hybrid sein. Auch über die Modularität wird keine Aussage getroffen.

Existierende Architekturen

Zwei Architekturen werden in der Literatur allgemein als kognitiv bezeichnet. Dies ist zum einen die Architektur ACT-R, die ursprünglich von John R. Anderson vorgeschlagen wurde [Anderson 98, Anderson 04], und zum anderen SOAR, urspünglich von Allen Newell et al. [Laird 87] vorgestellt.

Beide Architekturen bestehen im Kern aus Produktionsregelsystemen zur Informationsverarbeitung. Sie gehen also von einer symbolischen Informationsverarbeitung aus. Zudem enthalten sie bestimmte Mechanismen zur Repräsentation von Wissen sowie zur Kommunikation. Die Struktur ist dabei abgeleitet von der Struktur des menschlichen Gehirns.

Adaptive control of thought-rational (ACT-R) Die Struktur von ACT-R orientiert sich sehr streng an der Struktur des menschlichen Gehirns. Abbildung 2.4(a) gibt diese wieder und zeigt die direkte Assoziation der ACT-R Komponenten mit verschiedenen Regionen im Gehirn: Z.B. wird die zentrale Schlussfolgerungseinheit mit den Basalganglien im menschlichen Gehirn gleichgesetzt. Genauso existiert für jede weitere Komponente der Architektur ein Pendant im menschlichen Gehirn.



(a) (b)

Abbildung 2.4: Die Architektur von ACT-R (a), nach [Anderson 04], und (b) die SOAR Architektur, nach [Laird 87]

Bei der Wissensrepräsentation wird zwischen deklarativem und prozeduralem Wissen unterschieden. Deklaratives Wissen beinhaltet das Wissen über die Welt sowie eigene Ziele; es ist in unabhängigen Einheiten (engl.: *Chunks*) kodiert. Beziehungen zwischen einzelnen Einheiten können aufgebaut werden, indem diese sich gegenseitig referenzieren. Prozedurales Wissen besteht aus den für die Produktion verwendeten Regeln. Es ist in einem separaten Regelspeicher abgelegt.

SOAR Die Struktur von SOAR [The SOAR group 06, Lewis 99a, Laird 87] orientiert sich weniger stark an der Struktur des menschlichen Gehirns. Sowohl die internen Repräsentationen als auch die Architektur sind stark an klassische Ansätze aus der Künstlichen Intelligenz angelehnt (siehe Abbildung 2.4(b)).
Die Grundannahme eines symbolverarbeitenden Systems findet sich auch in SOAR. Ähnlich einer klassischen Architektur ist das Wissen in Regeln und Arbeitsspeicher getrennt, wobei letzterer sowohl das Faktenwissen als auch die momentanen Ziele beinhaltet.

Regeln werden aufgrund des bekannten Wissens gewählt, wobei bei der Auswahl jeweils alle Regeln in Betracht gezogen werden, deren Vorbedingung erfüllt ist. Aufgrund weiterer Regeln werden diese bewertet und letztendlich wird der beste Operator gewählt. SOAR trifft seine Entscheidungen also immer auf Basis aller bekannten Informationen.
Die Erzeugung von Unterzielen wird in SOAR nicht von Regeln, sondern von einer eigenen Systemeinheit vorgenommen. Mit Hilfe der *Mittel-Ziel-Analyse* werden Teilziele gefunden, die zur Lösung des aktuellen Problems beitragen. Die Problemlösung wird also als Suche in einem Problemraum betrachtet und modelliert. Dieser ist gegeben durch Anfangs- und Endzustand (oder -zustände), sowie die Menge der möglichen Operatoren.

Tabelle 2.1 gibt einen Überblick über die wichtigsten Unterschiede zwischen ACT-R und SOAR. Dabei wird besonders auf die Darstellung und Modellierung von Wissen eingegangen.

2.2.6 Bewertung

Die vorgestellten Architekturen repräsentieren einige Beispiele aktueller Ansätze und implementierter Systeme. Eine Bewertung dieser Ansätze kann unter zwei unterschiedlichen Aspekten erfolgen: Zum einen aus kognitionswissenschaftlicher Sicht, zum anderen auf Basis eines technologischen Standpunktes.
Bewertet man aus kognitionswissenschaftlicher Sicht den Ansatz selbst, so sind die kognitiven Architekturen ACT-R und SOAR als am fortschrittlichsten anzusehen. Ihre Struktur bietet keinen rein technischen Ansatz, sondern leitet sich ab von Erkenntnissen über die Struktur kognitiver Prozesse beim Menschen. Davon verspricht man sich eine Eigendynamik und Erweiterbarkeit der Systeme, vor allem aber eine höhere Generalisierungsfähigkeit.
Im Gegensatz dazu stehen die klassischen hierarchischen Architekturen. Diese meist mehrschichtigen Ansätze sind weit verbreitet und weisen in zahlreichen Anwendungen ihre Robustheit und Verwendbarkeit auf. Speziell die hybriden Architekturen, die die Vorteile verhaltensbasierter und deliberativer Architekturen vereinen, sind hier zu nennen.
Üblicherweise werden drei Hierarchieebenen realisiert: Unten die Regelungsebene, darüber die Sequenzierung oder Steuerung, und oben eine Planungsebene. Aus technologischer Sicht ist dieses Konzept sinnvoll und tragfähig. Die Aufgaben können klar einzelnen Modulen zugeordnet werden, Schnittstellen sind klein und übersichtlich. Alle Abhängigkeiten eines solchen technischen Systems können modelliert werden.
Abbildung 2.5 ordnet die in diesem Kapitel betrachteten Architekturen nach den genannten Kriterien *deliberativ/verhaltensbasiert* sowie *monolithisch/Multi-Agenten-Struktur*, und markiert die als kognitiv anerkannten Architekturen.
Die abschließende Wertung der einzelnen Architekturen entscheidet sich an der Frage, ob ein Serviceroboter mit kognitiven Fähigkeiten, die denen eines Menschen ähnlich sind, dazu auch eine dem Menschen ähnliche Verarbeitungsstruktur aufweisen soll.
Diese Frage ist in der Literatur nach wie vor umstritten und nicht eindeutig geklärt. Mit einer kognitiven Architektur Prozesse des Menschen nachbilden zu wollen, erscheint einerseits sinnvoll; dem kann entgegen gehalten werden, dass Robotern fundamental andere Funktionsmechanismen zu Grunde liegen. Dies betrifft sowohl die Sensorik und Aktorik als auch die Methoden der Informationsverarbeitung selbst. Unterschiedliche Repräsentationen und Funktionsweisen im Kern können also auch andere Verarbeitungsmechanismen verlangen.

ACT-R	SOAR
Annahmen	
Symbolverarbeitendes System Struktur des menschlichen Gehirns als strenges Vorbild	Symbolverarbeitendes System Lose Orientierung am menschlichen Gehirn
Darstellung der Ziele	
Zielstapel Unterziele durch Regeln generiert	Zielstapel Unterziele vom System generiert
Wissensrepräsentation	
Deklarativer und prozeduraler Langzeitspeicher Prozedurales Wissen: Regeln Deklaratives Wissen: Netzwerke von Chunks Arbeitsspeicher Teil des deklarativen Speichers	Prozeduraler Langzeitspeicher, Arbeitsspeicher flüchtig Prozedurales Wissen: Regeln Deklaratives Wissen: Regeln Separater Arbeitsspeicher
Auflösung von Konflikten	
Psychologisch motiviert	Wissensbasiert
Lernen	
Lernen prozeduralen Wissens durch Beispiele Speicherung von abgearbeiteten Zielen als Chunks	Zusammenfassung von Unterzielen zu neuen Regeln

Tabelle 2.1: Die wichtigsten Merkmale von ACT-R und SOAR im Vergleich. Siehe [Anderson 04, Laird 87, Wurm 03]

Wird also eine dem Menschen ähnliche Verarbeitungsstruktur angestrebt, sind kognitive Architekturen vorzuziehen; gilt dies nicht, so kann eine klassische dreischichtige hybride Architektur verwendet werden.

2.3 Repräsentation von Aufgabenwissen

Serviceroboter sollen in flexiblen, dynamischen Umgebungen arbeiten sowie neues Handlungswissen hinzulernen können. Diese beiden Bedingungen stellen hohe Forderungen an die Handlungsbeschreibung des Roboters. Im Gegensatz zu einem Industrieroboter, dessen Einsatzumgebung und Aufgabe klar strukturiert sind, muss ein Serviceroboter mit sich

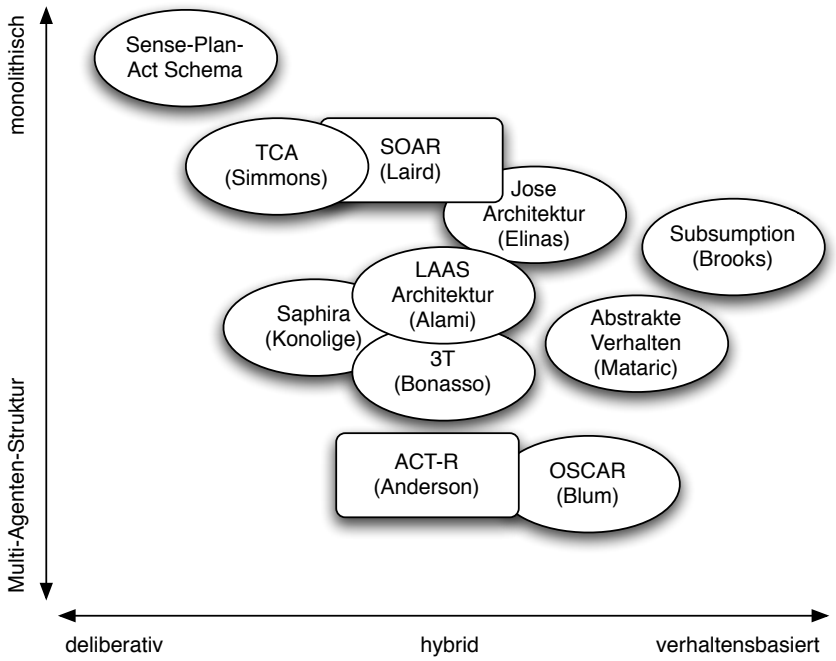


Abbildung 2.5: Einordnung der genannten Architekturen. Die in der Literatur als *kognitive Architekturen* bezeichneten Ansätze sind eckig (siehe Kapitel 2.2.5), alle anderen oval dargestellt.

ändernden, nicht oder nur schwer vorhersagbaren und teilweise sogar unbeobachtbaren Umweltbedingungen zurecht kommen. Um neues Handlungswissen zu lernen, muss die Struktur des Aufgabenwissens so geartet sein, dass sich neues Wissen entweder symbolischer oder subsymbolischer Art hinzunehmen lässt.

Grundsätzlich lassen sich die untersuchten Ansätze in symbolische und subsymbolische Ansätze unterteilen. Symbolische oder explizite Aufgabenbeschreibungen basieren dabei auf kleinsten Aktionseinheiten, auch *Elementarfähigkeiten* oder *Aktionsprimitive* genannt [Finkemeyer 05]. Die Aufgabenbeschreibung beinhaltet und verknüpft diese Primitive auf eine geeignete Weise. Ein Überblick findet sich in [Pembeci 02]. Subsymbolische oder implizite Beschreibungen besitzen keine symbolische Aktionsabstraktion, sondern erreichen das gewünschte Roboterverhalten durch eine direkte Verknüpfung der vorverarbeiteten Eingangs- und Ausgangsgrößen.

Im Folgenden werden markante Ansätze der symbolischen sowie der subsymbolischen Repräsentation vorgestellt und diskutiert.

2.3.1 Darstellungs- und Planungsansätze aus der Künstlichen Intelligenz

Die klassischen Ansätze, die auf Methoden der Künstlichen Intelligenz (Abkürzung: *KI*) aufbauen, sehen intelligente Systeme als symbolverarbeitende Systeme. Bei einer Repräsentation von Handlungswissen auf Basis von Symbolen werden Aktionsprimitive als kleinste Einheit

verwendet. Anhand dieser abstrakten Elementare können komplexe Handlungen geplant und ausgeführt werden.

STRIPS

Die 1971 von Fikes et al. [Fikes 71] entwickelte Repräsentation für Handlungswissen *STRIPS* kann als erster Ansatz zu expliziten Repräsentationen betrachtet werden.
STRIPS (Abkürzung für: *STanford Research Institute Problem Solver*) verknüpft die Darstellung von Aktionsprimitiven mit einer Repräsentation der Welt sowie einem Algorithmus zur Aktionsplanung.
Aktionsprimitive werden als so genannte *Operatoren* durch ein 4-tupel dargestellt: Ein Operator besitzt einen eindeutigen *Namen*, die notwendigen *Vorbedingungen*, sowie die geltenden Nachbedingungen in Form von *Add-* und *Delete-Listen*. Diese Nachbedingungen können auch als die Effekte des Operators in der Welt bezeichnet werden.
Die Handlungskette zur Lösung eines gegebenen Problems wird nun mit Hilfe des MZAMZK-Algorithmus (Abkürzung für: *Mittel-Ziel-Analyse mit Zielkeller*) bestimmt. Dazu müssen die Menge der Ziele, die Startsituation, sowie die Menge der möglichen Operatoren gegeben sein. Das resultierende Ergebnis stellt den *Plan*, also das erzeugte Handlungswissen dar. Dieser Plan wird dann von dem Roboter umgesetzt. Die Startsituation muss dabei den vollständigen relevanten Weltzustand enthalten. Der erzeugte Plan ist statisch, geht also davon aus, dass der Roboter als einzige Instanz die Welt verändert (Annahme der geschlossenen Welt).
Eine wichtige Eigenschaft des STRIPS Planers sowie der resultierenden Handlungsbeschreibung ist deren *Linearität*. Auf den Planungsalgorithmus bezogen bedeutet dies, dass der Algorithmus mit einem Zielstapel arbeitet und somit immer genau ein Teilziel gleichzeitig bearbeitet. Aufgrund dieser Eigenschaft ist der STRIPS Planer *unvollständig*, es wird also nicht immer ein Plan gefunden, wenn ein solcher existiert. Linearität in der Handlungsbeschreibung zeigt sich darin, dass ein STRIPS Plan immer eine eindeutige Ordnung der Operatoren vorgibt. Es wird also eine Sequenz von Operatoren erzeugt, die unter Umständen unnötige Abhängigkeiten enthält.
Aufgrund der Linearität und der Unvollständigkeit sowie aufgrund der Tatsache, dass STRIPS von einer geschlossenen Welt ausgeht, ist das Verfahren für reale Probleme nicht anwendbar. Ein von STRIPS generierter Plan enthält üblicherweise eine große Anzahl an unnötigen Abhängigkeiten. Die Art der Handlungsbeschreibung jedoch ist grundlegend für viele weitere Handlungsrepräsentationen. Die Darstellung des Planungsproblems durch Anfangs- und Endzustand sowie eine endliche Menge an ausführbaren Primitiven wurde in zahlreichen Planungssystemen übernommen.

Weitere lineare Planungsverfahren

Speziell der Schritt der Planerstellung ist vielfach untersucht und optimiert worden. So kann durch die Einführung von Heuristiken die Suche bedeutend beschleunigt werden. Ein klassisches Beispiel hierfür ist der $A\star$ Algorithmus. Verfahren wie der *FF-Planer* (engl.: *Fast Forward Planer*) [Hoffmann 01] implementieren verschiedene heuristische Kriterien zur Optimierung der Suche. Der jährlich ausgetragene Planungswettbewerb IPC (engl.: *International Planning Competition*) [ICA 06] bietet eine Plattform, die Leistung verschiedener Planungsverfahren zur vergleichen. Hierzu wurde die standardisierte Beschreibungssprache

PDDL (engl.: *Planning Domain Definition Language*) [Ghallab 98] geschaffen, die Planungsdomänen und Probleme vollständig beschreiben kann.

Repräsentation als Sequenz

Eine lineare Darstellung von Handlungswissen kann gewählt werden, um Lösungsstrategien direkt mit Aufgaben oder Szenarien zu verknüpfen. In SOUL (siehe [Jensen 02a, Siegwart 03]) wird eine Darstellung von Handlungswissen gewählt, die Szenarien jeweils Handlungssequenzen zuordnet, die ausgeführt werden sollen. Dies entspricht einem linearen Handlungsmodell (Abbildung 2.6). Die Handlungsketten können über eine Benutzerschnittstelle einfach erstellt und geändert werden.

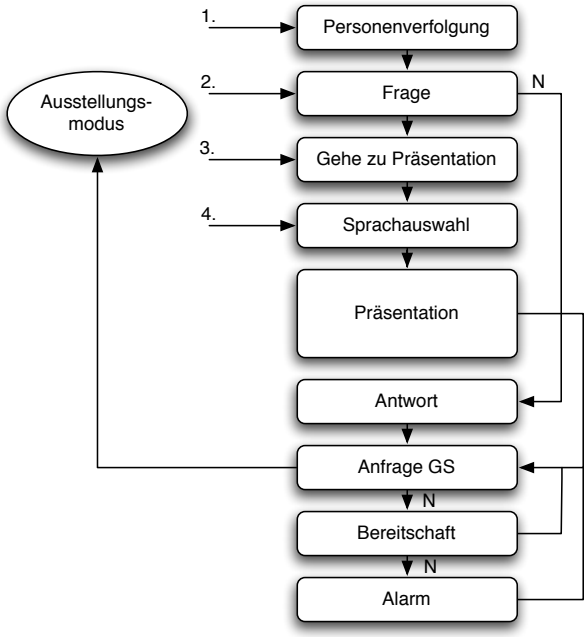


Abbildung 2.6: Ein Szenario aus SOUL, nach [Jensen 02b]

Eine lineare Darstellung von Handlungsketten ist allerdings nicht geeignet, komplexe Handlungsverläufe mit allen potentiellen Ausnahmen und Bedingungen darzustellen. Ebenso fehlt eine Repräsentation von Bedingungen, um Verzweigungen und Sprünge zu realisieren.

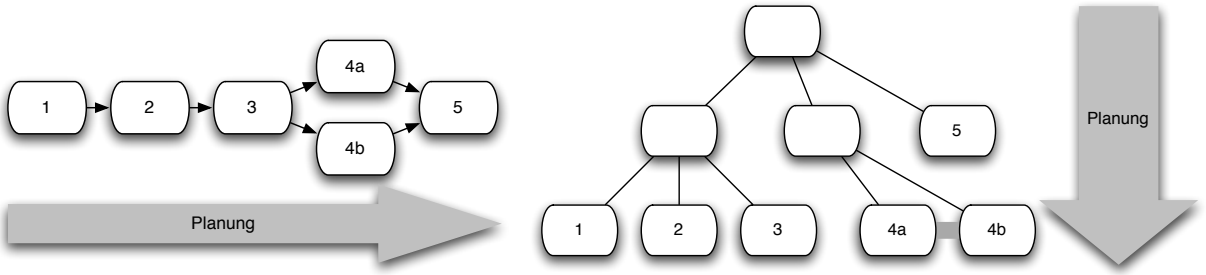
Nichtlineare Repräsentation

Diese Probleme werden durch nichtlineare Repräsentationen gelöst. Ein Plan ist hierbei durch einen Vorranggraphen dargestellt, dessen einzelne Teilzweige wiederum linearen Teilplänen entsprechen. Die Abhängigkeiten zwischen den Elementaraktionen entsprechen den tatsächlichen Abhängigkeiten. Beispiele hierfür sind in [Veloso 95] zu finden. Die Darstellung ist zielorientiert, Operatoren werden also anhand ihrer Vor- und Nachbedingungen charakterisiert.
Eine mehr prozessorientierte Sicht bieten RAPs (engl.: *Reactive Action Packages*) [Firby 01, Pembeci 02]. Hier werden zur Laufzeit verschiedene Prozesse (engl.: *Tasks*) sequentiell, parallel oder unabhängig voneinander gestartet und überwacht. Einzelne Tasks werden

dabei als Aktivitäten behandelt, so dass Systemzustände durch die Menge der laufenden Aktivitäten bestimmt sind. Planungsorientierte Systeme betrachten im Gegensatz dazu Systemzustände jeweils nach Beendigung einzelner Elementarhandlungen.
Der Nachteil der nichtlinearen Darstellung liegt in der Erstellung der Pläne sowie in der Tatsache, dass zur Laufzeit eine eindeutige Ausführungsreihenfolge gewählt werden muss. Die Erstellung nichtlinearer Pläne erfordert aufgrund des sehr viel größeren Suchraums einen größeren Planungsaufwand. Die Pläne während der Ausführung zu sequentialisieren ermöglicht zwar eine Optimierung, verlangt allerdings ein tiefes Verständnis der Optimierungskriterien, der Aufgabe und der Ziele.

2.3.2 Hierarchische Aufgabennetzwerke

Die bisher genannten Aufgabenrepräsentationen beruhen auf Sequenzen und Abhängigkeiten zwischen Aktionen, die nicht notwendigerweise inhaltlich zusammenhängen. Menschen gehen bei Planung und Repräsentation von Handlungen anders vor. Zunächst wird ein grober Plan erstellt, der nach und nach verfeinert wird. Diese Vorgehensweise wird adoptiert von einer Klasse von Ansätzen, die als *hierarchische Aufgabennetzwerke* (engl.: *Hierarchical Task Networks, HTN*) bekannt sind. Dabei wird als Abstraktion eine hierarchische Dekomposition von Aufgaben in Teilaufgaben gewählt. Abbildung 2.7 stellt diesen Unterschied zu STRIPS-ähnlichen Repräsentationen graphisch dar.



(a) (b)

Abbildung 2.7: Unterschiede in Repräsentation und Planung zwischen STRIPS-Verwandten (a) und HTNs (b). Die Nummern geben die Ausführungsreihenfolge an, während der graue Pfeil die Dimension der Planung anzeigt.

Aus der Menge der untersuchten Systeme sollen zwei HTN-Ansätze exemplarisch vorgestellt werden, die einerseits die inhaltlichen Konzepte Hierarchischer Aufgabennetzwerke gut darstellen und andererseits als sehr effizient bezeichnet werden können. Beide werden seit ihrer Entwicklung und Emplementierung am Beispiel verschiedener Roboter praktisch eingesetzt. Dies sind TDL (Abkürzung für: *Task Description Language*) und PRS (Abkürzung für: *Procedural Reasoning System*). Sie unterscheiden sich vor allem im Formalismus der Darstellung einer Handlung.

TDL: Task Description Language

Die von Simmons et al. entwickelte Beschreibungssprache für Roboterhandlungen TDL (siehe [Simmons 98, Simmons 94]) wird für unterschiedliche Aufgaben eingesetzt. TDL wurde

als Beschreibungssprache für Handlungen eines Serviceroboters entwickelt, wird inzwischen allerdings auch zur Koordination mehrerer Robotersysteme genutzt. TDL ist maßgeblich durch die ältere Beschreibungssprache ESL (Abkürzung für: *Execution Support Language*) geprägt (siehe [Gat 97a]).
Das Kernkonzept von TDL ist die Darstellung einer Handlung in hierarchischer Form als Baum. Dabei entspricht die Wurzel des Baums der Gesamthandlung, und auf jeder Hierarchiebene darunter wird die Handlung genauer spezifiziert. Die Blätter schließlich enthalten die einzelnen Elementaraktionen des Roboters. Bei der Ausführung werden diese entlang einer Tiefensuche geordnet von links nach rechts abgearbeitet. Abbildung 2.8 zeigt einen solchen Handlungsbaum.

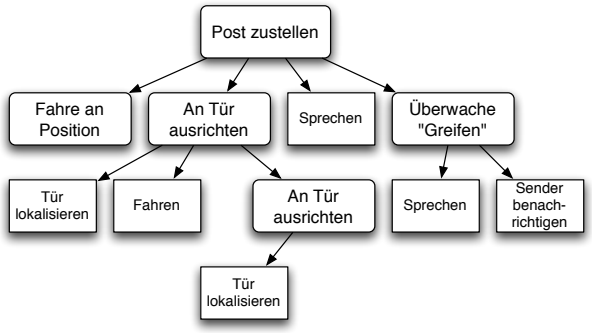


Abbildung 2.8: Beispiel eines Handlungsbaums nach [Simmons 98]

Die Elementaraktionen in den Blättern des Handlungsbaums sind frei durch den Entwickler spezifizierbar. Eine solche Aktion kann Berechnungen machen, den Handlungsbaum erweitern, oder Roboteraktionen auslösen. Während der Ausführung eines Programms kann also dynamisch auf Zustände und Ereignisse reagiert werden und der Baum entsprechend erweitert werden.
Die Mächtigkeit von TDL liegt folglicherweise im Inhalt der Blätter. Diese können den Baum beliebig manipulieren und somit auch alle benötigten Kontrollstrukturen wie Schleifen, bedingte Sprünge etc. implementieren. Es wird die Annahme getroffen, dass immer nur ein TDL Programm gleichzeitig im System existiert.
TDL ist als Erweiterung zu der Programmiersprache C++ entwickelt worden. Dies hat den Vorteil, dass alle Mechanismen einer verbreiteten und ausgereiften Programmiersprache zur Verfügung stehen. Der Nachteil hierbei ist jedoch, dass jedes erstellte Programm zuerst kompiliert werden muss und somit Änderungen oder Neuerstellungen nicht zur Laufzeit verwendet und eingebunden werden können.

PRS: Procedural Reasoning System

PRS [Ingrand 96, Alami 00, Alami 98] beinhaltet sowohl die Beschreibungssprache als auch die Ausführungskomponenten. Im Einzelnen ist es aus den folgenden Elementen aufgebaut:

- Ein aktuell beobachteter Umweltzustand wird in einer Datenbank gespeichert. Die hierin enthaltenen Hypothesen werden ständig durch neue Beoachtungen aktualisiert. Zusätzlich können auch nicht vorhandene Werte aus den vorhandenen berechnet werden.

- Eine Menge (Bibliothek) an Plänen beinhaltet alle momentan ausführbaren Handlungen. Diese Menge ist abhängig von der Anwendung und dem Kontext.

- Die momentan ausgeführten Aufgaben werden in einem Aufgabengraphen repräsentiert. Dieser ist ähnlich zu TDL aufgebaut, enthält also nur tatsächlich instanziierte Handlungen. Die Einschränkung auf Bäume als Graphenrepräsentation wird hier nicht getroffen.

- Die momentanen Ziele werden sowohl durch das System selbst als auch durch externe Ereignisse gesetzt. Anhand dieser Ziele wählt der Interpreter auszuführende Handlungen aus.

Die eigentliche Ausführungsbeschreibung, ein PRS Plan, besteht aus fünf wichtigen Schlüsselkomponenten:

- Die Ausführungsanweisung selbst besteht wiederum aus einer Menge zu erfüllender Ziele. Diese werden als weitere Handlungselemente betrachtet und zum Handlungsgraphen hinzugefügt.

- Um die Ausführbarkeit einer Handlung zu gewährleisten, enthält sie eine Ausführungsbedingung. Diese muss durch den momentanen Kontext erfüllt sein, um die Handlung ausführen zu können.

- Eine Handlungsanweisung kann als Reaktion auf beobachtete Zustandswechsel der Umwelt angestoßen werden. Dafür existiert die Möglichkeit, auf Zustandswechsel von Bedingungen zu warten.

- Laufzeitbedingungen für Handlungen können angegeben werden. Sind diese nicht mehr erfüllt, wird die Handlung abgebrochen.

- Bestimmte Bedingungen können aktiv aufrecht erhalten werden. Diese werden beobachtet, und falls sie nicht mehr erfüllt sind, werden Handlungen zur Wiederherstellung angestoßen.

PRS ist also ein prozedurales Expertensystem, das eng an die Datenbank des Momentanzustands gekoppelt ist. Diese wird verwendet, um Möglichkeiten, Ausnahmen und Umschaltpunkte für Handlungen zu identifizieren und umzusetzen. Die Handlungen sind als Handlungsgraph repräsentiert, der aus der Menge aller bekannten Handlungen zur Laufzeit durch den Interpreter aufgebaut und ausgeführt wird.

Weitere Handlungsdarstellungen als Hierarchische Aufgabennetzwerke sind SHOP2 (Abkürzung für: *Simple Hierarchical Ordered Planner 2*) (siehe [Nau 03]) oder UMCP (Abkürzung für: *Universal Method-Composition Planner*) (siehe [Erol 94c]). Beide Ansätze legen den Fokus auf eine effiziente Planung, und UMCP wurde vor allem zur Untersuchung und Formalisierung des Planungsvorgangs für HTN genutzt. [Erol 94a] gibt eine vollständige Formalisierung von HTN-Domänen, und zeigt, dass HTNs formal expressiver sind als Darstellungen ohne Dekomposition (wie STRIPS, siehe Kapitel 2.3.1). Insbesondere wird in [Erol 96] und [Erol 94b] gezeigt, dass STRIPS ein Spezialfall der HTNs ist.

2.3.3 Implizite Aufgabenbeschreibung

Implizite Handlungsbeschreibungen unterscheiden sich grundsätzlich von den klassischen Ansätzen der Künstlichen Intelligenz. Diese Ansätze verzichten mehr oder weniger vollständig auf die Verwendung von Symbolen als kleinste Einheit der Planung und Ausführung. Die Motivation hierfür ergibt sich aus zwei Überlegungen: Zum einen ist es strittig, dass das kognitive System des Menschen ein im Kern symbolverarbeitendes System darstellt. Vielmehr werden das Umweltmodell und die Handlungsplanung des Menschen als ein kontinuierlicher Prozess gesehen. Zum anderen ist der Vorgang der Symbolerstellung (engl.: *Symbol grounding*) ein schwieriger und mehrdeutiger Prozess, der bisher nicht automatisiert werden kann, da sich keine geschlossene Theorie findet, die die Eigenschaften eines solchen Symbols eindeutig festlegt.
In [Hart 04, Hart 05] wird die Handlung durch eine prozedurale Umweltmodellierung dargestellt. Der Zusammenhang zwischen Aktionen und dem jeweiligen Ergebnis ist in Form von Gewichten repräsentiert. Diese müssen aufgrund der hohen Dimensionalität mit Hilfe von Trainingsbeispielen gelernt werden. Vorteilhaft ist dabei, dass durch eine gute Wahl der Beispiele auch implizite Zusammenhänge zwischen Aktion und Ergebnis modelliert werden können. Nachteilig ist die schlechte Les- und Beweisbarkeit des Handlungswissens. Zudem müssen die Trainingsbeispiele umfassend und vollständig sein, um eine Domäne zu beschreiben.
Eine weitere Form der impliziten Handlungsmodellierung stellen die *Markov'schen Entscheidungsmodelle* (engl.: *Markov Decision Processes, MDP*) sowie deren Erweiterung für partielle Beobachtbarkeit der Umwelt POMDP (Abkürzung für: *Partially Observable Markov Decision Processes*), dar [Kaelbling 98, Schmidt-Rohr 06]. Hier besteht das Modell grundsätzlich aus zwei Komponenten: Weltmodellierung und Entscheidungsmodellierung. Das Weltmodell stellt auf subsymbolischer Ebene die Übergangswahrscheinlichkeiten für die Umwelt und jede mögliche Roboteraktion dar, während das Entscheidungsmodell oder *Belohnungsmodell* (engl.: *Reward model*) als ein über dem Zustandsraum definiertes Potential repräsentiert wird, anhand dessen die Aktionen des Roboters nach einem Maximierungskriterium gewählt werden.
Der Vorteil dieser Darstellungen besteht darin, dass Planungs- und Ausführungsschritt enger verzahnt werden können. Zudem kann das Welt- und Entscheidungsmodell gelernt und laufend verfeinert werden. Dagegen stehen zwei große Schwierigkeiten: Zum einen sind die benötigten Modelle hochdimensional, eine heuristische oder analytische Erstellung ist also unmöglich; zum anderen stellt das Planungs- bzw. Entscheidungsmodell eine lokale Optimierungsfunktion dar, das keine globalen Ziele explizit repräsentiert.

2.3.4 Bewertung

Tabelle 2.2 stellt die wichtigen Eigenschaften der vorgestellten Repräsentationen gegenüber. Die größte Bedeutung kommt dem Formalismus der jeweiligen Darstellung zu. Ein zusätzliches Kriterium ist die Erweiterbarkeit der Handlungsrepräsentation zur Laufzeit: Beispielsweise stellt TDL wie erwähnt eine Erweiterung der Übersetzersprache C++ dar, so dass eine Erweiterbarkeit oder Editierbarkeit von Handlungen zur Laufzeit nicht gegeben ist. SOUL dagegen verwendet eine eigene Repräsentation, die zur Laufzeit interpretiert wird. Es ist also möglich, Handlungen zur Laufzeit hinzuzufügen oder anzupassen. Die Mächtigkeit von SOUL

ist allerdings beschränkt, so dass keine komplexen Abläufe damit dargestellt werden können. RAP und PRS besitzen eine grundsätzlich editierbare Repräsentation, da sie ähnlich einer Erweiterung der interpretierten Programmiersprache LISP aufgebaut sind; praktisch ist ein Editieren und Verändern des Handlungswissens während der Ausführung jedoch schwierig, da die Interpretations- und Handlungsmechanismen darauf nicht ausgelegt sind. Diese Forderung ist jedoch grundlegend für Robotersysteme, die in Interaktion mit der Umwelt und dem Benutzer Handlungen lernen und anpassen können sollen. Implizite Repräsentationen wie POMDP Entscheidungsprozesse können nach heutigem Kenntnisstand nicht analytisch erzeugt werden, und sind also zur Darstellung von symbolischem Ausführungswissen ungeeignet.
Für die vorliegende Fragestellung kommt also keiner der betrachteten Ansätze in Betracht. Keines der untersuchten Systeme ist dafür ausgelegt, Handlungen während der Ausführung zu parametrieren oder zu verändern.
Das Konzept der Hierarchischen Aufgabennetzwerke besitzt allerdings wesentliche Vorteile gegenüber sequentiellen Darstellungen. Insbesondere die hohe Abstraktionsfähigkeit dieser Repräsentation erhöht die Verständlichkeit und Wiederverwendbarkeit von Teilhandlungen.

	STRIPS	RoboX	RAP	PRS	TDL	POMDP
Sprache	STRIPS	SOUL	RAP	PRS	C++ Erweiterung	implizite Repräsentation
Formalismus	Prädikatenlogische Zustandsdarstellung	Sequenz	Aufgabennetzwerk	Aufgabengraph	Aufgabenbaum	Markovprozess
Lesbarkeit	mäßig	gut	gut	gut	gut	schlecht
Editierbarkeit	nein	ja	schwierig	schwierig	nein	unklar

Tabelle 2.2: Vergleich der Eigenschaften der vorgestellten Handlungsrepräsentationen. Siehe auch [Pembeci 02].

2.4 Abbildung von Handlungswissen auf Robotersysteme

Ein bisher nur teilweise gelöstes Problem, und gleichzeitig eine der grundlegenden Fragestellungen der vorliegenden Arbeit, stellt die Übertragung abstrakten Handlungswissens auf ein dediziertes Robotersystem dar. Die Frage nach einer generalisierten Abbildungsvorschrift

wird in der Literatur oft übersehen, oder sie wird nur implizit gestellt. Es existieren zwei Forschungsrichtungen, die sich mit dieser Fragestellung auseinander setzen: Dies ist zum einen das *Lernen durch Imitation* (engl.: *Imitation Learning*), und zum anderen das *Programmieren durch Vormachen* (siehe auch Kapitel 3.1).
Das *Lernen durch Imitation* beschäftigt sich mit Lernverfahren, die mit minimalem Modellwissen von einem Menschen vorgeführte Aktionen für einen Roboter ausführbar machen sollen (siehe [Calinon 05, Billard 04, Breazeal 03, Matarić 02]). Es umfasst mehrere Fragestellungen. Die zwei wichtigsten werden oft als

1. *Was soll imitiert werden* (engl.: *What to imitate?*) und
2. *Wie soll imitiert werden* (engl.: *How to imitate?*)

tituliert [Calinon 05]. Dabei umfasst Fragestellung 1 die Frage nach den relevanten Aktionen und Bewegungen der Vorführung, während Fragestellung 2 die Übertragung des Gelernten auf den Roboter beinhaltet. Dieses Problem wird in der Literatur auch als *Korrespondenzproblem* (engl.: *Correspondence Problem*) bezeichnet [Alissandrakis 06, Alissandrakis 03]. Dieses Korrespondenzproblem umfasst die Fragestellung nach der Abbildung von Handlungen auf den Roboter, wird allerdings im Rahmen des Imitationslernens hauptsächlich für Bewegungen und Trajektorien betrachtet. Die Abbildung komplexer Handlungsketten wird nicht gelöst.
Die Fähigkeiten eines Roboters zur Imitation werden an denen des Menschen gemessen. Entsprechend orientiert sich die Entwicklung solcher Verfahren an Theorien über menschliches Lernen. Meltzoff und Moore postulieren für die Fähigkeiten von Kleinkindern zur Imitation eine Entwicklung in vier Schritten. Diese werden *Körperplappern* (engl.: *Body babbling*), *Imitation von Bewegungen* (engl.: *Imitation of body movements*), *Imitation von Aktionen an Objekten* (engl.: *Imitation of actions on objects*) und *Imitation basierend auf Intentionserkennung* (engl.: *Imitation based on inferring intentions of others*) genannt (siehe [Rao 04, Meltzoff 02]).
Interessant ist hierbei, dass erst der vierte Schritt das Verstehen von Intentionen und Zielen des Demonstrators voraussetzt. Existierende Theorien des Lernens durch Imitation beschränken sich allerdings oft auf die ersten drei Stadien, was sie ungeeignet macht zum Lernen komplexer, zielgerichteter Handlungen (siehe auch [Schaal 03]).

Im Gegensatz dazu wird bei aktuellen Ansätzen des *Programmierens durch Vormachen* der Fokus auf die symbolische Analyse der Vorführung sowie das Lernen von Handlungssequenzen gelegt. Da hier eine Extraktion der Intention (Ziele) aus der Vorführung notwendig ist, ist diese für die Übertragung auf den Roboter verfügbar. Allerdings wird diese Abbildung selten explizit betrachtet, und der Schwerpunkt eher auf die Extraktions- und Lernverfahren gelegt.

Für die Abbildung von Handlungswissen auf ein Robotersystem sind folgende Kriterien von zentraler Bedeutung:

- Die Ziele der Handlung sowie jeder einzelnen Teilhandlung müssen bekannt sein. Das Ergebnis von (Teil-)Handlungen kann dann mit dem erwarteten Ergebnis verglichen werden, um während der Ausführung Erfolg oder Misserfolg der Ausführung zu überwachen. Ansätze, die bei der Übertragung die Ziele nicht berücksichtigen, werden an dieser Stelle nicht betrachtet.

- Um die beobachtete Handlung auf die Darstellung von Handlungswissen des Roboters übertragen zu können, müssen beide Handlungsdarstellungen explizit definiert und bekannt sein. Dabei ist intuitiv klar, dass die Abbildung umso einfacher wird, je ähnlicher sich die Definitionsräume dieser beiden Darstellungen sind. Aus diesem Grund wird in vielen Ansätzen die Vorführung im *Konfigurationsraum des Roboters* durchgeführt. Dies vereinfacht die Abbildung, da alle vorgeführten Aktionen direkt in der Repräsentation des Roboters vorliegen. Allerdings ist dies für den Vorführenden wenig intuitiv. Es wäre also wünschenswert, den Definitionsraum von Vorführung und Ausführung trennen zu können, um die Demonstration durch den Menschen und die Ausführung durch den Roboter erfolgen zu lassen.

Im Folgenden werden in Kapitel 2.4.1 Ansätze aufgezeigt, die die Vorführung direkt im Konfigurationsraum des Roboters vornehmen und somit die Abbildung wesentlich vereinfachen. Kapitel 2.4.2 beschreibt aktuelle Arbeiten, die zwischen Vorführungs- und Ausführungsdefinitionsraum unterscheiden.

2.4.1 Vorführung und Beobachtung im Konfigurationsraum des Roboters

In vielen Ansätzen wird die Schwierigkeit der Abbildung auf den Roboter dadurch umgangen, dass die Vorführung direkt mit der Kinematik und den dem Roboter zur Verfügung stehenden Aktionsprimitiven durchgeführt wird. Die klassische Roboterprogrammierung mit *Teach-In*-Verfahren stellt dazu erste deterministische Ansätze und Lösungswege bereit; in ähnlicher Weise stellt sich diese Problematik bei Telepräsenzsystemen, wo der Mensch ebenso die erforderte Abbildung auf das ausführende System leistet.
Ein probabilistisches Modell für den Zusammenhang zwischen sensorischer Eingabe und aktorischer Ausgabe wird von Hart et al. in [Hart 05, Hart 04] vorgestellt. Dabei wird die Aufgabe nicht explizit vorgeführt; der Roboter führt vielmehr Versuche der Handlung mit Variation der Basisprimitive durch, wobei sowohl die grundsätzliche Handlungsfolge als auch das Erfolgsmaß a priori gegeben sind. Die Arbeiten wurden mit dem zweiarmigen Roboter "Dexter" durchgeführt. Die Ergebnisse der Versuche werden zum Aufbau eines probabilistischen Modells genutzt. Durch diese Vorgehensweise entfällt die Notwendigkeit der Abbildung des Handlungswissens auf den Roboter; alle Handlungsbeispiele sind direkt in dessen Konfigurationsraum gegeben.
Shon et al. stellen in [Shon 04] ein ebenfalls probabilistisches Modell des Imitationslernens vor. Dabei werden die notwendigen Schritte zum Lernen durch Imitation folgendermaßen bezeichnet (siehe auch [Rao 04]):

1. Zustandsidentifikation: Klassifikation von Zuständen durch Reduktion der Dimensionalität der zur Verfügung stehenden Sensordaten. Dabei muss relevante Information von unwichtiger getrennt werden.

2. Zustandsabbildung: Transformation vom Koordinatensystem des Demonstrators in das des Beobachters.

3. Lernen des Umweltmodells: Aufbau eines Modells über das Verhalten der Umwelt.

4. Lernen von Kriterien zur Handlungsauswahl: Aufbau einer Auswahlfunktion entsprechend der Vorführung.

5. Lernen von Sequenzen: Zerlegen und Speichern von Handlungen in Subhandlungen. Die Ziele einer Handlung müssen zusätzlich erkannt und zugeordnet werden können.

Die dabei betrachtete Abbildung von Handlungswissen in den Aktionsraum des Beobachters (Roboters) wird auf eine Koordinatentransformation reduziert und findet sich unter Punkt 2. Wichtige Fragestellungen wie die Abbildung zwischen grundsätzlich unterschiedlichen Sensor- und Aktorkonfigurationen werden also ausgeklammert.
Nicolescu und Matarić schlagen in [Nicolescu 03] vor, den Vorführungsprozess dahingehend zu modifizieren, dass der vorführende Mensch den Roboter durch die Vorführung steuert. Damit entfällt die Abbildung zwischen Demonstrator und Beobachter, und die verwendeten Aktionen und Bewegungen sind grundsätzlich durch den Roboter ausführbar. Den gleichen Ansatz wählen Peters und Campbell in [Peters II 03], indem die Vorführung mit Hilfe eines Teleoperationssystems durchgeführt wird. Es wird also ebenfalls die Vorführung im Konfigurationsraum des Roboters zur Verfügung gestellt.
Eine rein symbolische Abbildung wird von Ekvall und Kragic vorgeschlagen (siehe [Ekvall 06b, Ekvall 06a]). Der Vorgang der Programmierung und Wissensakquisition ist sehr ähnlich zu den Arbeiten von Pardowitz, Zöllner et al. (siehe Kapitel 3.1), die einzelnen Segmente der beobachteten Handlung werden allerdings ohne explizite Abbildung auf den Roboter übertragen. Die Ausführung erfolgt mit Hilfe eines klassischen Trajektorienplaners, die Vorführung wird also nur teilweise ausgenutzt.

Wählt man die Vorführung im Konfigurationsraum des Roboters, ergibt sich vor allem der Vorteil, dass die Abbildung zwischen Demonstrator und Beobachter entfällt. Allerdings ergeben sich auch wesentliche Nachteile:

- Oft wird zusätzliche Sensorik und Aktorik benötigt, um dem Menschen die Vorführung mit dem Roboter zu erlauben. Dies ist beispielsweise bei dem von Peters und Campbell verwendeten Teleoperationssystem der Fall.

- Während der Vorführung muss der Demonstrator das Aktionsrepertoire des Roboters verwenden. Dies hat zwei Konsequenzen: Zum einen muss der Mensch die geforderte Abbildung leisten, wozu nicht jeder Mensch in der Lage ist. Vor allem, wenn Menschen im Umgang mit Robotern nicht geübt sind, werden sich hier Probleme ergeben. Zum anderen ist diese vom Demonstrator geleistete Abbildung durch dessen Erfahrungen und Möglichkeiten gefärbt. Er wird also oft die Möglichkeiten des Roboters nicht ausschöpfen, oder für den Roboter suboptimale Wege wählen. Dies könnte verhindert werden, wenn das Robotersystem die Handlung beobachtet und dann selbst auf eigene Aktionen überträgt.

- Durch die Vorführung direkt im Roboterkonfigurationsraum ist es nicht möglich, eine Vorführung für mehr als einen Roboter zu nutzen. Da keine roboterunabhängige Repräsentation der beobachteten Handlung existiert, muss diese für jeden Robotertyp neu und anders demonstriert werden.

Um diese Probleme zu lösen, wird die Trennung der Konfigurationsräume von Vorführung und Ausführung gewählt. Entsprechende Ansätze werden im Folgenden vorgestellt.

2.4.2 Trennung der Konfigurationsräume von Vorführung und Ausführung

Arbeiten zur Abbildung von Handlungswissen für die Ausführung durch einen Roboter können unterschieden werden in Ansätze aus dem Imitationslernen und Ansätze des Programmierens durch Vormachen. Der Unterschied liegt zum einen in der Menge an verwendetem Modellwissen, zum anderen im Abstraktionsgrad der Handlung. Zusätzlich finden sich Abbildungen für spezielle Probleme, wie z.B. die Abbildung von erkannten Griffen auf eine Roboterhand. Diese werden oft im Rahmen von PdV-Systemen betrachtet.

Abbildung im Rahmen des Imitationslernens

Die Arbeit von Matarić et al. in [Matarić 02, Weber 00] basiert auf zwei wichtigen Voraussetzungen für die Abbildung:

- Die Basis für die Imitation von Bewegungen und Verhalten wird durch eine endliche Menge an Bewegungsprimitiven gelegt. Diese Menge an Primitiven ist ausreichend, um alle notwendigen Bewegungen auszuführen. Primitive können dafür parallel und sequentiell verkettet und kombiniert werden.

- Im Gehirn des Menschen werden sogenannte *Spiegelneuronen* (engl.: *Mirror neurons*) postuliert, deren Existenz durch verschiedene Untersuchungen bestätigt wurde. Diese sind in der Lage, beobachtete Handlungen direkt auf die Menge der eigenen Bewegungsprimitive abzubilden. Diese Repräsentation wird dann für das Lernen und die Imitation von Handlungen genutzt.

Auf Basis dieser Voraussetzungen wird ein Modell für Lernen durch Imitation vorgeschlagen. Der für die vorliegende Betrachtung relevante Teil betrifft dabei die Abbildung von Handlungen auf das eigene Repertoire an Bewegungsprimitiven. Diese Abbildung wird in [Matarić 02] auf die Abbildung zwischen Agenten mit identischer Morphologie beschränkt. Abbildungen zwischen unterschiedlich strukturierten Agenten werden nicht betrachtet. Es wird jedoch die grundlegende Einschränkung getroffen, dass zwischen Beobachtung und Interpretation sowie Ausführung nur Bewegungen abgebildet werden; die Abbildung von Handlungssequenzen und Zielen wird nicht betrachtet.

Explizit wird die Abbildung zwischen Demonstrator und Imitator als Korrespondenzproblem betrachtet (siehe Kapitel 2.4). Nehaniv definiert dieses Korrespondenzproblem in [Nehaniv 02] folgendermaßen:

> Gegeben sei ein beobachtetes Verhalten des Demonstrators, das bei gegebenem Startzustand den Demonstrator durch eine Sequenz (oder Hierarchie) von Zwischenzielen führt. Dieses Verhalten kann sowohl in Form von Zuständen, Aktionen oder Effekten dargestellt sein, als auch aus Reaktionen auf sensorische Stimuli und externe Ereignisse bestehen. Das Problem ist, eine Aktionssequenz für die eigene Ausführungsstruktur zu finden und auszuführen, die über die korrespondierenden Zwischenziele führt. Dabei sollen korrespondierende Aktionen, Zustände und Effekte gewählt werden.

Es sollen also Aktionen gefunden werden, die beobachtete und extrahierte Ziele erreichen. Dabei soll eine Abbildung von Aktionen, Zuständen und Effekten erfolgen, wobei Effekte als Zustandsänderungen zu verstehen sind.

Alissandrakis et al. [Alissandrakis 06] modellieren die Abbildung zwischen verschiedenen kinematischen Strukturen als eine lineare Abbildung von Freiheitsgraden. Dafür wird eine Korrespondenzmatrix $\mathcal{C}$ aufgestellt:

$$\mathcal{C} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{bmatrix} \tag{2.1}$$

Dabei soll der Demonstrator n und der Imitator m Freiheitsgrade besitzen; es gilt demnach $\vec{x}_{Demonstrator} = \mathcal{C} \cdot \vec{x}_{Imitator}$. Bei diesem Ansatz existiert also immer eine direkte Abbildung von Bewegungen. Basierend auf $\mathcal{C}$ werden verschiedene Abbildungstypen vorgestellt. Dies sind beispielsweise die Identität und die spiegelsymmetrische Abbildung für identische Kinematik von Demonstrator und Imitator, aber auch Abbildungen zwischen verschiedenen Körpern. Allerdings werden alle Korrespondenzmatrizen fest durch den Programmierer vorgegeben, es existiert also keine Möglichkeit, diese anhand der gegebenen Aufgabe zu erzeugen.
Imitation unter Einbezug der Effekte wird in [Alissandrakis 03, Alissandrakis 02b, Alissandrakis 02a] betrachtet. Die betrachtete Umgebung ist ein Schachfeld, auf dem sich verschiedene Figuren bewegen. Als Effekt wird die Positionsänderung der jeweiligen Figur angesehen; dies ist implizit gegeben. Abbildung 2.9 zeigt ein Beispiel. Die implizite Definition der

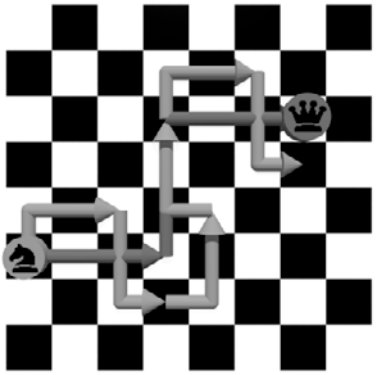


Abbildung 2.9: Schachfeld-Umgebung zur Darstellung von Effekten bei der Imitation. Aus [Alissandrakis 03]

Effekte birgt einen großen Nachteil, wenn der Ansatz auf reale Probleme übertragen werden soll. Die Annahme einer reinen Positionierungsaufgabe ist für komplexere Aufgaben nicht tragfähig. Speziell muss es für komplexe Handlungen möglich sein, Teilhandlungen auf völlig andere Weise als der Demonstrator zu lösen, solange diese die gleichen Ziele erreichen. Dies ist mit dem vorgestellten Ansatz nicht möglich.

Ein ähnlicher Ansatz wird von Billard et al. (siehe [Calinon 05]) gewählt. Zwischen Demonstrator und Imitator wird eine Abbildung von Merkmalen als bekannt vorausgesetzt; aus mehreren Demonstrationen werden die relevanten Merkmale mit statistischen Methoden bestimmt und für die endgültige Abbildung verwendet. Dabei werden die Einschränkungen von Demonstrator und Imitator nicht als identisch angenommen, die Arbeitsräume jedoch werden explizit durch eine einfache Skalierung abgebildet.
Breazeal und Scassellati betrachten das Problem des Imitationslernens in [Breazeal 03, Breazeal 02] nur für identische Kinematik zwischen Demonstrator und Imitator, wodurch eine explizite Abbildung überflüssig wird. Ähnlich gehen Chella et al. in [Chella 05] vor, allerdings wird hier die Existenz einer invertierbaren Abbildungsfunktion für Bewegungen postuliert.

Abbildung im Rahmen von PdV

Ansätze des Programmieren durch Vormachen sind üblicherweise sehr viel stärker modellbasiert aufgebaut als die des Imitationslernens. Dies beruht darauf, dass das Ziel wesentlich abstrakter und symbolischer definiert ist. Der Fokus liegt hierbei auf dem Lernen und Abstrahieren von Aktionssequenzen zum Erreichen eines Zielzustands. Dazu werden a priori definierte Handlungsmodelle verwendet, die symbolisch aufgebaut sind und durch die Vorführung parametriert werden. Dies stimmt wiederum überein mit dem Paradigma der Verwendung von Aktionsprimitiven (siehe Kapitel 2.3.1 und 2.4.2), das auch vielen Ansätzen des Imitationslernens zu Grunde liegt.

Programmieren durch Vormachen wird in der Robotik seit den 80er Jahren erforscht. Die Grundlagen hierzu können in [Ude 96, Kaiser 96] und [Friedrich 98] gefunden werden, spätere Arbeiten in [Rogalla 02, Ehrenmann 02, Zöllner 05b].
Rogalla betrachtet in [Rogalla 02] die Frage der Abbildung von Handlungswissen auf ausführende Roboter. Dabei werden zur Generierung eines Roboterprogramms verschiedene Phasen durchlaufen. Ausgehend von einem Makrooperator und einer Falldatenbasis wird ein Roboterprogramm aus den Trajektorien des Makrooperators erzeugt. Dieses wird anhand einer Simulation auf Ausführbarkeit und Kollisionsfreiheit geprüft und gegebenenfalls halbautomatisch oder manuell durch den Menschen auf die Ausführungssituation angepasst. Hierzu können auch Beispiele aus der Falldatenbasis herangezogen werden.
Die Abbildung erfolgt auf subsymbolischer Ebene. Es wird vorausgesetzt, dass die im Makrooperator abgelegte Reihenfolge und Anordnung der Elementaroperatoren auf das Zielsystem übertragbar ist. Aus diesem Grund kann die Abbildung auf eine Transformation zurückgeführt werden, um die Regelungsparameter für den Roboter aus den beobachteten Trajektorien zu erzeugen. Abbildung 2.10 zeigt einen Überblick über dieses Verfahren.
Besonderes Augenmerk wird dabei auf die Abbildung von Griffen auf Roboterhände gelegt. Diese Abbildung basiert auf dem Konzept der *virtuellen Finger* (siehe Abbildung 2.11). Diese virtuellen Finger dienen als unabhängige Griffdarstellung, die sowohl aus beobachteten Griffen erzeugt werden kann, als auch zur Erzeugung von Robotergriffen verwendet wird. Virtuelle Finger fassen reale Finger ihrer Wirkung nach zusammen; es werden also solche Finger gemeinsam betrachtet, die ihrer Kraft nach eine ähnliche Funktion innerhalb des speziellen Griffs besitzen.
In Abbildung 2.12 wird das Vorgehen bei der Abbildung von Griffen auf das Zielsystem aufgezeigt. Dabei ist zu erkennen, dass die Abbildung subsymbolisch auf Trajektorienebene

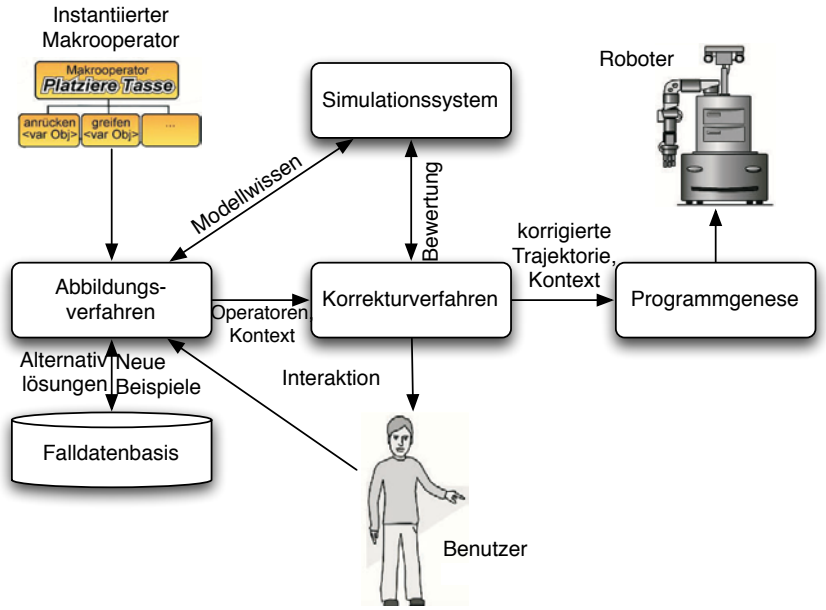
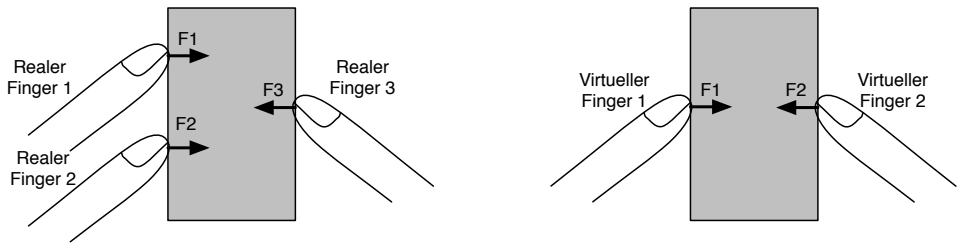


Abbildung 2.10: Abbildung von Handlungen auf einen Roboter, nach [Rogalla 02]



(a) (b)

Abbildung 2.11: Schematische Darstellung eines Präzisionsgriffs nach [Rogalla 02]. (a) der Griff mit drei Fingern, (b) die Abbildung auf zwei virtuelle Finger

erfolgt. Dies geschieht einzeln für jeden Elementaroperator; die Gesamtübertragung wird dann aus den einzelnen Teilen zusammengesetzt. Die erzeugte Sequenz hat also den gleichen Aufbau wie der aus der Beobachtung extrahierte Makrooperator, wobei jeder Elementaroperator (Aktionsprimitive) separat abgebildet wurde.

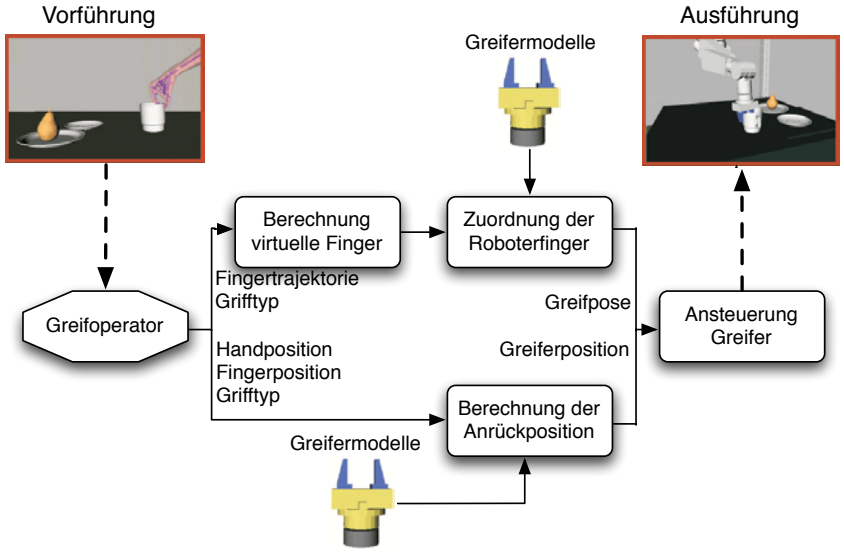


Abbildung 2.12: Phasenmodell der Griffabbildung nach [Rogalla 02].

2.4.3 Bewertung

Die betrachteten Ansätze zur Abbildung von Handlungswissen auf Robotersysteme nähern sich der Problematik grundsätzlich aus zwei unterschiedlichen Richtungen: Ansätze des Imitationslernens betrachten hauptsächlich die Übertragung von Bewegungen, sowohl nach Kriterien der Trajektorie als auch des Ziels. Dabei wird das Ziel oft über Merkmale der Effektorposition beschrieben. Ansätze des Programmieren durch Vormachen dagegen basieren auf einem a priori gegebenen Handlungsmodell, das die entsprechenden Aktionsprimitive sowohl in der Vorführung als auch in der Ausführung umfasst. Die Abbildung besteht dann in einer Parametrierung der Ausführungsprimitive.
Um komplexe Handlungen auf Robotersysteme abbilden zu können, ist es zwingend notwendig, die Ziele der Handlung mit einzubeziehen. Diese Forderung wird bei Ansätzen des Imitationslernens oft vernachlässigt, und es werden nur subsymbolische Eigenschaften abgebildet. Andererseits ist die bisher im Rahmen des Programmierens durch Vormachen praktizierte sequentielle Abbildung nicht ausreichend, um Handlungen auf ein Robotersystem mit stark unterschiedlichem Aufbau zu übertragen. Speziell die Abbildung auf eine Handlungsbeschreibung mit anderem Aufbau ist bisher unbeachtet geblieben.
Abbildung 2.13 ordnet die genannten Verfahren zur Abbildung von Handlungswissen nach den in Kapitel 2.4 beschriebenen Kriterien. Es wird dafür unterschieden zwischen Systemen mit identischem Aufbau von Demonstrator und Imitator, was auch Vorführungen im Konfigurationsraum des Roboters umfasst, und solchen Systemen, die unterschiedliche Konfigurationsräume zulassen. Auf der zweiten Achse ist das Abstraktionsniveau der Ansätze

aufgetragen. Es zeigt sich, dass die Abbildung von Handlungen auf komplexer Ebene bisher nur unzureichend betrachtet wurde.

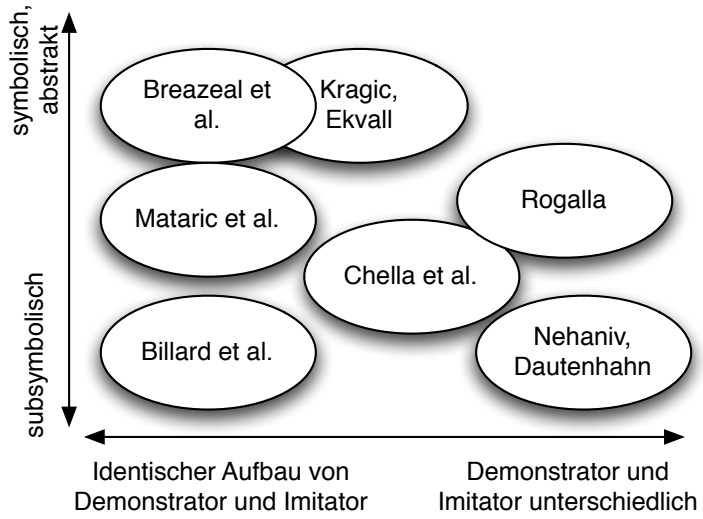


Abbildung 2.13: Einordnung der betrachteten Abbildungsverfahren

2.5 Interaktion mit Robotersystemen

Die Interaktion zwischen Mensch und Roboter ist in den letzten 10 Jahren zu einem sehr lebendigen und aktuellen Forschungsthema geworden. Grundsätzlich umfasst diese Interaktion beide Kommunikationsrichtungen: Vom Menschen zum Roboter und umgekehrt. Die größere Herausforderung stellt dabei unbestritten Ersteres dar: Die Erkennung und Interpretation von Sprache, Gestik, Bewegungen des Menschen durch den Roboter.
Ziel der Forschung ist es, direkte, natürliche und spontane Interaktion zwischen Robotern und Menschen zu ermöglichen. Roboter sollen in der Lage sein, den Menschen mit Hilfe verschiedener Kommunikationskanäle wie Sprache, Gestik, Mimik oder Körpersprache zu verstehen. Auch die im Programmieren durch Vormachen verwendete Handlungsbeobachtung kann hier genannt werden.
Die Entwicklung von Verfahren zur Beobachtung und Interpretation menschlicher Aktionen und Handlungen ist in den letzten Jahren Gegenstand großen Interesses. Einerseits liegt dies daran, dass die Entwicklung von entsprechender Sensorik stark vorangeschritten ist. Dies betrifft in erster Linie bildgebende Sensoren wie Kameras, aber auch andere Messverfahren wie magnetfeldbasierte Lokalisation oder laserbasierte Tiefenmessungen. Andererseits ermöglicht die Rechenleistung moderner Computer inzwischen aufwendige Verfahren in Echtzeit zu berechnen, so dass auch komplexere Algorithmen zur Beobachtung eingesetzt werden können.
Verfahren zur Beobachtung und Interpretation menschlicher Handlungen können für den Bereich der Servicerobotik grundsätzlich nach der verwendeten Sensorik unterschieden werden. Dabei kann einerseits zwischen *invasiven* Messverfahren, die mit am Körper des Menschen angebrachten Sensoren oder Landmarken arbeiten, und andererseits *nicht invasiven* Verfahren, die ohne zusätzlich angebrachte Hilfsmittel auskommen, unterschieden werden.

Beispiele zu invasiven Verfahren finden sich auf dem Gebiet der *Erweiterten* oder *Virtuellen Realität*, wo der Mensch z.B. eine Durchsicht- oder Projektionsbrille trägt. Deren Lage im Raum wird mit verschiedenen lokal angebrachten Landmarken oder Sensoren gemessen [Hanheide 06, Giesler 05, Salb 03a]. Auf diese Weise wird die Kopfbewegung und Blickrichtung des Menschen gemessen (siehe Abbildung 2.14). Die Brille wird dann verwendet, um dem Menschen zusätzliche Information anhand seiner momentanen Lage in den Sichtbereich einzublenden.

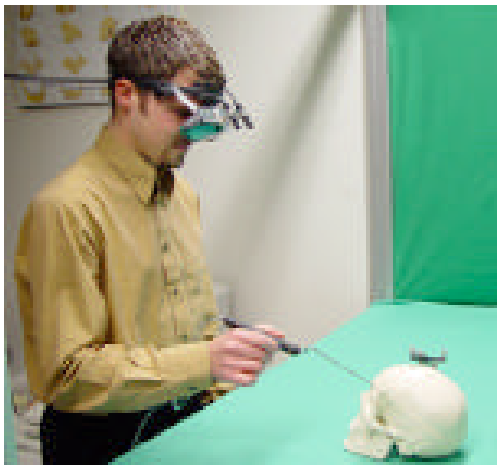


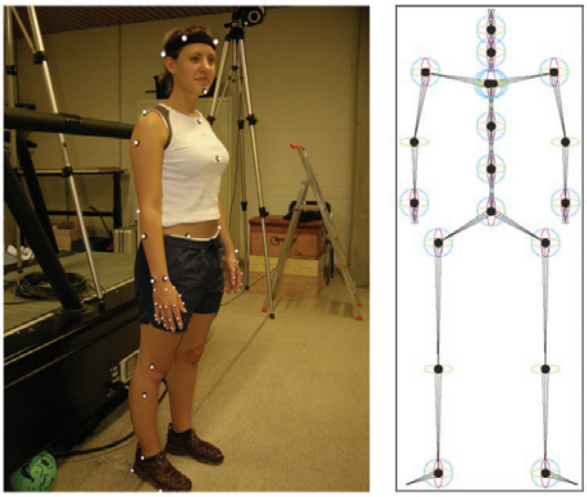
Abbildung 2.14: Beispiel einer Umgebung zur Erweiterten Realität mit invasiver Messung und Verfolgung der Kopfbewegung des Menschen, aus [Salb 03b]

Ein weiteres Beispiel für ein invasives Messverfahren ist das Bewegungserfassungssystem (engl.: *Motion Capture System, MCS*) der Firma Vicon (siehe [Vicon 06]). Es erlaubt eine hochgenaue Verfolgung der beobachteten Person, benötigt allerdings eine große Anzahl verteilter Spezialkameras, außerdem muss die beobachtete Person mit ca. 80 kalibrierten Landmarken versehen werden (Abbildung 2.15).
Invasive Verfahren haben grundsätzlich den Nachteil, dass die jeweiligen Sensoren oder Landmarken zuerst angebracht werden müssen. Dies ist für intuitive Interaktion mit ungeübten Personen nicht erwünscht; deshalb werden solche Systeme hauptsächlich als Referenzsysteme genutzt, um die Sensorsysteme des Roboters zu bewerten.

Die vorliegende Arbeit verfolgt den Ansatz, bereits vorhandenes Handlungswissen über Interaktion mit dem Benutzer während der Ausführung zu parametrieren (siehe Kapitel 1.1.2). Es ist also notwendig, Bewegungen des Menschen beobachten und interpretieren zu können. Hierfür sind invasive Messverfahren aus den genannten Gründen ungeeignet. Deshalb werden im Folgenden nur nicht invasive Verfahren betrachtet, die für die Beobachtung und Interpretation der Handlungen des Menschen und speziell seiner Bewegungen eingesetzt werden können.

2.5.1 Beobachtung des Menschen

Ehrenmann stellt in [Ehrenmann 02] einen Katalog von *kognitiven Operatoren* vor, die zur Handlungsbeobachtung dienen können. Es wird hier grundsätzlich zwischen Beobachtung



(a) (b)

Abbildung 2.15: Bewegungsbeobachtung mit dem Vicon-System, aus [Stein 06]. Die Verteilung der Landmarken ist in (a) zu sehen; (b) zeigt das verwendete Körpermodell.

in Vorführumgebung und Ausführumgebung (siehe auch [Ehrenmann 03, Ehrenmann 01b]) unterschieden, wobei die Ausführumgebung dem Roboter entspricht.

Die vorliegende Arbeit konzentriert sich auf diesen Ausführungskontext, und setzt keinerlei externe Sensorik voraus. In [Ehrenmann 02] werden für die Beobachtung des Menschen in diesem Kontext Farbkameras verwendet, um Hände und Kopf des Menschen verfolgen zu können (siehe Abbildung 2.16). Aus den so extrahierten Konturen und Trajektorien der

Abbildung 2.16: Verfolgung der Hände und des Kopfes, aus [Ehrenmann 02]

Hände werden sowohl statische als auch dynamische Gesten erfolgreich extrahiert. Dies ist ein wichtiger Schritt hin zu intuitiver Interaktion; ähnliche Ansätze können in [Nickel 04a, Nickel 04b] gefunden werden.

Das Verfahren versagt allerdings dann, wenn die zu Grunde liegende Sensorik keine ausreichende Information liefern kann. Dies ist bei Verdeckungen der Fall, oder bei unzureichenden Lichtverhältnissen. Nicht zu vernachlässigen ist auch die begrenzte Auflösung der Kameras, die korrekte Messungen beispielsweise der Handkontur nur in einem kleinen Abstandsbereich zulassen. Sind Hände oder Kopf nicht mehr korrekt detektierbar, kann keine weitere Aussage über die Körperhaltung (Pose) und Bewegung des Menschen getroffen werden.

Im Gegensatz dazu stehen Verfahren, die die gesamte Haltung des Körpers verfolgen. Dadurch kann ein höheres Maß an Robustheit erreicht werden, allerdings müssen auch aufwendigere Algorithmen und Modelle angewendet werden.
Azad et al. schätzen die Körperpose aus Mono-Kamerabildern mit Hilfe von Partikelfiltern (siehe [Azad 04]). Jedes Partikel bildet eine Hypothese der Körperpose, so dass die Dimensionalität der Anzahl der Freiheitsgrade im Modell entspricht. Die Wahrscheinlichkeit jedes Partikels wird über eine Projektion des zugehörigen Modells mit nachfolgendem Vergleich von Bild und Projektion berechnet. Ein Vorteil des Partikelfilters liegt darin, dass die gewählten Konfigurationen a priori im Definitionsbereich der Gelenkwinkel gehalten werden können. Der Nachteil ist die hohe Dimensionalität des Suchraums, und vor allem die mit der Dimensionalität steigende Anzahl benötigter Partikel. Durch die Wahl des 2D-Sensors ergeben sich zusätzlich zwangsläufig Mehrdeutigkeiten, die durch das Filter aufgelöst werden müssen. Dies führt zu hohen Laufzeiten der Berechnungen, was die Echtzeitfähigkeit solcher Ansätze begrenzt.
Die Verwendung unterschiedlicher Sensorik zur Positionsschätzung eines Menschen wird von Sagerer et al. in [Spexard 06, Fritsch 03, Kleinehagenbrock 02] mit einem *Multi-Modal Anchoring* (engl. für: *Multimodale Verankerung, oder freier: Einhängen verschiedener Modalitäten*) Ansatz vorgeschlagen. Das ursprünglich von Coradeschi und Saffiotti (siehe [Coradeschi 00, Coradeschi 01]) vorgeschlagene Multi-Modal Anchoring bildet ein einheitliches Rahmenwerk, um verschiedene Sensordaten zu fusionieren. Dazu wird der zu beobachtende Mensch als ein zusammengesetzter Körper modelliert, dessen Komponenten mit verschiedenen Modalitäten lokalisiert werden können. Eine für die Fusion verwendete Messung besteht aus der Messung selbst, dem zugehörigen Symbol, und der Schätzung für die zu beobachtenden Parameter, also die Position und Orientierung. Ein solcher Komplex wird *Anker* genannt, und alle Anker werden durch die Fusion zu einer Gesamtschätzung verarbeitet. Abbildung 2.17 zeigt schematisch den Aufbau des Multi-Modal Anchoring. Dabei zeigt sich auch, dass das Multi-Modal Anchoring einen Rahmen und eine geschlossene Modellierung von Messungen und den zugehörigen Symbolen darstellt, die Fusion selbst allerdings nicht beschreibt. Ein probabilistisches Modell der Sensoren sowie der Fusion ist hierdurch nicht festgelegt, dies muss laut Abbildung 2.17 im Modul *Fusion Models* beschrieben und im Modul *Anchor Fusion* verarbeitet werden. Die geschätzten Parameter umfassen Position und Orientierung der Person, nicht jedoch die Körperhaltung. Um beispielsweise Gesten erkennen zu können, ist der vorgeschlagene Ansatz so also nicht geeignet.
Im Gegensatz dazu steht der von Sidenbladh in [Sidenbladh 01, Sidenbladh 00] vorgeschlagene Ansatz. Das verwendete Körpermodell für die Posenschätzung ist aus Zylindern zusammengesetzt, um verschiedene menschliche Posen modellieren und verfolgen zu können. Abbildung 2.18 zeigt dieses Modell. Jedes Gelenk wird mit den ihm eigenen Freiheitsgraden modelliert, so dass sich ein Modell mit insgesamt 25 Parametern ergibt. Das Modell ist hierarchisch aufgebaut, mit dem Torso als Wurzel des kinematischen Modells, und den Extremitäten als untergeordneten Elementen. Auf Basis eines Kamera-Projektionsmodells, einer Vereinfachung für die Projektion von Zylindern und einer Annahme über die Kontinuität der Textur kann eine Wahrscheinlichkeitsfunktion für die Ähnlichkeit zwischen einer Hypothese und dem Bild definiert werden, die aus einer Gleich- und einer Gaußverteilung besteht:

$$p_{image} = \frac{\epsilon}{256} + \frac{1-\epsilon}{\sqrt{2\pi}\sigma(\alpha_j)} \exp\left(- \sum_{i=1}^{n} \frac{(\mathbf{I}_t(x_{j,i}) - \hat{\mathbf{I}}_t(x_{j,i}))^2}{2\sigma^2(\alpha_j)} \right) \tag{2.2}$$

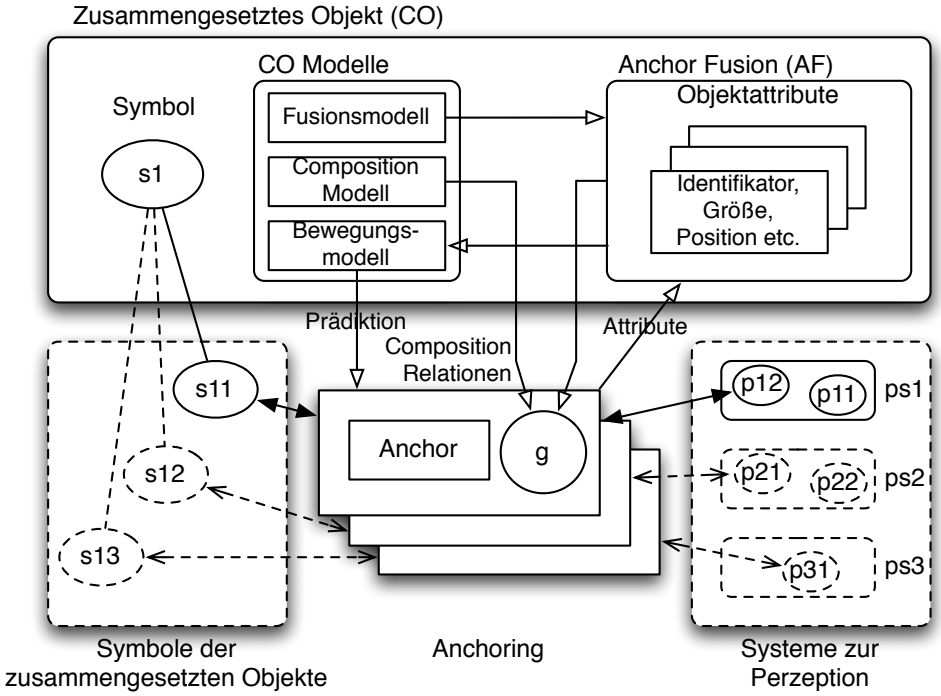


Abbildung 2.17: Multi-Modal Anchoring Ansatz zur Fusion verschiedener Modalitäten für die Verfolgung der Position und Orientierung von Menschen, nach [Kleinehagenbrock 02]

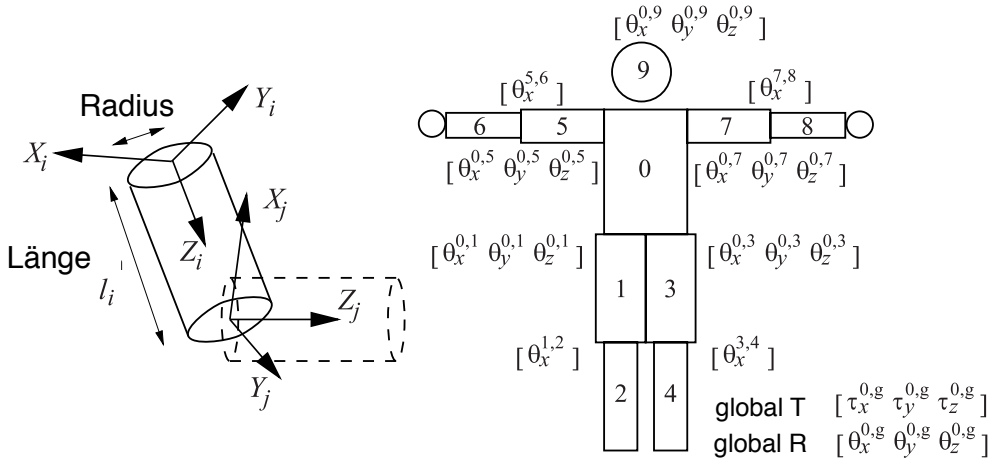


Abbildung 2.18: Modellierung eines starren Körperteils mit einem Zylinder, sowie das zusammengesetzte Körpermodell für den menschlichen Körper, aus [Sidenbladh 00]

Diese Wahrscheinlichkeit wird für jedes modellierte Körperteil separat berechnet. Der Index $i = 1 \ldots n$ wird über alle Pixel des jeweiligen Modells gezählt, und $\hat{\mathbf{I}}$ bezeichnet die erwartete Intensität. Um Ausreißer im Modell zu berücksichtigen, wird ein Gleichverteilungsanteil einbezogen. α_j bezeichnet den Winkel zwischen der Hauptachse des Zylinders und der Blickrichtung. Damit werden im Profil sichtbare Teile stärker gewichtet. Die Gesamtwahrscheinlichkeit ergibt sich aus

$$p(\mathbf{I}_t|\phi_t, \mathbf{R}_t) = \prod_j p_j \tag{2.3}$$

Mit dieser Wahrscheinlichkeitsfunktion wird jede Hypothese bewertet.
Zusätzlich zum kinematischen Modell und der Modellierung der Ähnlichkeitsfunktion führt Sidenbladh ein Dynamikmodell ein. Dieses besteht aus zwei Teilen:

- Ein Glättungsoperator glättet die Bewegung auf Ebene der Gelenkwinkel. Dabei werden alle Freiheitsgrade als unabhängig angenommen.
- Ein spezifisches Dynamikmodell wird für die beobachtete Aktivität aufgestellt. Das in [Sidenbladh 00] aufgezeigte Bewegungsmodell für *Laufen* wurde mit Hilfe von aufgezeichneten Bewegungstrajektorien gelernt. Die Trajektorien wurden von verschiedenen Personen mit Hilfe eines kommerziellen Systems zur Posenverfolgung (engl.: *Motion Capture*) Systems aufgenommen.

Der beschriebene Ansatz benötigt laut [Sidenbladh 00] für die Berechnung ca. fünf Minuten pro Bild, bei 10.000 Zustandshypothesen.
Speziell das aktivitätsspezifische Dynamikmodell stellt in diesem Ansatz eine starke Einschränkung dar. Die Verwendung eines solchen Modells setzt das Wissen über die Aktivität der beobachteten Person voraus, was für die Anwendungen der hier vorliegenden Arbeit nicht gegeben ist. Gleichzeitig ist die notwendige Mächtigkeit des eingesetzten Dynamikmodells ein Hinweis darauf, dass die Ähnlichkeitsfunktion allein für die Verfolgung nicht ausreichend ist. Die hohe Laufzeit des Verfahrens, bedingt durch die hohe Komplexität, macht es zusätzlich untauglich für Echtzeitanwendungen.
Ein Ansatz zur 3D-Posenschätzung auf Basis von Stereo-Tiefenbildern wird von Demirdjian et al. in [Demirdjian 03, Demirdjian 02] vorgeschlagen. Die Posenschätzung soll hier zur Erkennung von Gesten und allgemein zur Unterstützung der Mensch-Roboter Interaktion verwendet werden. Das Körpermodell, das für die Schätzung verwendet wird, besteht aus starren Teilkörpern, die über Kugelgelenke verbunden sind. Die Körperteile werden einzeln mit Hilfe des *Iterative Closest Point* Ansatzes (engl. für: *Iterative Nächste-Punkte-Berechnung*, Abk.: *ICP*) iterativ anhand der gemessenen Tiefenbilder verfolgt. In weiteren Schritten werden die Gelenkbedingungen, die durch die separate Behandlung jedes Körperteils im Allgemeinen verletzt werden, wiederhergestellt. Abbildung 2.19 zeigt dieses Vorgehen schematisch auf.
Der ICP-Schritt kann nach [Demirdjian 03] folgendermaßen beschrieben werden:

- Für jeden Punkt P_i der Messung wird der Punkt P'_j auf der Modelloberfläche mit minimaler Distanz $d(P_i, P'_j)$ gefunden. Der Vektor $\overrightarrow{P'_j P_i}$ wird als lokale Verschiebung zwischen Modell und Messung interpretiert.
- Die benötigte Transformation wird durch Integration über alle lokalen Verschiebungen geschätzt.

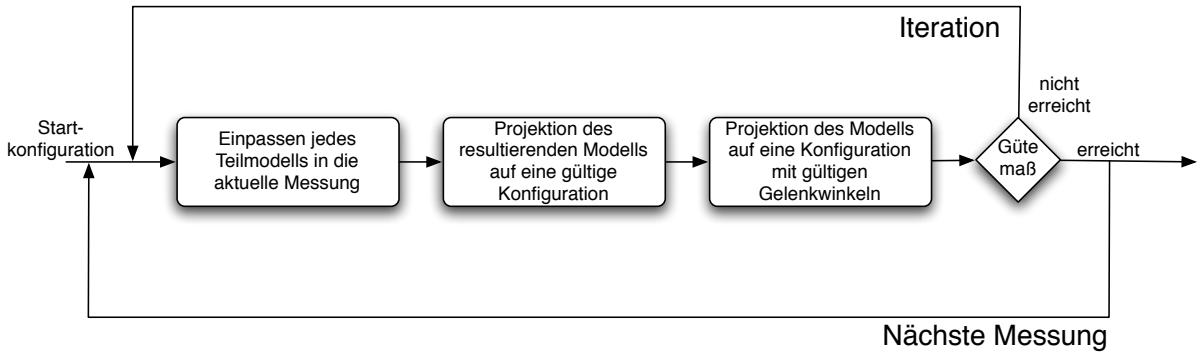


Abbildung 2.19: Iterative Posenschätzung nach Demirdjian et al.[Demirdjian 03]. Jedes Körperteil wird separat an den aktuellen Datensatz angepasst, die notwendigen Randbedingungen aus der kinematischen Modellierung werden in weiteren Schritten wiederhergestellt.

- Die Transformation wird auf das Modell angewandt. Ist die Veränderung unter einem festgelegten Schwellwert, wird abgebrochen.

Diese Schritte werden wiederholt, bis das Abbruchkriterium erfüllt ist.
Danach sind alle Modellteile unabhängig angepasst worden, und die Gelenkbedingungen im Allgemeinen nicht mehr erfüllt. Dies wird durch weitere Berechnungen sichergestellt (siehe Abbildung 2.19). Sei Δ die Gesamttransformation, die durch den ICP für alle Modellteile berechnet wurde. Die gültige Transformation $\Delta^\star$ kann gefunden werden, indem Δ in den Raum aller gültigen Transformationen projiziert wird. Dazu wird die Mahalanobis-Distanz berechnet und minimiert (siehe auch [Steinbach 05]):

$$E^2(\Delta^\star) = (\Delta^\star - \Delta)^T \cdot \Sigma^{-1} \cdot (\Delta^\star - \Delta) \tag{2.4}$$

Dabei ist Σ die Kovarianzmatrix der Transformation, die aus den einzelnen Transformationen gewonnen wird. Die Projektion lässt sich als

$$P = V \cdot (V^T \cdot \Sigma^{-1} \cdot V)^{-1} \cdot V^T \cdot \Sigma^{-1} \tag{2.5}$$

ausdrücken. V stellt dabei eine orthogonale Basis der Transformation dar und kann durch Singulärwertzerlegung gewonnen werden. Σ kann während der Berechnung des ICP schrittweise aufgebaut werden, indem die einzelnen Transformationen der gefundenen Punktpaare in sogenannte *Twists* konvertiert werden. Twists sind 6-tupel und können zur Transformationsdarstellung verwendet werden. Dabei muss eine Linearisierung erfolgen; Für Details sei auf [Steinbach 05] verwiesen. Die weiteren Einschränkungen werden laut [Demirdjian 03] auf ähnliche Weise integriert, wobei die Bewertungsfunktion aus Beispielen gelernt wurde. Eine Sequenz von Originalbild und projizierter Posenschätzung ist in Abbildung 2.20 zu sehen.
Das Verfahren beinhaltet mehrere Matrixinversionen, sowie eine Näherung. Dies resultiert in hoher Komplexität bei der Berechnung, die Verfolgung läuft laut [Demirdjian 03] mit 10 Hertz auf einem P4-Rechner 2GHz. Ein größeres Problem stellt allerdings die Trennung von Anpassung und Gelenkeinschränkung dar. Die nachträgliche Korrektur der Pose anhand der Gelenkeinschränkungen erfolgt nicht auf Basis der aktuellen Messung. Dies bedeutet, dass Effekte des ICP-Schritts teilweise rückgängig gemacht werden und somit die Messung verloren geht. Bestätigende Experimente hierzu können in [Steinbach 05] gefunden werden.

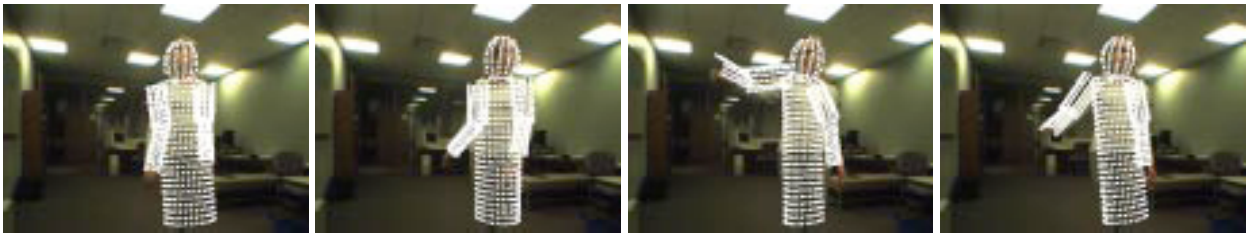


Abbildung 2.20: Beispielkonfigurationen bei einem Modell mit sechs Körperteilen, aus [Demirdjian 03]

2.5.2 Interpretation menschlicher Bewegungen

Im Rahmen der Mensch-Roboter-Interaktion soll die Information über die Pose des menschlichen Kommunikationspartners genutzt werden, um die jeweilige Aktivität des Menschen schätzen zu können. Aktivitätserkennung wird in der Literatur für unterschiedliche Zwecke eingesetzt, und entsprechend vielfältig sind die Forschungsrichtungen und Ergebnisse. Im Rahmen dieser Arbeit beschränkt sich die Anwendung auf die Parametrierung von Roboteraktionen, es soll also keine komplexe Handlungsbeobachtung betrachtet werden; einfache Gesten sind im Allgemeinen ausreichend, um Kommandos wie *Stop* oder Parameter wie *Schneller* zu signalisieren.
Ein großes Anwendungsfeld ergibt sich aus der Überwachung von Räumen. Dies kann mit unterschiedlichem Hintergrund geschehen: Einerseits sind dies Anwendungen aus der Sicherheitstechnik, um Verbrechen erkennen zu können. Andererseits werden auch sogenannte *Smart Rooms* (engl. für: *Intelligente Räume*) aufgebaut, um dem Menschen bei verschiedenen Aufgaben assistieren zu können. Dies können beispielsweise intelligente Konferenzräume sein, die entsprechend der Aufmerksamkeit und Blickrichtung der Teilnehmer Beleuchtung oder Ton steuern (siehe [Chil 06]).
Grundsätzlich lassen sich Ansätze zur Aktivitätserkennung nach den verwendeten Daten bzw. der Sensorik einordnen. Die vorliegende Arbeit basiert auf einer engen Interaktion zwischen Mensch und Roboter ohne weitere Komponenten, es sind also zwei Voraussetzungen zu treffen:

- Als Sensorik und Vorverarbeitung können nur die dem Roboter verfügbaren Sensoren und Ressourcen verwendet werden. Im Raum verteilte Sensoren, sowie zusätzliche Rechenkapazitäten, sind nicht vorhanden.

- Auf dieser Grundlage wird im Rahmen dieser Arbeit ein Ansatz zur 3D-Posenschätzung des Menschen vorgestellt. Die resultierenden Daten über Pose und Bewegung des Menschen können für die Aktivitätserkennung verwendet werden.

Im Weiteren werden also Ansätze zur Aktivitätserkennung betrachtet, die eine Erkennung mit der Sensorik des Roboters zulassen.

Die Erkennung speziell von Zeigegesten wird von Nickel et al. in [Nickel 04a, Stiefelhagen 04, Nickel 04b] basierend auf der dreidimensionalen Verfolgung von Kopf und Händen vorgeschlagen. Die Handpositionen werden transformiert in einen menschzentrierten, zylindrischen Raum (siehe Abbildung 2.21). In diesem Raum werden Zeigegesten mittels mehrerer *Hidden Markov Models (HMMs)* (für eine Einführung zu HMMs siehe [Rabiner 89, Rao 99])

klassifiziert. Als zusätzlicher Eingabewert dient die Blickrichtung. Für die Erkennung wurden Zeigegesten in drei Phasen modelliert: Armbewegung zur Zeigerichtung hin, Halten der Zeigerichtung, und Bewegung weg von der Zeigerichtung. Für jede Phase wurde ein eigenes HMM aufgestellt und trainiert. Die Erkennung der Zeigegesten erfolgt mit dem beschriebenen System robust und zuverlässig. Einen großen Nachteil stellt allerdings die fehlende Erweiterbarkeit und Skalierung des Systems dar: Weitere Gesten müssen entsprechend dem Vorgehen für Zeigegesten von Hand in einzelnen Phasen modelliert und die entsprechenden HMMs aufgestellt und trainiert werden. Es ist also nicht möglich, das System auf weitere Gesten direkt zu trainieren.

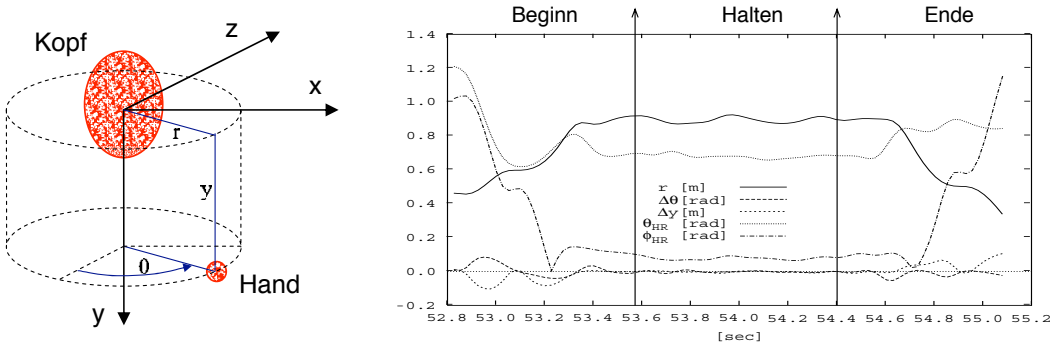


Abbildung 2.21: Erkennung von Zeigegesten durch Klassifikation von Handbewegungen im zylindrischen, menschzentrierten Raum, aus [Nickel 04a]

Ein allgemeinerer Ansatz zur Erkennung und Klassifikation von Gesten auf Basis der Handbewegung wird von Ehrenmann in [Ehrenmann 02] verfolgt. Abbildung 2.22 zeigt den Ablauf des Verfahrens. Die Bewegungen der menschlichen Hand werden in 2D parallel zur Aufnahmeebene des bildgebenden Sensors verfolgt. Nach einer Abtastung der Trajektorie wird für jeden Schritt die Bewegungsrichtung bestimmt und einer von 16 Klassen zugeteilt, die durch gleichmäßige Diskretisierung des 360°-Bereichs bestimmt wurden. Die entstandene Folge von Klassen wird dann zur Klassifikation verwendet, wobei für jede zu klassifizierende Geste ein unabhängiges HMM trainiert wird.
Dieser Ansatz erlaubt die Erkennung von beliebigen Gesten, solange diese so ausgeführt werden, dass sie für den Bildsensor sichtbar sind. Hierin liegt auch der Schwachpunkt des Ansatzes. Die Beschränkung auf Bewegungen in zwei Dimensionen erlaubt nur eine begrenz-

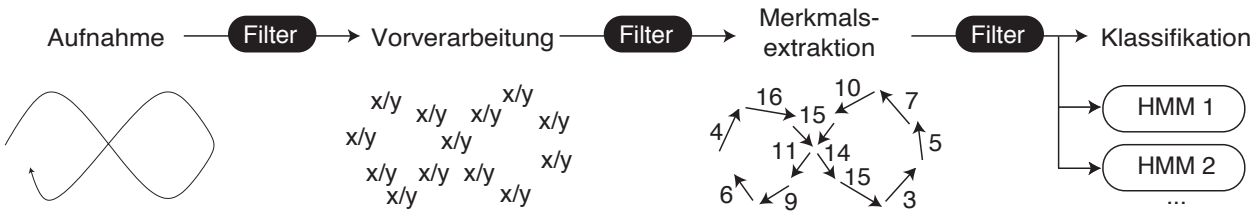


Abbildung 2.22: Dynamische Gestenerkennung auf Basis der Handtrajektorie, aus [Ehrenmann 02]

te Freiheit bei trainierbaren Gesten, und eine Erweiterung auf drei Dimensionen wäre nicht mehr handhabbar. Zusätzlich ergäben sich Mehrdeutigkeiten bei der Richtungsbestimmung. Darüber hinaus wird die Handposition als einziges Merkmal verwendet, es können also keine Körperposen klassifiziert werden.

Das Erkennen komplexer Aktivitäten erfordert zusätzliches Wissen über den Kontext. Kontextwissen wird z.B. von Crowley et al. (siehe [Coutaz 05, Crowley 04]) verwendet, um Mehrdeutigkeiten bei der Interpretation aufzulösen. Dies ist für komplexe Aktivitäten, die nicht allein aufgrund der Bewegung erkannt werden können, unabdingbar. Abhängig ist es allerdings von der Granularität der Bewegungsbeobachtung; je ungenauer diese ist, umso mehr Mehrdeutigkeiten müssen durch zusätzliches Kontextwissen aufgelöst werden. Die Fragestellung für die vorliegende Arbeit umfasst nur einfache Gesten; die Aktivitätsklassifikation mit Hilfe von Kontextwissen soll deshalb nicht weiter betrachtet werden.

2.5.3 Bewertung

In diesem Kapitel wurden aktuelle Ansätze zur Interaktion zwischen Mensch und Roboter auf Basis der Beobachtung und Interpretation menschlicher Bewegungen durch den Roboter dargestellt. Besonderes Augenmerk galt den Verfahren zur Erkennung und Verfolgung der Körperpose, da mit Hilfe dieser Information Rückschlüsse auf Gesten und Aktivitäten des Menschen gezogen werden können.
Verfahren, die nur einzelne Merkmale des Körpers erkennen und diese verfolgen (meist Gesicht und Hände), haben Vorteile bei der Laufzeit. Allerdings ergeben sich signifikante Nachteile:

- Die entsprechenden Merkmale müssen kontinuierlich sichtbar sein. Dreht der beobachtete Mensch beispielsweise das Gesicht von der Kamera weg, kann dieses nicht erkannt werden, und die Verfolgung schlägt fehl.
- Aus einigen ausgezeichneten Punkten kann die Pose nur unvollständig rekonstruiert werden. Es ergeben sich also für die Interpretation Mehrdeutigkeiten, die durch Hintergrundwissen oder Wissen über den Kontext aufgelöst werden müssen.

Um Gesten klassifizieren zu können, sind Verfahren, die die gesamte Körperhaltung verfolgen, vorzuziehen. Solche Verfahren stützen sich auf ein Modell des zu verfolgenden Körpers, das anhand der Messungen nachgeführt wird und eine Schätzung der momentanen Pose enthält. Hier existieren grundsätzlich Ansätze, die mehrere mögliche Konfigurationen gleichzeitig verfolgen (*Multi-Hypothesen-Ansätze*), im Gegensatz zu Verfahren, die eine einzige Schätzung mitführen. Multi-Hypothesen-Ansätze sind robuster bei schlechten oder falschen Messungen; allerdings ist die Anwendbarkeit in Echtzeit eingeschränkt.

2.6 Zusammenfassende Bewertung

Das vorliegende Kapitel gibt einen Überblick über den Stand der Forschung bei Architekturen für flexible Robotersysteme, Repräsentationen von Handlungswissen, für die Abbildung abstrakten Handlungswissens auf konkrete Robotersysteme sowie für die Interaktion mit diesen. Im Speziellen wird die Klassifikation und Erkennung von Gesten und einfachen

Aktivitäten zur Kommandierung und Kommentierung im Rahmen der Mensch-Roboter-Interaktion betrachtet. Die Bewertung der einzelnen existierenden Ansätze erfolgte dabei im Hinblick auf die dieser Arbeit vorangestellten Ziele und Anforderungen. So lassen sich Lücken und Schwächen identifizieren, die in den betrachteten Ansätzen existieren:

- Bei der Entwicklung von Handlungsbeschreibungen wurde bisher wenig Wert darauf gelegt, dass diese zur Laufzeit sowohl um Basisaktionen als auch um zusammengesetzte Handlungen erweitert werden können. Dabei ist die Transparenz und Erklärbarkeit der Repräsentation ein nicht zu unterschätzender Faktor in der Mensch-Roboter-Interaktion, der der Kommunikation und Erklärung von Roboteraktionen und Zuständen dient.

- Es fehlt bisher eine allgemeine Betrachtung der Abbildung von Handlungswissen auf ein konkretes Robotersystem. Dazu gehören Betrachtungen über die Darstellung des Handlungswissens, über die kinematische Abbildung sowie die Abbildung von Zielen und Bedingungen.

- In der engen, auch physischen Interaktion zwischen Mensch und Roboter mit dem Ziel der Kommandierung, Kommentierung und Parametrierung des Roboters existieren keine Ansätze, um die Bewegungen, Gesten und Aktivitäten des Menschen kontinuierlich und in Echtzeit mit den Möglichkeiten des Roboters zu schätzen. Hier wird meist auf die isolierte Verfolgung einzelner Merkmale zurückgegriffen und diese mit verschiedenen Annahmen und Randbedingungen zur Schätzung herangezogen.

Diese identifizierten Lücken werden im Rahmen der vorliegenden Arbeit analysiert, wobei Lösungsansätze vorgestellt und bewertet werden sollen. Das nächste Kapitel gibt zunächst einen Überblick über Vorarbeiten und den zur Lösung eingeschlagenen Weg.

Kapitel 3

Von der Vorführung zur Ausführung

Ein Robotersystem, das flexibel in dynamischen Umgebungen agieren und reagieren können soll, sollte eine klare und mächtige Aufgabenbeschreibung besitzen. Diese Aufgabenbeschreibung soll verständlich und zur Laufzeit veränderbar sein; zusätzlich muss die Möglichkeit einer automatischen Generierung solcher Roboterprogramme bestehen.
Dieses Kapitel gibt einen Überblick über den zur Lösung dieser Problemstellung eingeschlagenen Weg. Dazu werden in Kapitel 3.1 und 3.2 zunächst der Rahmen und bereits vorhandene Systeme und Vorarbeiten erläutert. Kapitel 3.3 stellt dann den Ansatz der Arbeit im Überblick vor.

3.1 Programmieren durch Vormachen

Das Paradigma des Programmierens durch Vormachen beschreibt den Anspruch, ein Robotersystem durch Vormachen und Zeigen einer Aufgabe durch einen Menschen darauf zu trainieren, diese auszuführen. PdV ist ein vielversprechender Ansatz, der in den letzten 10 Jahren sehr viel Aufmerksamkeit erfahren hat. Grundlegende Arbeiten hierzu können in [Ude 96, Friedrich 98, Rogalla 02, Ehrenmann 02, Zöllner 05b] gefunden werden.
Abbildung 3.1 zeigt das Phasenmodell des PdV-Prozesses. Nach [Rogalla 02] gliedert sich der PdV-Prozess in sieben Schritte. Diese sind im Einzelnen:

- Die *Beobachtung der Vorführung* und Aufzeichnung der Daten stellt den Ausgangspunkt des PdV-Prozesses dar. Dabei werden sowohl die Aktionen des Benutzers als auch die resultierenden Veränderungen in der Umwelt sensorisch erfasst.

- Die Rohdaten der Vorführung werden in einem *Segmentierungsschritt* in semantisch beschreibbare Teilstücke zerlegt. Dabei liegt ein generisches Handlungsmodell zu Grunde, dessen elementare Operationen zur Beschreibung der Teilstücke der beobachteten Handlung parametriert werden. Diese elementaren Operationen werden Elementaroperatoren genannt.

- Der *Interpretationsschritt* dient dazu, die parametrierten Elementaroperatoren zu einer Hypothese über den tatsächlichen Handlungsverlauf zusammenzufassen. Auf Basis der in der Umwelt festgestellten Effekte (Zustandsänderungen) und der Elementaroperatoren wird mit Hilfe eines hierarchisch aufgebauten Handlungsmodells eine Hypothese für

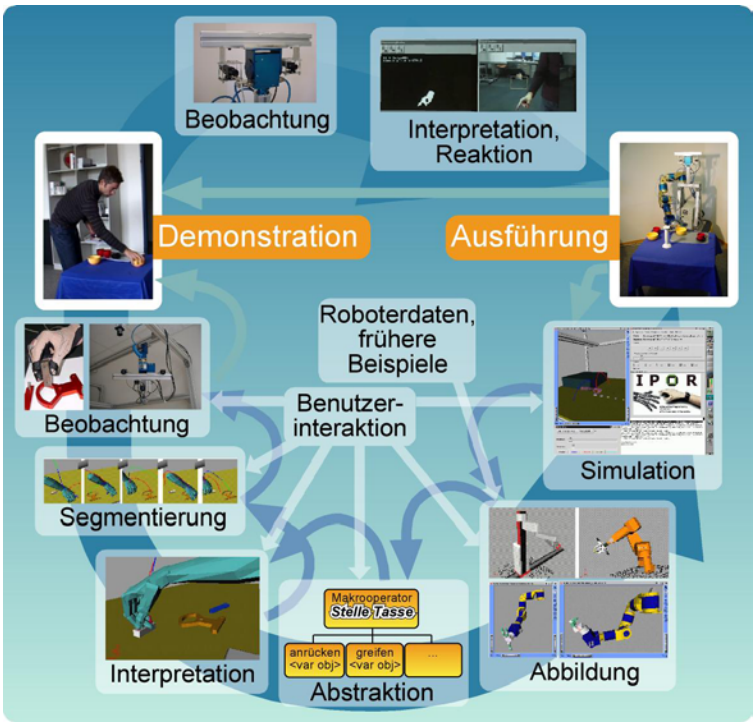


Abbildung 3.1: Programmieren durch Vormachen: Von der Vorführung zur Ausführung. Nach [Rogalla 02]

die Handlung selbst und für die erzielten Effekte aufgestellt. Hier besteht die Möglichkeit für den Benutzer, einzelne Hypothesen und Parameter zu korrigieren.

- Die *Abstraktion* überführt das erzeugte Handlungswissen in eine generische Form, so dass es allgemein für Problemklassen mit ähnlicher Fragestellung anwendbar ist. Zusätzlich muss das neu erworbene Wissen in die vorhandene Wissensbasis konsistent eingepflegt werden.

- Um das erlernte Problemlösungswissen mit einem Roboter umsetzen zu können, muss eine *Abbildung auf das Zielsystem* erfolgen. Diese Abbildung transformiert das abstrakte Handlungswissen in eine für den Roboter ausführbare Form. Dabei sind verschiedene Randbedingungen zu beachten, da Mensch und Roboter nicht den gleichen Funktionsumfang und nicht den gleichen Arbeitsraum besitzen.

- Das so erzeugte Roboterprogramm wird in einer *Simulation* validiert. Diese Validierung dient dazu, offensichtliche Fehler aus einem der vorherigen Schritte zu erkennen. Dies kann sowohl das Handlungswissen selbst (fehlerhafte Handlungsbeschreibung) als auch die Form (Syntax des Roboterprogramms, Arbeitsraum etc.) betreffen.

- Als letzter Schritt folgt die tatsächliche *Ausführung* des Handlungswissen mit dem Roboter. Im Rahmen dieser Arbeit soll nun zusätzlich die Möglichkeit geschaffen werden, die entstandenen Roboterprogramme während der Ausführung durch den Menschen parametrieren zu lassen. Auf diese Weise wird eine weitere Rückkopplung zur Validierung und Adaption der Programme zwischen Roboter und Umwelt geschaffen. Mit

dieser Adaption ist es dann möglich, Programme zur Laufzeit zu verändern (siehe auch Kapitel 1.1).

In jedem einzelnen Schritt dieser Prozesskette können Fehler auftreten. Diese müssen entweder vom System selbständig oder vom Benutzer erkannt werden. In einem solchen Fall wird zum vorherigen Schritt zurückgesprungen (siehe Abbildung 3.1).
Die Erstellung von abstraktem Handlungswissen aus Vorführungen wird ausführlich von Friedrich [Friedrich 98], Rogalla [Rogalla 02], Zöllner [Zöllner 05b] und Pardowitz [Pardowitz 06a, Pardowitz 05] diskutiert.
Im Folgenden werden insbesondere die Schritte von einer abstrakten Handlungsbeschreibung bis zur Ausführung auf einem Robotersystem betrachtet. Dazu werden zunächst die abstrakte Repräsentation von Handlungswissen in Form von Makrooperatoren und die Organisation dieses Wissens in der Wissensbasis beleuchtet. Kapitel 3.2 stellt dann die Ausführungsumgebung, insbesondere den Roboter, dar.
Mit diesen Voraussetzungen wird dann in Kapitel 3.3 der gewählte Ansatz im Überblick dargelegt.

3.1.1 Makrooperator: Abstrakte Handlungsbeschreibung

Während der Vorführung wird der Benutzer mit Hilfe verschiedener Messverfahren beobachtet. Diese Sensorik dient dazu, ein möglichst genaues Bild der Handlungen des Benutzers und der Veränderungen in der Umwelt zu akquirieren. Die Sensorik besteht im Einzelnen aus zwei Datenhandschuhen, die die Fingerstellung des Menschen aufnehmen, sowie zwei magnetfeldbasierten Positions- und Lagesensoren. Des Weiteren werden der vorführende Mensch und die Demonstrationsszene mit Hilfe zweier aktiver Stereokameras beobachtet. Der Aufbau ist in Abbildung 3.2 zu sehen. Diese Vorführumgebung unterscheidet sich wesentlich von der Umgebung der Ausführung, die aus dem Roboter und insbesondere nur seiner eingebauten Sensorik besteht. Die Verwendung invasiver Messverfahren (Datenhandschuhe, magnetfeldbasierte Positions- und Lagesensoren) lässt sich für die Vorführumgebung rechtfertigen, um hohe Genauigkeiten bei der Beobachtung zu erzielen. Für die Ausführung selbst und Parametrierung während der Ausführung durch den Roboter ist der beschriebene Sensoraufwand nicht praktikabel (siehe auch Kapitel 2.5.1).
Aus den aufgezeichneten Daten der Benutzervorführung werden elementare Aktionen extrahiert. Diese beinhalten beispielsweise *Greifen eines Objekts*, *Loslassen*, oder *Lineare Bewegung*. Elementare Aktionen werden auch Elementaroperatoren genannt und stellen die kleinste Einheit für die Analyse dar. Details zur sensorischen Erkennung dieser parametrierbaren Primitive können in [Ehrenmann 02] gefunden werden.
Die detektierten Elementaroperatoren werden analysiert und zu größeren Einheiten zusammengefasst. Aus mehreren Elementaroperatoren bestehende Handlungshypothesen werden als Makrooperatoren bezeichnet. Solche Makrooperatoren können zum Beispiel *Transportbewegungen* oder *Anrückbewegungen* beinhalten. Makrooperatoren werden wiederum hierarchisch zusammengefasst, so dass der Makrooperator auf der höchsten Ebene die vollständige Vorführung umfasst. Zur Unterstützung der Interpretation dienen zusätzlich gesprochene Kommentare des Benutzers [Pardowitz 06b].
Die Wirkungen jedes einzelnen Operators auf die Umwelt werden durch die jeweiligen Vor- und Nachbedingungen beschrieben. Dafür werden die geltenden Bedingungen durchpropa-

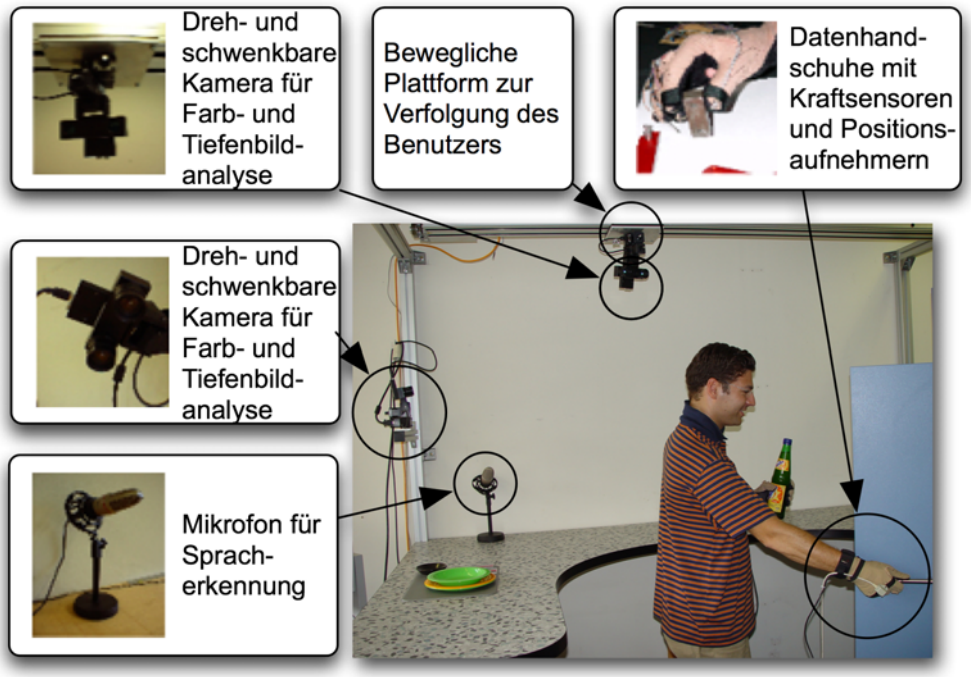


Abbildung 3.2: Aufbau zur Aufnahme von Benutzerdemonstrationen in einer Küchenumgebung, aus [Pardowitz 05]

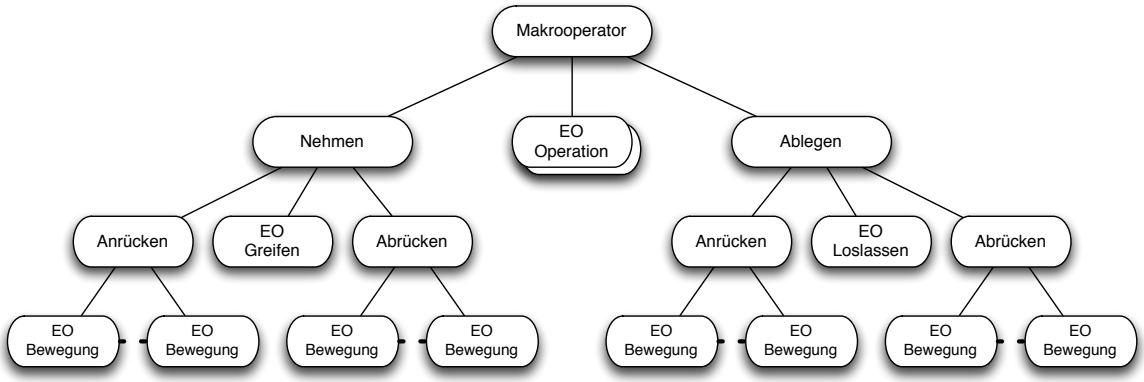


Abbildung 3.3: Schematische Darstellung eines Makrooperators, nach [Pardowitz 05]. Auf der obersten Ebene fasst ein Operator die Vorführung zusammen, während die hierarchisch untergeordneten Ebenen jeweils Teilstücke der Vorführung umfassen. Jeder Teilknoten des Makrooperators kann wiederum als vollständiger Makrooperator aufgefasst werden.

giert, so dass für jeden Makrooperator auf jeder Ebene die geltenden Vor- und Nachbedingungen zur Verfügung stehen. Auf höchster Ebene werden dadurch die Effekte der Handlung beschrieben.
Bedingungen werden berechnet aus den Änderungen der Umwelt während der Demonstration. Das System geht von einer *geschlossenen Welt* aus, es wird also angenommen, dass alle Änderungen der Umwelt durch den Benutzer während der Vorführung entstehen. Umweltzustände werden dargestellt durch Ausdrücke der Prädikatenlogik erster Ordnung. Die Beschreibung eines Umweltzustands setzt sich zusammen aus einzelnen Basiselementen, die geometrische Relationen zwischen Objekten der Szene beschreiben. Diese Relationen können unär, binär oder ternär sein, Beispiele sind *Öffnungswinkel einer Tür*, *Füllstand* (unäre Relationen), *auf*, *unter*, *rechts von* (binär), oder *zwischen* (ternär).
Mit Hilfe dieser Handlungsbeschreibung kann ein breites Spektrum an Handlungen abgebildet werden. Die Erkennung und Modellierung sind auf Handhabungsaufgaben im Haushaltsbereich ausgelegt. Insbesondere können Transportaufgaben (engl.: *Pick & Place*), Manipulation von Objektzuständen (z.B. Kühlschrank öffnen), und Werkzeugbenutzung (einschenken von Flüssigkeit aus einer Flasche in ein Glas) erkannt und dargestellt werden.

3.1.2 Organisation von Handlungswissen

Makrooperatoren werden in einer Wissensbasis organisiert und geordnet. Dazu werden verwandte Makrooperatoren gefunden und miteinander verglichen. Nach [Pardowitz 05] können dabei insbesondere Ordnungsabhängigkeiten zwischen Teilen der Vorführung gelernt werden, wenn mehrere Vorführungen der gleichen Aufgabe vorliegen.
Auf diese Weise kann die totale Ordnung der Teilaktionen innerhalb einer Handlungsrepräsentation teilweise aufgelöst werden [Zöllner 05a]. Die resultierende teilgeordnete Darstellung wird auch als *Handlungsvorranggraph* oder *Handlungspräzedenzgraph* (engl.: *Task Precedence Graph, TPG*) bezeichnet. Ein solcher Handlungspräzedenzgraph ist ein gerichteter Graph $P = (M, N)$, wobei M die Menge der Teilhandlungen darstellt. $N \subset M \times M$ stellt die Menge der Präzedenzrelationen dar, die für die gegebene Handlung gelten muss. Gilt beispielsweise $(s_1, s_2) \in N$ mit $s_1, s_2, s_3 \in M$, so muss die Aktion s_1 abgeschlossen sein, bevor Aktion s_2 begonnen werden darf, während s_3 von s_1, s_2 unabhängig ist und zu einem beliebigen Zeitpunkt ausgeführt werden kann. Es wird also eine Teilordnung der Teilhandlungen aufgebaut. Der zugehörige Handlungspräzedenzgraph ist in Abbildung 3.4 aufgezeigt.

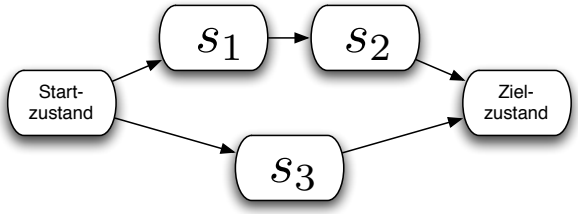


Abbildung 3.4: Handlungspräzedenzgraph für $s_1, s_2, s_3 \in M$ und die Relation $(s_1, s_2) \in N$

Für die Ausführungseinheit bedeutet die vorgegebene Teilordnung der Handlungsteile die Möglichkeit, zwischen verschiedenen Ausführungsreihenfolgen zu wählen. Dieser zusätzliche

Freiheitsgrad gibt einerseits Raum zur Optimierung der Ausführung, beispielsweise nach zeitlichen oder anderen Kriterien. Andererseits muss für diese Entscheidung üblicherweise zusätzlicher Aufwand in der Modellierung und auf sensorischer Seite betrieben werden, um diese Entscheidungen treffen zu können. Die Modellierung und Umsetzung dieser Entscheidungskomponente wird in Kapitel 6 ausgeführt.

3.2 Ausführungsumgebung

Zur Ausführung und Interaktion stand der mobile Assistenzroboter ALBERT II als Experimentierplattform zur Verfügung. Abbildung 3.5 zeigt den Roboter in einer Küchenumgebung, die ein mögliches Einsatzgebiet solcher Roboter darstellt.

Abbildung 3.5: Der mobile Assistenzroboter ALBERT II in einer möglichen Einsatzumgebung.

Der Roboter ALBERT II besteht aus einer mobilen Plattform und einem starren Oberkörper, dessen Äußeres der Uniform eines Butlers entfernt nachempfunden ist. An dieser Oberkörper-Konstruktion sind zusätzliche Sensoren und Aktoren angebracht, die zur Wahrnehmung und Manipulation der Umwelt verwendet werde können. Im Oberkörper befinden sich zwei Computer mit jeweils *Pentium 3 Dual 800 MHz*-Prozessoren. Die Sensorik und Aktorik besteht im Einzelnen aus folgenden Komponenten:

- Zur Umwelterfassung ist der Roboter mit einer Stereo-Farbkamera der Firma *Videre Design* ausgestattet [Vid 06]. Diese kann zur Objekterkennung und zur Menschenerkennung verwendet werden. Die Kamerabilder können in einer Auflösung bis 1280×1044 Pixel ausgelesen werden.

- Der Sensorkopf kann um eine Tiefenbildkamera (engl.: *Time-of-Flight Camera*) erweitert werden. Diese Kamera vom Typ *Swissranger* der Firma CSEM (siehe [CSEM 06]) besitzt eine Auflösung von 160×124 Pixeln und eine Tiefenauflösung von 8 Bit. Je nach konfiguriertem Messbereich beträgt die Genauigkeit der Tiefenmessungen wenige Millimeter bis wenige Zentimeter. Die Kameras sind auf einer Schwenk-Neige-Einheit montiert und können flexibel ausgerichtet werden. Details finden sich in Anhang A.2.

- Zur Manipulation ist der Roboter mit einer 3-Finger Hand der Firma Barrett ausgestattet [Bar 06]. Diese Hand besitzt vier Freiheitsgrade: Jeder Finger kann über einen Freiheitsgrad unabhängig gebeugt werden, zusätzlich können die äußeren beiden Finger gekoppelt am Ansatz geschwenkt werden, so dass sich die Möglichkeit für verschiedene Griffarten ergibt.

- Zur Messung der an der Hand angreifenden Kräfte wird ein nachgiebiger Kraft-Momenten-Sensor verwendet (siehe [Meusel 01]). Dieser kann Kräfte bis $160\,N$ und Momente bis $2,8\,Nm$ messen, so dass zum einen Objektgewichte gemessen werden können, aber zum anderen auch eine kraftgeregelte Bewegung der Roboterhand möglich ist.

- Der Arm des Roboters besitzt sieben Freiheitsgrade und ist aus Armelementen der Firma Amtec [Amtec 00] aufgebaut. Der Arm ist robust ausgelegt, so dass eine Tragkraft von bis zu 5 Kilogramm möglich ist.

- Die mobile Plattform "Odette 2" verfügt über einen Differentialantrieb und einen Laserscanner der Firma SICK [SICK 06]. Die Navigation und Lokalisierung wird auf Basis einer a priori bekannten Umgebungskarte ausgeführt.

- Zentral im Roboter ist ein berührungssensitiver Bildschirm angebracht. Dieser dient sowohl als Ausgabe- als auch als Eingabekanal. Relevante Informationen für den Benutzer können hier dargestellt werden, er kann aber auch zur Eingabe von Kommandos oder Parametern verwendet werden.

- Im Körper des Roboters ist ein Lautsprecherpaar untergebracht, mit dessen Hilfe Sprachausgaben erzeugt werden können. Gleichzeitig besitzt der Roboter einen Empfänger für ein Funkmikrophon, so dass auch Spracheingaben mit einem tragbaren Mikrophon möglich sind.

3.3 Interaktive Erstellung und Ausführung von Handlungswissen

Die Prozesskette des Programmierens durch Vormachen besteht aus Beobachtung, Segmentierung, Interpretation, Abstraktion, Abbildung, Simulation und Ausführung. Die ersten vier Schritte, die zu einer abstrahierten Handlungsbeschreibung führen, sind in der Literatur ausführlich beschrieben [Ehrenmann 01a, Dillmann 99, Kang 97]. Auch die Ausführung von Roboterprogrammen durch einen Serviceroboter wurde betrachtet [Simmons 98, Jensen 02a, Simmons 03]. Eine Lücke klafft dagegen im Abbildungsschritt (engl.: *Mapping*). Dieser ist für spezielle Probleme gelöst worden. Eine allgemeingültige, generelle Betrachtung der Abbildung von abstraktem Handlungswissen auf Serviceroboter ist dem Autor nicht bekannt.

Es wird daher im Folgenden untersucht, welche Komponenten und Verfahren notwendig sind, um wirkungsvoll und effizient das abstrakt vorliegende Handlungswissen für die Ausführung auf einem Serviceroboter aufzubereiten. Besonderes Augenmerk wird dabei auf drei Teilbereiche gelegt, die für die erfolgreiche Abbildung eminent wichtig sind; dies sind Erzeugung, Parametrierung und Ausführung von Handlungswissen. Abbildung 3.6 fasst diese zusammen.

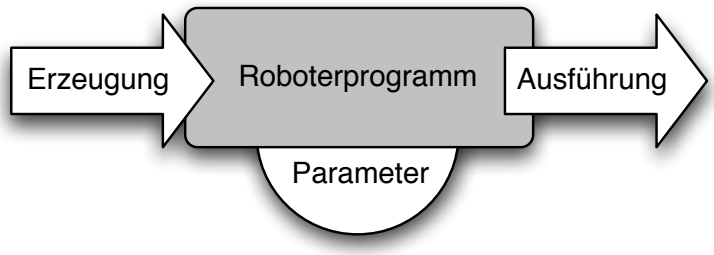


Abbildung 3.6: Darstellung, Parametrierung, Erzeugung und Ausführung von Handlungswissen für mobile Serviceroboter.

3.3.1 Repräsentation von Handlungswissen

Anforderungen und Eigenschaften einer für die Fragestellung geeigneten Handlungsbeschreibung für den Roboter werden untersucht. Eine hierarchische Handlungsbeschreibung wird vorgestellt, die sich einerseits aus abstraktem Handlungswissen erzeugen und andererseits auch manuell generieren lässt. Die Handlungsbeschreibung wird eingebettet in eine Ausführungskomponente, die im Rahmen einer komponentenbasierten Architektur für den Roboter vorgestellt wird: Handlungsbeschreibung, Ausführungskomponente und Architektur sind eng gekoppelt und lassen sich deshalb nicht unabhängig entwerfen und betrachten.
Die Handlungsbeschreibung wird in Form eines *Hierarchischen Aufgabennetzwerkes* repräsentiert. Diese Darstellung ermöglicht es einerseits, Teilhandlungen sinnvoll zu verketten. Andererseits ist sie der abstrakten Handlungsbeschreibung der Makrooperatoren hinreichend ähnlich, um eine erfolgreiche Abbildung gewährleisten zu können. Diese *Flexible Programme* genannte Handlungsbeschreibung wird von einer echtzeitfähigen Ausführungskomponente interpretiert und in Roboteraktionen umgesetzt. Dabei werden Konstrukte wie bedingte Ausführung, Schleifen, parallele Ausführung mehrerer Programme usw. zur Verfügung gestellt. Insbesondere kann zwischen enthaltenen Alternativlösungen gewählt werden. Diese Alternativen können beispielsweise aus dem ursprünglichen Makrooperator übernommen worden sein.
Eine verteilte, komponentenbasierte Systemarchitektur wird vorgeschlagen. Die Kommunikation zwischen den Komponenten ist streng ereignisbasiert, und es existieren unterschiedliche Komponenten für Handlungskontrolle, Umweltmodell, Ausführung von Aktionen mit Aktorik und Sensorik, sowie für die Kommunikation.

3.3.2 Der Abbildungsprozess

Auf Basis der Definition von Makrooperatoren und der Flexiblen Programme wird der Abbildungsprozess betrachtet. Dieser besteht aus den Schritten *Strukturabbildung*, *Abbildung der Elementaroperationen*, *Regelbasierte Anpassung*, *Ressourcenverwaltung* und *Serialisierung/Parallelisierung von Teilaufgaben*. Es lässt sich zeigen, dass diese Abbildung auf andere Roboter übertragen werden kann. Um dies zu gewährleisten, wird das Hintergrundwissen, das über den ausführenden Roboter benötigt wird, explizit untersucht und definiert. Dadurch wird eine klare Abgrenzung von Verfahren und Daten im Abbildungsprozess erreicht, was die Übertragbarkeit gewährleistet und erleichtert.

Die Abbildung beruht also auf Modellwissen über das Zielsystem, sowie den a priori bekannten Eigenschaften der Makrooperatoren und der Handlungsbeschreibung für den Roboter. Beide Beschreibungen werden systematisch untersucht und gegenübergestellt. Zusammen mit dem Wissen über das Zielsystem wird dann ein regelbasierter Übersetzungsprozess vorgeschlagen, mit dem ausführbare Roboterprogramme aus Makrooperatoren und Hintergrundwissen generiert werden können. Das zu Grunde liegende Regelsystem zur Erzeugung von Roboterprogrammen enthält dafür einerseits für die Ausführung *notwendige* Regeln, andererseits können auch nützliche, aber nicht notwendige, also *optionale* Regeln hinzugefügt werden. Dies geschieht beispielsweise, um Sprachausgaben des Roboters zum aktuellen Zustand und Ziel zu erzeugen.

Die erzeugte Handlungsbeschreibung kann graphisch dargestellt werden, um zusätzlich eine qualitative Validierung zu erlauben. Diese Darstellung wird mit Hilfe eines Editors für Flexible Programme ermöglicht, der im Rahmen der vorliegenden Arbeit entwickelt wurde.

3.3.3 Interaktion und Parametrierung

Durch die großen Unterschiede im Aufbau und in den Möglichkeiten zwischen vorführendem Mensch und ausführendem Roboter kommt es immer wieder zu Mehrdeutigkeiten oder Kollisionen bei der automatischen Übertragung von Handlungswissen. Diese Fälle treten selbst beim Vorführen und Lernen von Handlungen zwischen Menschen auf, und werden meistens durch Rückfragen oder Wiederholungen aufgelöst.

Ein Interaktionskanal zwischen Mensch und Roboter wird vorgeschlagen, der zur Parametrierung und Adaption von Handlungswissen genutzt werden kann. Eine Beobachtungskomponente nimmt dabei kontinuierlich die vollständige Körperpose des Menschen auf. Auf Basis dieser Daten können Kommandos, Gesten oder zusätzliche Informationen extrahiert und gewonnen werden. Dazu wird der Mensch mit mehreren Sensoren vom Roboter beobachtet, deren Information für eine modellbasierte Verfolgung der Pose genutzt wird. Das geometrische Körpermodell des Menschen wird durch mehrere Zylindern approximiert, die über verschiedene Gelenktypen verknüpft werden können und so verschiedene Körperposen nachbilden können. Aus diesen Daten lassen sich Gesten und andere Aktivitäten klassifizieren, die zur Parametrierung und Kommandierung des Roboters dienen können.

Die Beobachtung erfolgt mit Hilfe aller dem Roboter zur Verfügung stehenden Sensoren. Die Verwendung und Fusion unterschiedlicher Sensoren ist von besonderer Bedeutung, da die vollständige Körperpose des Menschen rekonstruiert werden soll: Verschiedene Sensoren haben unterschiedliche Messbereiche und Genauigkeiten, und ein Körpermodell mit mehreren Gelenken und bis zu drei Freiheitsgraden pro Gelenk besitzt eine hohe Anzahl von Freiheitsgraden, die bestimmt werden müssen. Die Sensorinformation wird gesammelt und mit einem iterativen Verfahren zur Anpassung des Modells verwendet. Dabei können einzelne Sensoren unterschiedlich stark gewichtet werden, abhängig von der Genauigkeit der jeweiligen Daten.

Die Interpretation der Bewegungen des Menschen zur Klassifikation von Gesten und Aktivitäten basiert auf einer Analyse der beobachteten Bewegungen. Menschliche Bewegungen haben eine sehr hohe Aussagekraft für die Bestimmung der Aktivitäten; so werden unter anderem Zeigegesten, Laufen, Sitzen oder Winken anhand der Bewegung des Körpers erkannt und klassifiziert.

3.4 Zusammenfassung

Mit den genannten Teilbereichen werden das Format, die Ausführung, sowie die Erzeugung und Parametrierung von Roboterprogrammen beleuchtet. Ziel ist es also, aus abstrakten Handlungsbeschreibungen ausführbare Roboterprogramme zu erzeugen, die darüber hinaus editierbar und erweiterbar sind. Die Erweiterung und Anpassung soll sowohl über graphische Eingabemethoden als auch über eine intuitive Mensch-Roboter-Schnittstelle möglich sein.

Kapitel 4

Wissensrepräsentation und Modellierung

Im vorliegenden Kapitel werden die vorgeschlagenen Modelle für die Systemarchitektur und das grundlegende Modell der Ausführungsbeschreibung dargelegt. Diese gliedert sich in die Repräsentation selbst und die Modellierung von Relationen. Die verwendeten Modelle zur Beobachtung und Interpretation der Bewegungen des Menschen werden in den Kapiteln 4.4 und 4.5 vorgestellt. Abbildung 4.1 visualisiert diesen Aufbau.

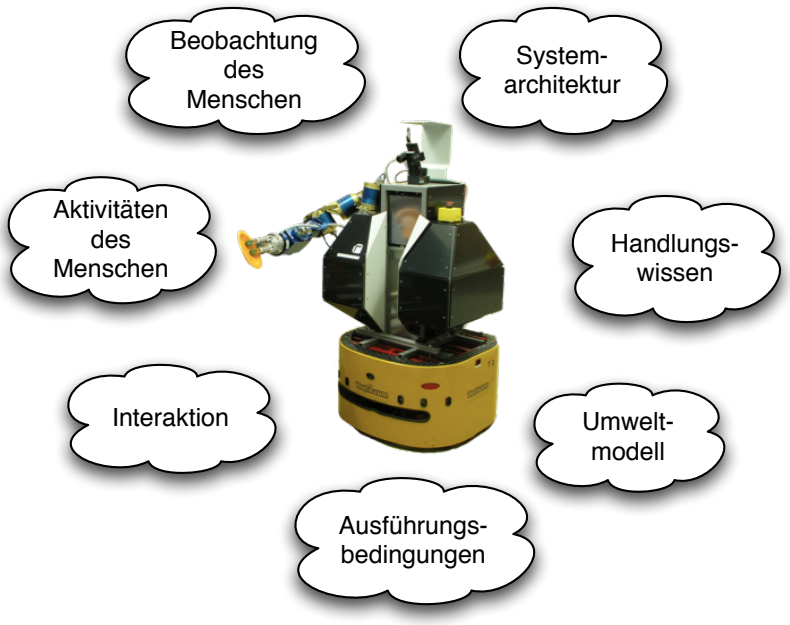


Abbildung 4.1: Betrachtetes Modellwissen eines Roboters

4.1 Systemarchitektur

Die Architektur eines Robotersystems bestimmt die Funktionalität und Benutzbarkeit eines solchen Systems. In einem Roboter werden verschiedenste Komponenten benötigt, die von komplexer Aktorik über Sensorik bis zu Netzwerkkomponenten und Rechnern reichen. Im Folgenden wird die verwendete Architektur dargestellt.

4.1.1 Anforderungen an eine Architektur für Serviceroboter

Zusätzlich zu den Sensoren und Aktoren lassen sich eine Reihe weiterer Anforderungen für lernende Serviceroboter definieren; ein Überblick wurde bereits in Kapitel 2.2.1 gegeben.
Dies ist zum einen die *Reaktivität*, also die Fähigkeit des Roboters, spontan auf äußere Einflüsse zu reagieren. Beim Menschen findet sich die zugehörige Entsprechung auf der Ebene der Reflexe, und wird vor allem zum eigenen Schutz vor plötzlichen Gefahren oder Umweltänderungen verwendet. Diese Aktionen finden ohne explizite Planungsphase statt.
Im Gegensatz dazu steht die Forderung nach *Deliberation*, also die Fähigkeit des Roboters, auch langfristige Ziele gerichtet zu verfolgen. Dies beinhaltet die Fähigkeit zur Planung, Dekomposition von Aufgaben, sowie zur Verfolgung des Fortschritts und Überwachung der Einzelschritte.
Das Wissen des Roboters über die Umwelt zum einen und über Problemlösungsstrategien zum anderen muss so abgelegt und organisiert sein, dass es sich jederzeit erweitern und auch verändern oder sogar revidieren lässt. Diese *Erweiterbarkeit und Adaptivität* folgt unmittelbar aus der Definition der Zielanwendung: Ein solches System soll in menschenzentrierten Umgebungen arbeiten, zusammen mit Menschen und in dessen Arbeitsraum und Kontext. Dadurch muss sich der Roboter immer wieder auf neue Situationen, Objekte, Kontexte und Aufgaben einstellen. Dies erfordert eine ständige Validierung und Erweiterung der Wissensgrundlage.

4.1.2 Verwendete Architektur

Eine Systemarchitektur, mit der ein mobiler Serviceroboter gesteuert werden soll, muss dazu drei grundsätzliche Anforderungen erfüllen: Sie muss die Steuerung und Regelung der Aktorik und Sensorik beinhalten, eine Darstellung der Umwelt und des Roboters selbst besitzen und fortschreiben, sowie Handlungswissen auf allen Planungsebenen repräsentieren und integrieren. Dies beinhaltet sowohl die reaktiven als auch die deliberativen Schichten.
Die vorgeschlagene Systemarchitektur orientiert sich an dieser Einteilung und unterscheidet dafür die funktionalen Bereiche *Hardware-Abstraktion*, *Umweltmodell* und *Repräsentation und Ausführung von Handlungswissen*. Dabei werden folgende Komponenten definiert: In den *Hardware Agenten* ist sämtliche Funktionalität der Sensorik und Aktorik gekapselt. Für jede anzusteuernde Einheit existiert dabei ein Agent, z.B. jeweils für den Arm mit sieben Freiheitsgraden, für die Dreifingerhand und für die Sprachausgabe. Das Handlungswissen liegt in Form sogenannter *Flexibler Programme* vor und wird von entsprechenden Interpretern bearbeitet und ausgeführt. Format und Aufbau dieser Flexiblen Programme werden in Kapitel 4.2 ausführlich vorgestellt.
Abbildung 4.2 zeigt den Aufbau der Architektur und die Verknüpfung der einzelnen Teilkomponenten. Im Einzelnen ist das System mit folgenden Komponenten modelliert:

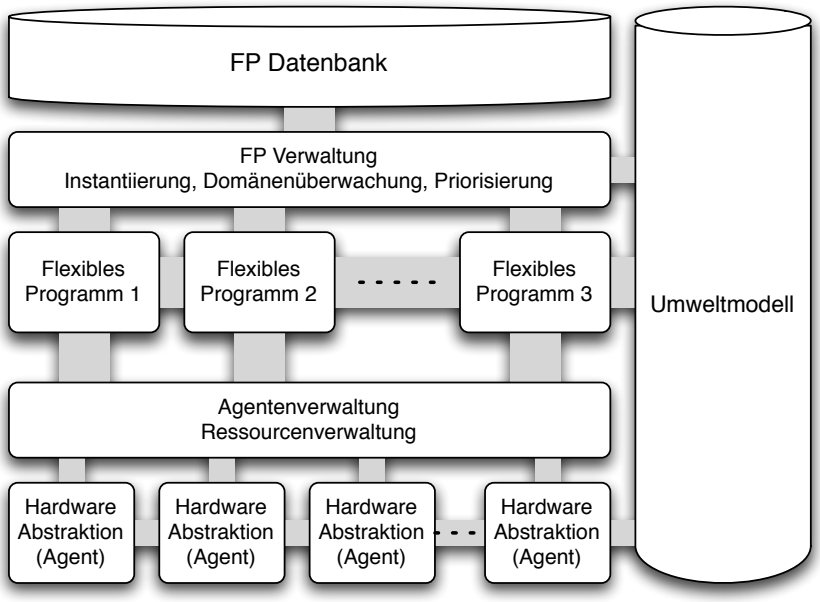


Abbildung 4.2: Systemarchitektur mit Handlungswissen als Flexible Programme, Hardware Abstraktion in Form von Agenten, Umweltmodell und Kommunikation

- Die einzelnen Sensor- und Aktorkomponenten werden durch Hardware-Agenten abstrahiert. Solche Agenten können reale oder auch virtuelle Sensoren und Aktoren darstellen. Agenten werden auch als Ressourcen bezeichnet und genauso behandelt, müssen also von auszuführenden Roboterprogrammen angefragt, belegt und freigegeben werden.

- Die Agentenverwaltung stellt die Mechanismen zur Ressourcenverwaltung zur Verfügung und überwacht alle Agenten. Alle Anfragen an Agenten werden von der Agentenverwaltung gefiltert, um nicht autorisierte Kommandos zu identifizieren und zu blockieren.

- Flexible Programme (Abkürzung: *FP*) kodieren das dem Roboter bekannte Handlungswissen. Diese Handlungen werden ausgeführt von entsprechenden Interpretern. Es ist möglich, dass zur gleichen Zeit mehrere solche Interpreter existieren und verschiedene Handlungen zur gleichen Zeit ausgeführt werden. Eine detaillierte Beschreibung der Flexiblen Programme findet sich in Kapitel 4.2.

- Die FP Verwaltung administriert die auszuführenden Flexiblen Programme. Dies umfasst zum einen die Filterung und Überprüfung aller Nachrichten für Flexible Programme. Zum anderen nimmt die FP Verwaltung die Handlungsauswahl vor. Hierfür wird aufgrund eingehender Signale und des aktuellen Umwelt- und Roboterzustands (der aktuellen Domäne) das jeweils passendste Roboterprogramm ausgewählt, parametriert und gestartet. Existierende FPs können neu priorisiert werden. Modellierung und Mechanismen dieser Auswahl und Priorisierung werden in den Kapiteln 4.3.1 und 6 detailliert.

- Das Umweltmodell beinhaltet sowohl Daten der Umwelt des Roboters als auch den eigenen internen Zustand. Die Daten sind entsprechend dem Entwurfsmuster *Tafel*

(engl.: *Blackboard*) organisiert (siehe [Gamma 04]). Es kann damit von allen beteiligten Instanzen sowohl geschrieben als auch gelesen werden.

- Die Kommunikation zwischen den beteiligten Modulen wird mittels einer zentralen Nachrichtenverteilung vorgenommen, bei der sich Module für bestimmte Nachrichtentypen an- und abmelden können.

Nachfolgend werden die Eigenschaften dieser Komponenten im Einzelnen beschrieben.

Die Kommunikationsschicht

Grundlegend für eine verteilte Architektur ist der Kommunikationsrahmen. Zur Kommunikation zwischen den einzelnen Komponenten wird hier ein spezielles Nachrichtenformat verwendet, welches im weiteren *Notifikation* (engl.: *Notification*) genannt wird. Notifikationen sind Strukturen, in denen die meiste Information als einfacher Text abgelegt ist. Dies unterstützt sowohl den Entwurf als auch die Fehlersuche.
Notifikationen gehören immer einer von drei Klassen an:

- *Ereignisse* (engl.: *Events*) fließen immer den Flexiblen Programmen zu. Sie stammen entweder von Hardware Agenten oder aus externen Quellen (z.B. eine Benutzerschnittstelle). Ereignisse entsprechen Datenfluss von der Sensor-Aktor-Ebene zur Kontrollebene.

- *Aktionen* (engl.: *Actions*) fließen immer in Richtung Agenten. Sie stammen meistens von der Ausführungsschicht der Flexiblen Programme. Aktionen entsprechen einem Datenfluss von der symbolischen Kontrollebene auf die Sensor-Aktor-Ebene.

- *Anfragen* (engl.: *Requests*) beinhalten Ressourcenanfragen und werden deshalb immer der Agentenverwaltung übergeben. Anfragen können einer Belegung, Freigabe oder versuchten Belegung entsprechen.

Die Spezifikation einer Notifikation findet sich in Tabelle 4.1.

Kopf	
Typ	Ereignis, Aktion, Anfrage
Name	Nachrichtenspezifikation
Zeit	Datum und Uhrzeit der Erstellung
Sender	Eindeutige Identifikation des Senders
Empfänger	Eindeutige Identifikation(en) des (der) Empfänger
Körper: Schlüssel-Wert-Paare	
Schlüssel	Wert

Tabelle 4.1: Definition einer Notifikation

Ressourcenverwaltung

Als Agenten modellierte reale oder virtuelle Sensoren und Aktoren sind in der Architektur als Ressourcen vorhanden. Diese können von einzelnen Flexiblen Programmen verwendet werden, und müssen zu diesem Zweck entsprechend belegt und freigegeben werden. Dies stellt sicher, dass keine Konflikte entstehen, wenn mehrere Flexible Programme gleichzeitig aktiv sind. Die Ressourcenverwaltung wird von der Agentenverwaltung vorgenommen und beinhaltet die Statusüberwachung der Agenten und Zugehörigkeiten, Sicherheitsaspekte und die Überwachung der Kommunikation. Letzteres stellt sicher, dass keine Notifikationen von nicht autorisierten Komponenten an einen Agenten verteilt werden. Abbildung 4.3 zeigt ein Beispiel für eine erfolgreiche Ressourcenzuteilung.

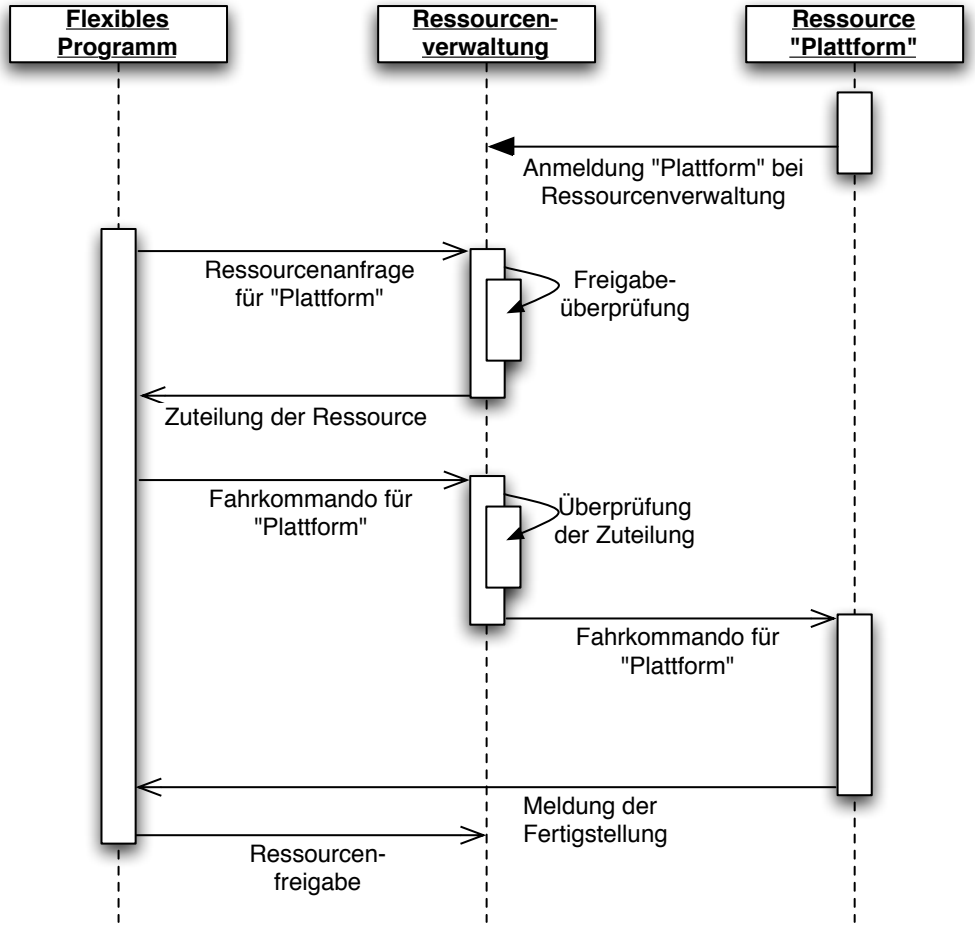


Abbildung 4.3: Beispiel einer Ressourcenanfrage als UML-Sequenzdiagramm. Eine Ressource muss sich zuerst registrieren, dann kann sie angefragt und zugeteilt werden. Nur an eine regulär zugeteilte Ressource können Kommandos geschickt werden; dies wird von der Ressourcenverwaltung sichergestellt, die jedes Kommando überprüft.

Hardware Agenten

Jeder Agent wird über eine eindeutige Identifikationsnummer und einen Agententyp referenziert. Der Typ eines Agenten legt die möglichen Aktionen fest, indem die Menge der akzeptierten Notifikationen definiert wird.
Die möglichen Aktionen eines Agenten sind kontextunabhängig durch die darunterliegende Hardware festgelegt. So sind mögliche Aktionen z.B. der Drei-Finger-Hand *Schließe Hand*, *Zirkulärer Griff mit Öffnungswinkel* x oder *Griffsteuerung mit Parametern* x, y, z, a. Die Hand kann also auf verschiedenen Abstraktionsebenen über die Agentenschnittstelle gesteuert werden.

Unterbrechung und Ausnahmebehandlung

Es existieren zwei verschiedene Fälle, in denen laufende Aktionen eines oder mehrerer Agenten unterbrochen werden müssen:

- Das Ziel hat sich während der Ausführung geändert. Dies kann sowohl durch externe Vorgaben (z.B. menschliche Intervention) als auch durch interne Vorgänge ausgelöst werden.
- Eine Ausnahmesituation tritt auf. Dies kann z.B. durch die Erkennung einer Kollisionsgefahr (physischer oder anderer Art) auftreten, oder durch externe Einflüsse (Benutzerinteraktion).

Die Unterbrechung im Ausnahmefall wird von der Agentenverwaltung direkt in den Agenten vorgenommen, so dass auch laufende Aktionen sofort angehalten werden. Die Kommunikation wird hierbei nicht über Notifikationen, sondern über direkte Aufrufe gelöst, um ohne Verzögerung eingreifen zu können. Bei einer Zieländerung hingegen ist das Vorgehen dem Agenten überlassen, um ein sicheres Umschalten der Ziele zu gewährleisten.

Zusammenfassung

Die vorgeschlagene Architektur ist demnach eine verteilte, hybride, ereignisbasierte Steuerungsarchitektur (siehe auch [Knoop 04]). Es wird unterschieden zwischen Agenten (ausführende Komponenten), Flexiblen Programmen und entsprechenden Interpretern (Handlungswissen), sowie dem Umweltmodell (realisiert mit einem Blackboard). Die Kommunikation erfolgt durch Notifikationen, welche entweder ein Ereignis, eine Aktion oder eine Anfrage darstellen.

4.2 Flexible Programme: Hierarchische Repräsentation von Handlungswissen

Der Repräsentation des Handlungswissens des Roboters kommt eine zentrale Bedeutung innerhalb der vorliegenden Arbeit zu. Abbildung 4.4 zeigt den Zusammenhang zwischen Lernen, Ausführung und Adaption des gelernten Handlungswissens.
Im Folgenden werden zunächst noch einmal die Anforderungen erläutert, die an die Repräsentation des Handlungswissens nach den gegebenen Voraussetzungen gestellt werden. Anschließend werden Struktur und Aufbau der Flexiblen Programme detailliert dargestellt.

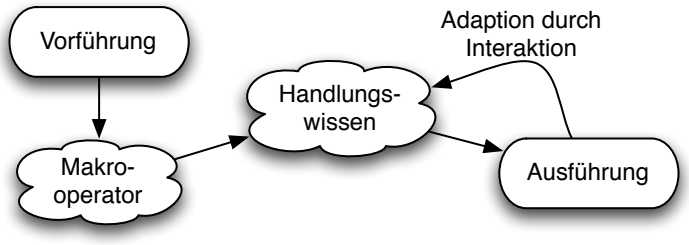


Abbildung 4.4: Handlungswissen des Roboters zwischen Lernen, Adaption und Ausführung

4.2.1 Anforderungen an die Repräsentation des Handlungswissens

Folgende Anforderungen können aus den Voraussetzungen extrahiert werden und gelten für den vorgestellten Entwurf:

- Komplexe Handlungen sollen repräsentierbar und lesbar darzustellen sein. Dabei sollen Handlungen sowohl aus rein sequentiellen als auch aus parallelen Teilhandlungen aufbaubar sein.
- Diese Teilhandlungen sollen wiederverwendbar und parametrierbar sein. Diese Forderung ermöglicht die Wieder- und Weiterverwendung bereits akquirierten Wissens in neuen Handlungen. Hieraus ergibt sich unmittelbar, dass komplexe Handlungen aus einfacheren Teilhandlungen aufgebaut sein sollen.
- Zur Laufzeit soll auf Veränderungen in der Umwelt oder unvorhergesehene Ereignisse und Zustände reagiert werden können. Dies bedeutet, dass die Handlungsbeschreibung Bedingungen und Verzweigungen des Programms repräsentieren können muss.
- Roboterprogramme sollen zur Laufzeit änderbar und parametrierbar sein. Dies kann durch den Roboter oder durch einen Benutzer geschehen. Dadurch muss die Handlungsrepräsentation direkt ausführbar sein, und nicht eines weiteren Übersetzungsschritts bedürfen.
- Handlungen, die durch Programmieren durch Vormachen gelernt wurden, und somit in Form von Makrooperatoren vorliegen, sollen in diese Repräsentation überführbar sein. Die Repräsentation von Handlungswissen für den Roboter muss also mindestens die gleiche Mächtigkeit (in Form von Kontrollstrukturen) besitzen wie die gelernte Beschreibung der Makrooperatoren. Darüber hinaus sollen beide Strukturen ineinander überführbar sein.

Die im Folgenden vorgestellte Repräsentation von Roboterprogrammen, genannt Flexible Programme, erfüllt diese Anforderungen.

4.2.2 Struktur des Handlungswissens

Ähnlich zu TDL (siehe Kapitel 2.3.2) werden Flexible Programme als *Hierarchische Aufgabennetzwerke* (engl.: *Hierarchical Task Network, HTN*) dargestellt. Das Aufgabenwissen einer Handlung hat hierbei Baumstruktur (siehe auch Anhang B). Die Wurzel des Baums

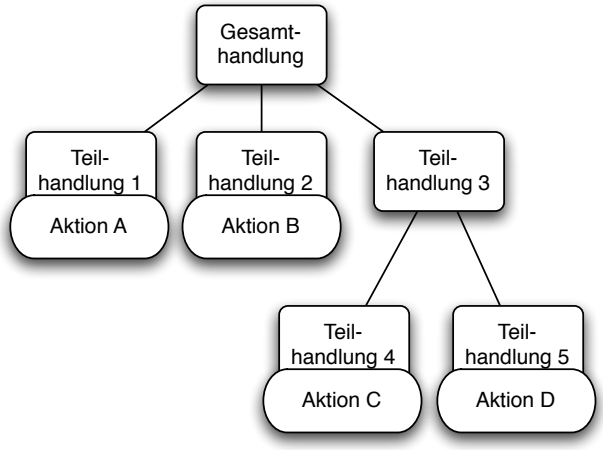


Abbildung 4.5: Schematische Darstellung eines hierarchischen Aufgabennetzwerkes

bildet die oberste Ebene der Handlung, und die Gesamthandlung ist entlang einer Tiefensuche von links nach rechts innerhalb des Baums definiert. Knoten beinhalten dabei entweder weitere Nachfolgeknoten (auch *Kinder* genannt), oder atomare Handlungen. Die Gesamthandlung setzt sich also einerseits aus den atomaren Handlungen und andererseits aus der Struktur des Baums zusammen.
Abbildung 4.5 zeigt eine Beispielhandlung, bestehend aus 6 Knoten und 4 Aktionen. Hier werden die Teilhandlungen der Gesamthandlung in der Reihenfolge ihrer Nummerierung interpretiert, woraus sich entsprechend die Aktionsfolge A – B – C – D ergibt.
Flexible Programme basieren auf dem Konzept *atomarer Handlungen*. Diese elementaren Aktionen werden in der Literatur auch Elementaroperatoren (engl.: *Motor Primitives*) genannt (siehe [Friedrich 99]) und stellen eine allgemeine Abstraktion für die Handlungsbeschreibung dar. In [Matarić 02] begründet M. Matarić die Verwendung von Elementaroperatoren über Analogien zur menschlichen Gehirnstruktur, wo das Vorhandensein sogenannter Spiegelneuronen ebenfalls die Existenz von Elementaroperatoren wahrscheinlich macht (siehe auch Kapitel 2.4.2).
Im Folgenden wird die Darstellung eines Knotens sowie des gesamten Baums diskutiert.

4.2.3 Formalisierung

Eine grundlegende Eigenschaft von Flexiblen Programmen ist, dass ein Programm (ein Baum) vollständig beschrieben ist durch die beinhalteten Knoten. Präziser ist eine Handlung beschrieben durch ihre Wurzel. Dies wird deutlich an der formalen Beschreibung eines Knotens. Umgekehrt beschreibt damit auch ein Knoten immer ein gültiges Flexibles Programm, da seine Beschreibung die vollständige Liste der Nachfolgeknoten, Vor- und Nachbedingungen etc. enthält.
Ein Knoten wird formal durch ein 8-Tupel beschrieben:

$$\mathcal{P} = \{Id, C_{pre}, C_{post}, C_{rt}, R, S, P, A\} \tag{4.1}$$

Dabei stellt Id einen eindeutigen Bezeichner dar, über den die jeweilige Handlung identifiziert wird. $C_{pre}, C_{post}, C_{rt}$ bezeichnen die Vorbedingungen, Nachbedingungen und Laufzeitbedingungen (Währendbedingungen), die für den jeweiligen Knoten gelten müssen. Diese werden

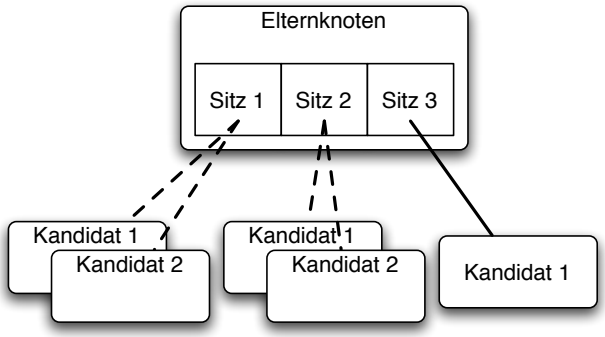


Abbildung 4.6: Beispiel mit 3 Sitzen in einem Knoten, wobei die ersten beiden Sitze jeweils 2 Kandidaten beinhalten. Der letzte Sitz hat nur einen Kandidaten. Potentielle Eltern-Kind-Beziehungen sind mit unterbrochener Linie gekennzeichnet.

in Kapitel 4.3 näher erläutert. R (engl.: *Rating*) stellt ein kontinuierliches Bewertungsmaß des Knotens dar, das zur Auswahl verwendet werden kann (siehe Kapitel 6.2). S (engl.: *Success*) stellt ein Bewertungsmaß für den Erfolg der Ausführung eines Knotens dar, das nach Ausführung berechnet wird. P (engl.: *Prospect*) als der Kindprospekt (im Folgenden Prospekt) enthält die Liste und Reihenfolge der möglichen Kindknoten[1], aus denen zur Laufzeit die passendsten ausgewählt werden. Die Aktion A beinhaltet die eigentliche auszuführende Aktion des Knotens.
Per Definition ist bei einem Knoten entweder der Prospekt oder die Aktion leer. Dies bedeutet, dass die Aktionen innerhalb eines Baums immer *Blätter* darstellen. Diese Konvention schränkt die Möglichkeiten der Repräsentation nicht ein und wurde aus Übersichtlichkeitsgründen festgelegt. Würde diese Forderung fallen gelassen, brächte speziell die Frage der Reihenfolge zwischen Aktion A eines Blattes und den Kindknoten schlechtere Handhabung und eine verminderte Lesbarkeit der Handlungen mit sich.

Der Prospekt

Ist ein Knoten kein Blatt und enthält somit keine Aktion, so ist sein Prospekt P nicht leer. Der Prospekt enthält alle möglichen Konfigurationen von Kindknoten.
Der Prospekt ist aufgebaut aus einer Menge von $n > 0$ Stellen, *Sitze* genannt, mit jeweils $m > 0$ *Kandidaten* pro Sitz. Die Sitze sind in $p \leq n$ parallelen Gruppen angeordnet. Analog zu den Bezeichnungen entspricht die Anzahl der Sitze der Anzahl der späteren Teilhandlungen, während für jeden Sitz mindestens ein Kandidat (eine Teilhandlung) existiert, der diesen einnehmen kann.
Die Sitze eines Knotens können als Teilhandlungen verstanden werden, wovon jeweils benachbarte Sitze als parallel ausführbar markiert werden können. Die Kandidaten eines Sitzes stellen verschiedene Alternativen für die Teilhandlung dar. Existiert also nur ein Kandidat für einen Sitz, ist die Teilhandlung eindeutig bestimmt. Existieren mehrere Kandidaten, muss zur Laufzeit diejenige Teilhandlung ausgewählt werden, die für die gegebene Situation am besten geeignet scheint. Dies geschieht über die Bedingungen und das Bewertungsmaß. Abbildung 4.6 zeigt ein Beispiel für eine solche Anordnung.

[1]Der Begriff *Prospekt* soll betonen, dass es sich um einen Container für die Menge der *potentiellen* Kindknoten (Teilhandlungen) handelt, aus denen die tatsächlichen Handlungen ausgewählt werden.

Die Verknüpfung von Teilhandlungen durch den Prospekt führt zu der Baumstruktur der Handlungen. Die resultierenden Bäume können dabei eine beliebige Tiefe besitzen, da auf jeder Ebene nur die direkten Kindknoten bekannt sind. Durch rekursive Verwendung des selben Knotens kann zudem ein Baum mit unendlicher Tiefe konstruiert werden.

Parallele Ausführung mehrerer Handlungsstränge

Werden mehrere Sitze in der selben parallelen Gruppe eingeordnet, so werden die entsprechenden Handlungsteile parallel ausgeführt. Auf diese Weise können also Handlungsteile, die konfliktfrei gleichzeitig ausgeführt werden können, parallel angeordnet werden.
Die parallele Gruppe von Sitzen wird für die Fertigstellung dabei wie ein einzelner Sitz behandelt. Die Gruppe wird dann als abgearbeitet gekennzeichnet, wenn alle Sitze innerhalb der Gruppe abgearbeitet sind.
Die Abläufe zur Kandidatenauswahl und Abarbeitung eines Handlungsbaums werden in Kapitel 6.2 detaillierter dargestellt.

Abhängigkeiten Flexibler Programme

Es enthält also jeder Knoten alle Information über Vor- und Nachbedingungen, Aktionen und Kindknoten. Umgekehrt enthält kein Knoten Informationen über etwaige Elternknoten. Dies ist wichtig für die Flexiblität der vorgestellten Aufgabenmodellierung. Erst indem jeder Knoten unabhängig existiert, kann er auch in jedem Flexiblen Programm als Teilprogramm verwendet werden.
Bei der Synthese von Handlungswissen können also existierende Programme als Teilprogramme zusammengesetzt werden. Abhängigkeiten zwischen diesen Teilprogrammen entstehen zum einen durch die Art der Verknüpfung: Durch Sequentialisierung und Parallelisierung werden Teile explizit kausal verknüpft. Zum anderen entstehen Abhängigkeiten mittelbar über die Erzeugung und Verwendung von Merkmalen. Ein Beispiel ist hier die Verknüpfung einer sensoriellen Aktion (z.B. *Tisch lokalisieren*) und einer ausführenden Aktion (z.B. *fahre zu Tisch*). Diese Aktionen sind voneinander abhängig über das erzeugte Merkmal der Tischposition.
Dieses Merkmal wird nicht unmittelbar zwischen den einzelnen Knoten ausgetauscht; dies würde die Bedingung der Flexibilität und Unabhängigkeit einzelner Teilprogramme verletzen. Zwischen verschiedenen Flexiblen Programmen werden Daten über einen lokalen Datenspeicher ausgetauscht, der in Kapitel 4.3.1 näher erläutert wird.
Die Modellierung von Handlungswissen wird in [Knoop 06a, Schmidt-Rohr 05] weiter detailliert.

Das Handlungswissen in Form eines Baums wird also rekursiv aufgebaut, indem jeder Knoten eine eindeutige Teilhandlung beschreibt und dafür die wiederum zugehörigen Teilhandlungen definiert. Ein Flexibles Programm beschreibt also alle möglichen Handlungsstränge, die das System bei der Ausführung verfolgen kann. Dabei beinhaltet jeder Knoten zusätzlich zu Aktion und Kindknoten seine Vorbedingungen und Nachbedingungen zur Überprüfung der Ausführbarkeit und des Erfolgs. Die Modellierung von Bedingungen und das dafür notwendige Relationenmodell werden im folgenden Kapitel vorgestellt.

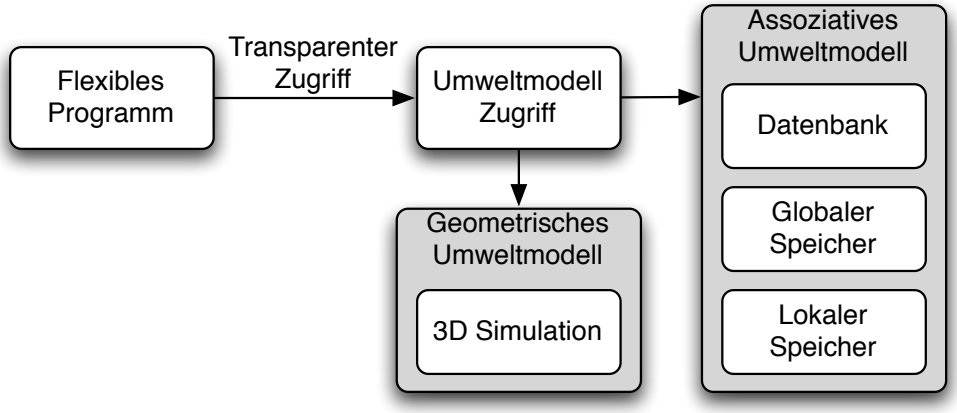


Abbildung 4.7: Hierarchischer Aufbau des Umweltmodells mit assoziativem und geometrischem Modell

4.3 Modellierung von Umwelt und Bedingungen

Üblicherweise beinhalten Roboteraktionen eine physische Interaktion mit der Umwelt. Diese beschränkt sich bei reiner Navigation auf das Durchqueren verschiedener Bereiche, bei der Manipulation werden Objekte (oder ihr Zustand) in der Welt verändert. Diese physische Interaktion hat in den meisten Fällen einige Voraussetzungen: Zu manipulierende Objekte müssen existieren, erreichbar sein, eine gewisse Anordnung muss gegeben sein oder Objektzustände einer Bedingung entsprechen.
Diese Vorbedingungen müssen vor einer entsprechenden Aktion des Roboters überprüft werden. Zusätzlich können nach der Aktion die gewünschten Effekte mit dem Umweltzustand verglichen und so der Erfolg der Aktion geprüft werden. In einigen Fällen ist es sogar notwendig, dass gewisse Bedingungen während der Ausführung der Aktion ständig erfüllt sind. Um eine solche Menge an Bedingungen mit dem aktuellen Umweltzustand vergleichen zu können, muss zum einen der Umweltzustand modelliert und soweit bekannt sein, und zum anderen müssen die gegebenen Bedingungen modelliert und daran überprüft werden können.

4.3.1 Zustandsmodellierung der Umwelt und des Roboters

Der Zustand der Umwelt und des Roboters setzt sich zusammen aus zwei Modelltypen. Einerseits wird der Weltzustand beschrieben durch Schlüssel-Wert Paare, die Eigenschaften und den entsprechenden Wert darstellen. Dies ist z.B. die Position des Roboters im Raum, die durch drei Koordinaten dargestellt und unter einem vereinbarten Schlüssel abgelegt wird. Andererseits wird die Umwelt durch ein geometrisches Modell beschrieben, das in einer Simulationsumgebung (KAVIS) abgelegt wird. Diese Simulationsumgebung ist auch in der Lage, definierte Relationen zwischen verschiedenen Objekten in der Simulation zu überprüfen [Schaude 96].
Das assoziative Umweltmodell besteht aus einer Hierarchie von Datenspeichern (siehe Abbildung 4.7). Auf der unteren Ebene befindet sich ein lokaler Datenspeicher, darüber ein globaler Speicher. Ganz oben befindet sich der Zugriff auf eine globale Datenbank, in der Relationen und Eigenschaften abgelegt sind, die dauerhaft oder langfristig gelten (siehe [Becher 06]).

Flexible Programme und Agenten, die Informationen über die Umwelt entweder verwenden oder zur Verfügung stellen wollen, können über eine gemeinsame Schnittstelle darauf zugreifen. Hier wird auch die Konsistenz zwischen den verschiedenen Hierarchieebenen des Modells sichergestellt. Eigenschaften von Objekten und Relationen zwischen diesen können entweder über das assoziative oder über das geometrische Modell überprüft werden.

4.3.2 Modellierung von Bedingungen

Um eine Vorbedingung überprüfen und als WAHR oder FALSCH klassifizieren zu können, müssen diese Bedingungen zuerst explizit modelliert werden.
Prinzipiell lässt sich jede Art von Bedingung sowie Verknüpfungen davon als prädikatenlogische Formel darstellen. Hierfür lässt sich die Prädikatenlogik erster Ordnung (PL1) heranziehen. Diese definiert sich wie folgt:

Definition 4.1 Die Menge aller Formeln der PL1 $\mathbb{F}$ ist die kleinste Menge mit:

- $\{\text{WAHR}, \text{FALSCH}\} \cup \mathbb{A} \subseteq \mathbb{F}$
- Wenn $G, H \in \mathbb{F}$ und $x \in \mathbb{V}$ dann:
  $(\neg G), (G \wedge H), (G \vee H), (G \rightarrow H), (G \leftrightarrow H), (\forall x G), (\exists x G) \in \mathbb{F}$

Dabei ist $\mathbb{A}$ die Menge der atomaren Formeln, die Prädikate, Konstanten und Funktionen wie z.B. $f(x, y)$ umfasst. $\mathbb{V}$ ist die Menge der Variablen.
Für die Darstellung von Bedingungen für die Ausführung von Roboterprogrammen stellt sich nun die Frage, welche Teile der PL1 für die Repräsentation abzubilden sind.
In einer realen Umwelt sind quantifizierte Ausdrücke besonders wichtig. Dies lässt sich allein schon daraus erkennen, dass bei einem Roboterprogramm die beteiligten Objekte nicht explizit angegeben werden können, sondern über Variablen spezifiziert und in der Umwelt gefunden und im Programm ersetzt werden müssen. Hierfür wird offensichtlich der Existenzquantor benötigt. Die logischen Quantoren müssen also integriert werden.
Ein ebenso wichtiges Instrument ist die Negation. Im Gegensatz zum Ansatz bei STRIPS kann im realen Einsatz nicht die gesamte Umwelt bekannt sein; alle nicht erwähnten Relationen als FALSCH zu definieren ist also nicht sinnvoll. Es muss demnach möglich sein, sowohl das Erfüllt-Sein als auch das Nicht-Erfüllt-Sein einer Bedingung zu fordern.
Mit Konjunktion und Negation lassen sich alle möglichen logischen Terme bereits darstellen. Dennoch ist es sinnvoll, die Disjunktion zur Verfügung zu stellen, um Formulierungen zu vereinfachen. Alle weiteren logischen Operatoren werden einerseits selten verwendet und können andererseits leicht nachgebildet werden. Auf den Einsatz von $\rightarrow$ und $\leftrightarrow$ wird deshalb verzichtet. Eine vollständige Aufzählung der Operatoren aus PL1 und deren Verwendung für die Modellierung von Bedingungen ist in Tabelle 4.2 gegeben.
Zusätzlich zu den Standardoperatoren werden die für den realen Einsatz wichtigen Operatoren $=, >$ und $<$ zur Verfügung gestellt. Dies erlaubt Ausdrücke wie $\exists x \in \mathbb{M} : x < 3$.
Die zur Verfügung gestellten weiteren Prädikate richten sich nach den Relationen, die mit dem gegebenen Umweltmodell überprüft werden können [Bertram 05]. Das geometrische Modell, das von der Umgebung KAVIS zur Verfügung gestellt wird, bietet die Überprüfung einer Reihe von Relationen an. Das assoziative Umweltmodell bietet selbst nur das Auslesen von Attribut-Wert Paaren an. Hier können also die Operatoren $=, >, <$ angewendet werden, z.B.

Logische Komponente	STRIPS	Flexible Programme
Prädikate	√	√
=	×	√
Funktionen	×	×
¬	×	√
∧	√	√
∨	×	√
→	×	×
↔	×	×
∀	×	√
∃	×	√

Tabelle 4.2: Unterstützte Aspekte der Logik zur Modellierung von Bedingungen in STRIPS und Flexiblen Programmen

zur Überprüfung von erwarteten Werten. Hierfür kann auf Attribute der im Modell repräsentierten Objekte einfach zugegriffen werden: Z.B. wird die Farbe eines bestimmten Objekts über *Objektbezeichner/Farbe* adressiert. Wie erwähnt können für ein Roboterprogramm entsprechende Ort- und Objektbezeichner nicht statisch vergeben werden, um die Flexibilität nicht einzuschränken. Aus diesem Grund können bei der Formulierung von Bedingungen auch Variablen statt Bezeichner verwendet werden.
Die logischen Terme zur Darstellung und Verknüpfung von Bedingungen können in Form eines Syntaxbaums dargestellt werden. So kann die Bedingung

```
(or (and (Auf(Tasse,Kiste1))
         (Tasse/Farbe = rot))
         (Tasse/Füllkapazität > 20)
    (and (Auf(Tasse,Kiste2))
         (not (Tasse/Füllkapazität = 0))))
```

wie in Abbildung 4.8 dargestellt werden.
Die Überprüfung einer solchen Bedingung ist in den meisten Fällen recht einfach. Zwei Eigenschaften sollen im Folgenden allerdings genauer beleuchtet werden.
In vielen Fällen müssen nicht alle Terme betrachtet werden, um das Gesamtergebnis bestimmen zu können. Dies ist z.B. der Fall, wenn bei einer Disjunktion ein Operand das Ergebnis WAHR oder bei einer Konjunktion ein Operand das Ergebnis FALSCH hat. Dadurch kann die Überprüfung in vielen Fällen beschleunigt werden.
Wie erwähnt, kann im realen Einsatz nicht der gesamte Weltzustand bekannt sein. Dies bedeutet, dass bei der Überprüfung von Prädikaten die möglichen Ergebnisse WAHR, FALSCH

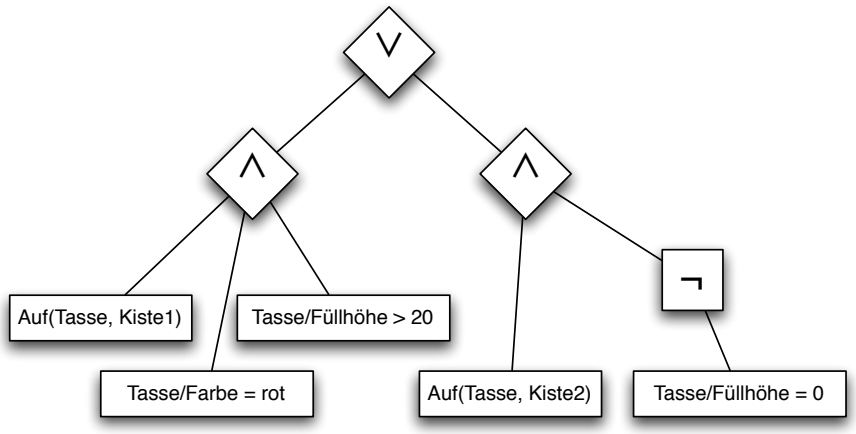


Abbildung 4.8: Graphische Darstellung einer verknüpften Bedingung

um *unbekannt* erweitert werden müssen. Wird nun jedes unbekannte Ergebnis als FALSCH oder WAHR angenommen, wird das unvollständige Wissen über die Umwelt nicht in das Ergebnis propagiert. Beide Fälle ergeben suboptimale Lösungen.
Kodiert man WAHR = 1 und FALSCH = 0, so kann *unbekannt*= 0.5 definiert werden. Propagiert man nun diese Kodierungen durch die Auswertung der Bedingung, ergeben sich zwei Vorteile gegenüber einer reinen WAHR/FALSCH Logik:

- Teilweise unbekannte Terme können, kombiniert mit eindeutigen Termen, einen insgesamt eindeutigen Wahrheitswert ergeben:

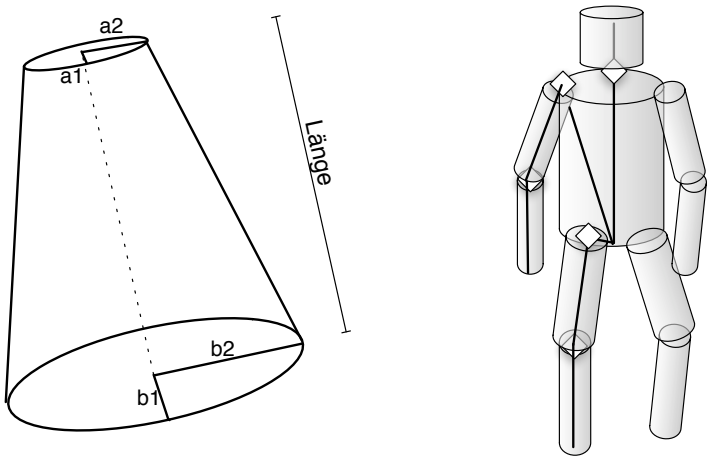
  FALSCH $\wedge$ *unbekannt* = FALSCH
  WAHR $\vee$ *unbekannt* = WAHR

- In einer dem Roboter teilweise unbekannten Umwelt ist das Wissen über die Falschheit einer Aussage nicht gleichbedeutend mit dem Nichtwissen über deren Wahrheitswert. Das bedeutet, dass bei einer als *unbekannt* klassifizierten Bedingung der Roboter trotzdem versuchen kann, die entsprechende Aktion auszuführen. Die Bedingung könnte sich im Laufe der Ausführung als WAHR herausstellen.

Ist dieses Verhalten nicht erwünscht, kann durch die Abbildung von *unbekannt* auf FALSCH die binäre Entscheidung wiederhergestellt werden.

4.4 Körpermodell des Menschen

Für die Beobachtung und Interpretation des Menschen und seiner Bewegungen werden kinematisch-geometrische Modelle benötigt, mit deren Hilfe die Bewegungen verfolgt werden können. Da mit einem *Time-of-Flight*-Sensor (Swissranger^{tm}) eine Tiefenbildkamera als 3D-Sensor zur Verfügung steht, und die Bewegungen des Menschen dreidimensional stattfinden und interpretiert werden müssen, findet ein 3D Modell des menschlichen Körpers Anwendung.
Im Folgenden wird dieses Modell sowie die Modelle für verschiedene Gelenktypen erläutert.

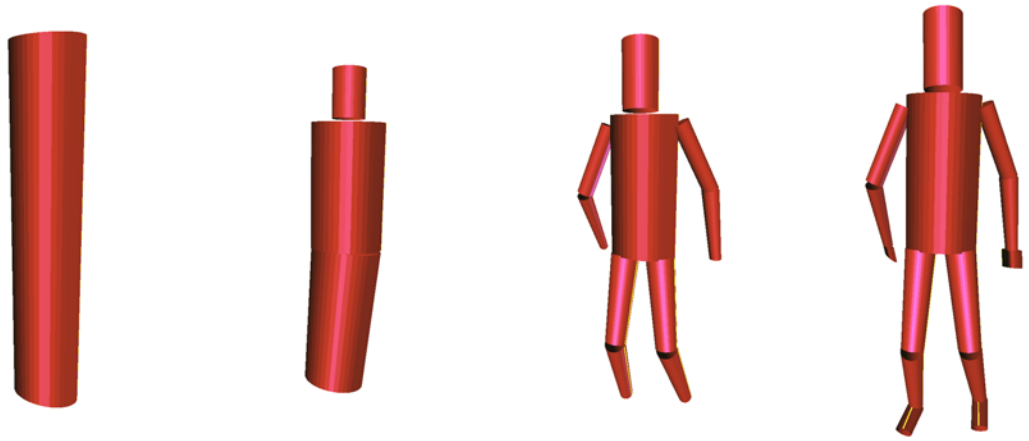


(a) (b)

Abbildung 4.9: Verallgemeinerter Zylinder (a), vollständiges Körpermodell aus 10 verallgemeinerten Zylindern (b)

4.4.1 Geometrische Modellierung

Um die Bewegungen und Handlungen eines Menschen im Sichtbereich eines Roboters verfolgen und interpretieren zu können, ist die Nutzung von geometrischem und kinematischem Modellwissen sinnvoll. Aus Rechenzeitgründen sowie unter dem Aspekt der begrenzten sensorischen Beobachtbarkeit ist es sinnvoll, Geometrie und Biokinematik zu abstrahieren sowie die signifikanten Merkmale zu modellieren, um die Bewegungen zu diskriminieren, die mit den gegebenen Sensoren sichtbar und unterscheidbar sind.
Das hierzu vorgeschlagene geometrische Modell besteht dabei aus parametrierten Grundkörpern. Dieser Grundkörper kann als *verallgemeinerter Zylinder* bezeichnet werden. Er entsteht aus einem Zylinder, dessen Boden und Deckelfläche durch eine Ellipse ersetzt wurden, wobei die Halbachsen der beiden Ellipsen parallel sind.
Ein solcher Körper besitzt also 5 Parameter: Die Längen der Halbachsen beider Ellipsen und die Länge des Zylinders. Abbildung 4.9(a) zeigt einen solchen Körper.
Das vollständige Körpermodell wird nun aus parametrierten vereinfachten Zylindern zusammengesetzt. Dabei werden zwischen den einzelnen Teilen Gelenke modelliert (siehe Kapitel 4.4.2). Für Torso und Kopf wird jeweils ein Zylinder verwendet, und für Arme und Beine jeweils zwei. Ein solches Modell ist schematisch in Abbildung 4.9(b) abgebildet.
Auf dieselbe Weise kann das Modell verfeinert werden und beispielsweise mit Händen und Füßen versehen werden. Im vorliegenden Fall ist dies allerdings aufgrund der Sensorauflösung nicht sinnvoll, da die Granularität des Modells mit der Messgenauigkeit der Sensoren übereinstimmen sollte. Abbildung 4.10 zeigt das Menschmodell in verschiedenen Granularitäten, die für verschieden komplexe Anforderungen und Sensorkonzepte verwendet werden können. Der Zusammenhang zwischen vorhandener Information aus der Sensorik und der verwendeten Komplexität im Modell ist in Abbildung 4.11 qualitativ skizziert. Je höher also die Anzahl der Freiheitsgrade eines Modells ist, umso mehr Information wird für die korrekte Anpassung aus der Sensorik benötigt. Fehlende Information kann dabei nur teilweise durch weiteres Modellwissen kompensiert werden.
Die vorgestellte Modellierung des Menschen ist in [Knoop 05a] und [Knoop 06b] zu finden.



(a) (b) (c) (d)

Abbildung 4.10: Körpermodell des Menschen aus verallgemeinerten Zylindern in verschiedenen Granularitäten. Der Körper kann mit einem (a) oder drei (b) Grundkörpern approximiert werden, oder genauer mit 10 (c) bzw. 14 (d) verallgemeinerten Zylindern.

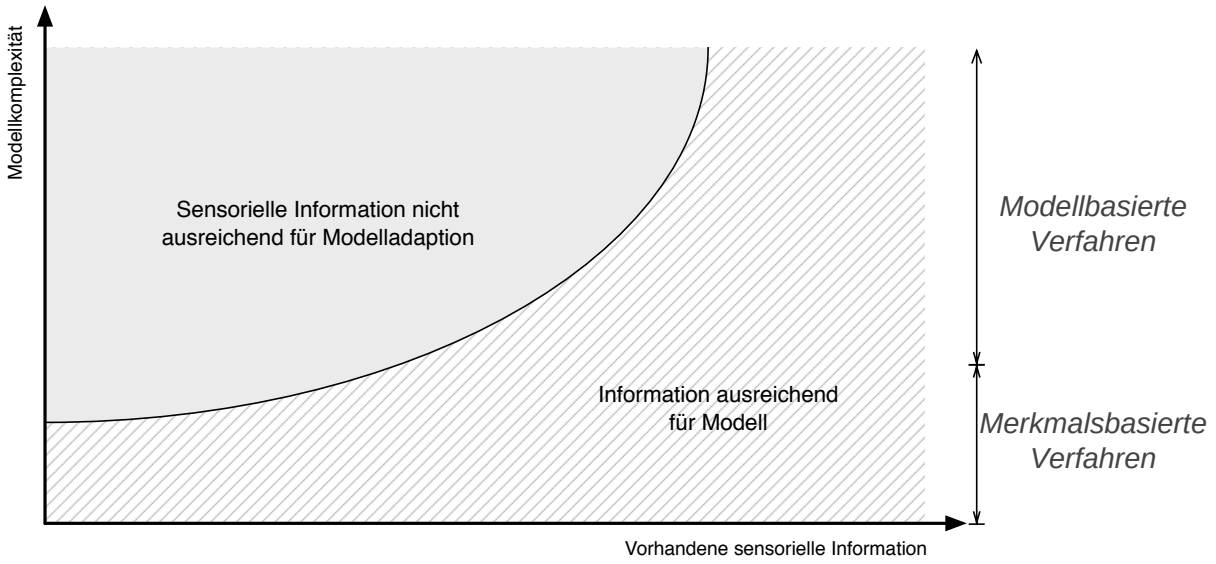


Abbildung 4.11: Qualitativer Zusammenhang zwischen Sensoraufwand und Modellkomplexität. Je mehr Freiheitsgrade das Modell besitzt (in topologischer, geometrischer, dynamischer Sicht), umso mehr Sensorinformation wird für die Verfolgung benötigt.

4.4.2 Gelenkmodell

Der menschliche Körper besteht aus einzelnen Körperteilen, die über Gelenke verbunden sind. Dabei sind die Körperteile in sich starr aufgrund des Knochengerüsts; dies wird durch die Modellierung als starre verallgemeinerte Zylinder nachgebildet. Die Gelenke zwischen den einzelnen Teilen haben beim Menschen unterschiedliche Eigenschaften; als Beispiel sei der Ellenbogen genannt, dessen Aufbau sich stark von z.B. der Schulter unterscheidet. Dabei variieren die Gelenke sowohl in der Anzahl der Freiheitsgrade (Ellenbogen: 1 Freiheitsgrad, Schulter: 3 Freiheitsgrade) als auch in der Flexibilität und dem möglichen Winkelbereich.
Im vorliegenden Körpermodell werden die Gelenke explizit mit diesen Eigenschaften modelliert, um die möglichen Konfigurationen in Modell und Wirklichkeit ähnlich zu gestalten.
Ein Gelenkmodell besteht dafür aus einer Menge von *elastischen Bändern*, die die Modelle durch ein Gelenk verketteter Körperteile verbinden. Diese elastischen Bänder stellen also Relationen zwischen verschiedenen Körperteilen auf, indem sie Punkte auf dem einen mit Punkten auf dem anderen assoziieren. Werden diese Korrespondenzen als Gummibänder mit Elastizitätskräften proportional zur Ausdehnung interpretiert, so erhält man eine resultierende Menge an Kräften auf die einzelnen Modellteile, die diese an den Gelenken zusammenhält oder entsprechend zueinander gehörende Punkte annähert. Abbildung 4.12 zeigt zwei Zylinder, die durch vier elastische Bänder verbunden sind.

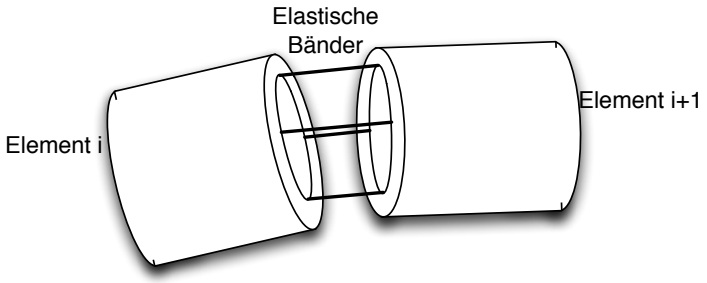


Abbildung 4.12: Elastische Bänder als Gelenkmodell. Die resultierenden Kräfte treiben die verbundenen Teile aufeinander zu

Mit dem elastischen Band als kleinster Einheit zur Gelenkmodellierung ergibt sich ein weiterer Vorteil dieser Modellierung, wenn das Gelenk mit $n > 1$ Bändern modelliert wird. Wird ein Gelenk mit mehreren Bändern modelliert, die geometrisch sinnvoll angeordnet sind, so lassen sich verschiedene Gelenkcharakteristiken erzeugen.
Abbildung 4.13 zeigt drei verschiedene Gelenkmodelle: Kugelgelenk, Knickgelenk und ein in allen Achsen eingeschränktes Gelenk. Diese werden im Folgenden vorgestellt.

Kugelgelenke haben 3 rotatorische Freiheitsgrade. Z.B. entspricht das Schultergelenk einem solchen Modell. Der Oberarm kann hier in allen drei Achsen bewegt werden: Rotation nach oben/unten, Rotation nach vorne/hinten, sowie die Drehung um die eigene Längsachse. Kugelgelenke werden mit einem einzelnen elastischen Band modelliert, das die zugehörigen Punkte verbindet. Durch diese Korrespondenz werden zwei Punkte auf den benachbarten Elementen einander zugeordnet, wodurch sich ein Kugelgelenk ergibt.

Knickgelenke besitzen einen rotatorischen Freiheitsgrad. Diese können in Ellbogen und Knie gefunden werden, wo die restlichen Freiheitsgrade vollständig fixiert sind. Knickgelenke werden mit 2 elastischen Bändern modelliert, die jeweils auf der Knickachse auf beiden Teilen fixiert sind. Durch die Anordnung zweier Kugelgelenke auf der Knickachse wird so ein Knickgelenk modelliert.

Eingeschränkte Gelenke werden modelliert, indem die elastischen Bänder an den Endpunkten der Halbachse einer Ellipse fixiert werden (deshalb auch *elliptisches Gelenk*). Das Hüftgelenk entspricht einem solchen Gelenkmodell. Es könnte auch als Knickgelenk modelliert werden, besitzt allerdings eine kleine Beweglichkeit gegen die Knickachse, die nicht zu vernachlässigen ist. Hier zeigt sich ein Vorteil dieser Modellierung: Durch geeignete Wahl der Parameter kann eine teilweise Einschränkung dieser Achsen erreicht werden. Eingeschränkte Gelenke sind in allen Achsen beweglich, die Beweglichkeit hängt allerdings stark von den gewählten Parametern ab.

Kugel- und Knickgelenk sind dabei Spezialfälle des eingeschränkten Gelenks. Für das Knickgelenk wird eine der Halbachsen der Ellipse, auf der die elastischen Bänder angeordnet sind, mit Länge $l = 0$ gewählt. Beim Kugelgelenk werden beide Halbachsen in der gleichen Weise gewählt, wodurch alle elastischen Bänder in einem Punkt zusammen fallen.

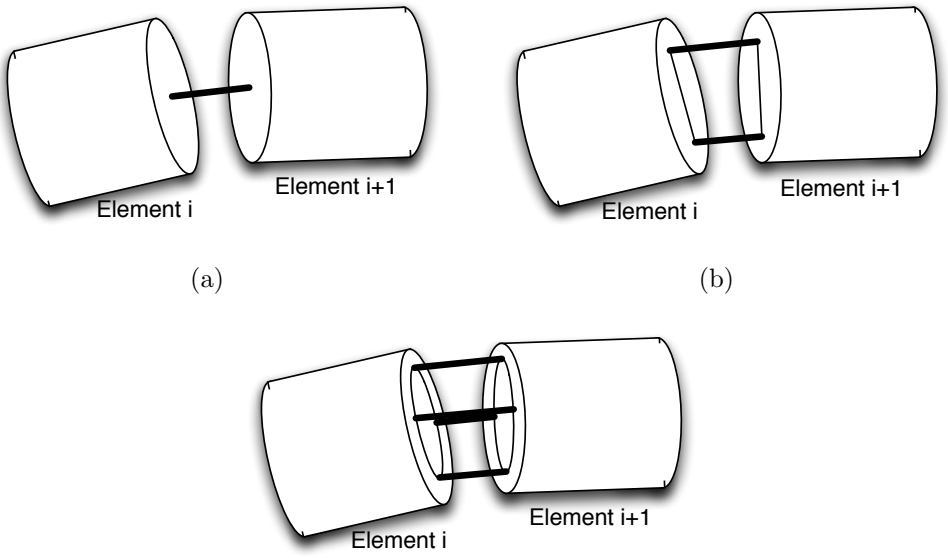


Abbildung 4.13: Verschiedene Gelenktypen, die mit elastischen Bändern modelliert werden können: Kugelgelenk (a), Knickgelenk (b) und in allen Achsen eingeschränktes Gelenk (c)

Wie man an der Modellierung des Hüftgelenk sieht, ergibt die vorgestellte Gelenkmodellierung eine "weiche" Einschränkung der Freiheitsgrade. Dies ist besonders vorteilhaft, weil die verwendeten Modelle approximativ sind und ein Modell so für verschiedene Personen verwendet werden kann, ohne dass einzelne Parameter ständig angepasst und individuell eingestellt werden müssen.

Hals	In allen Achsen schwach eingeschränktes Gelenk (3 eingeschränkte Freiheitsgrade)
Schultern	Kugelgelenk (3 Freiheitsgrade)
Ellenbogen	Knickgelenk (1 Freiheitsgrad)
Hüftgelenke	In allen Achsen eingeschränktes Gelenk: Sagittalebene schwach, Frontalebene stark eingeschränkt (3 eingeschränkte Freiheitsgrade)
Knie	Knickgelenk (1 Freiheitsgrad)

Tabelle 4.3: Verwendete Gelenktypen zur Modellierung des menschlichen Körpers

Mit dem Modell für Körperteile (siehe Kapitel 4.4.1) und dem vorgestellten Gelenkmodell wird nun das Menschmodell hierarchisch aufgebaut (siehe Abbildung 4.9). Dabei werden die in Tabelle 4.3 angegebenen Gelenktypen verwendet.

4.5 Modellierung menschlicher Aktivitäten

Während der Ausführung von Roboterprogrammen soll der Roboter Gesten und andere Aktionen des Menschen erkennen und klassifizieren können. Zeigegesten sowie andere der Interaktion dienende Bewegungen werden hier als *Aktivitäten* bezeichnet. Eine Aktivität ist also eine *charakteristische Bewegung mit Bedeutung.*
Eine konzeptuelle Modellierung für Aktivitäten im Kontext der Mensch-Maschine Interaktion stellt die *Aktivitätstheorie*[Kaptelinin 95] dar (engl.: *Activity theory*). Die Aktivitätstheorie bietet einen Rahmen, der Struktur, Entwicklung und Kontext von Computer-unterstützten Aktivitäten modelliert und einschließt. Fünf Prinzipien liegen der Aktivitätstheorie zugrunde (siehe [Kaptelinin 95]):

- Aktivitäten sind in ihrer Repräsentation hierarchisch gegliedert und organisiert. Sie bestehen aus zielgerichteten Aktionen, und die kleinste Einheit wird Operation genannt. Sie entspricht einer Elementaroperation (siehe Kapitel 3.1.1). Aktivitäten werden immer bewusst ausgeführt.

- Aktivitäten sind auf ein Objekt gerichtet. Aktivitäten dienen der Interaktion mit der Umwelt, und werden konsequenterweise mit einem Objekt als Ziel ausgeführt. Dieses Objekt ist nicht zwingend ein physikalisches Objekt; es kann auch ein abstrakteres Ziel oder ein Zustand sein.

- Es wird zwischen internen und externen Aktivitäten unterschieden. Interne Aktivitäten entsprechen Planungsoperationen, während externe Aktivitäten tatsächliche Interaktion mit der Umwelt beinhalten.

- Die Aktivitätstheorie unterscheidet zwischen einer Aktivität und dem Werkzeug, das zur Ausführung verwendet wird. Aktivitäten werden also mit Hilfe von Werkzeugen vermittelt oder umgesetzt.

- Die Entwicklung und Veränderung der Interaktionsformen, Werkzeuge und Beteiligten ist intensives Ziel der Forschung und gleichzeitig ein Grundprinzip der Vorgehensweise. Diese Faktoren werden aktiv im Modell berücksichtigt.

Die Erkennung von Aktivitäten zur Kommandierung und Parametrierung stellt hier eine Untermenge dar, die durch die Anwendung und die sensorischen Möglichkeiten eingeschränkt ist. Einerseits muss nur eine begrenzte Menge an Aktivitäten modelliert und erkannt werden; andererseits ist auch die Genauigkeit und Auflösung der Robotersensorik beschränkt, so dass nicht alle Aktivitäten beobachtbar sind. Grundsätzlich können nur externe Aktivitäten beobachtet werden, da nur diese physikalisch ausgeführt werden.
Aus der Aktivitätstheorie lassen sich einige Annahmen für die Erkennung und Klassifikation von Aktivitäten für die Interaktion ableiten: Aktivitäten sind zusammengesetzt aus einer endlichen Menge an Elementaroperationen. Sie sind immer gerichtet, haben also ein Ziel. Aktivitäten sind für die Interaktion semantisch fassbar, und die beobachtete Manifestation entspricht nur dem Werkzeug, nicht der Aktivität selbst; eine Aktivität kann also auf verschiedene Arten (mit unterschiedlichen Werkzeugen) ausgeführt werden.
Entsprechend wird das Aktivitätsmodell für die Klassifikation gewählt. Eine endliche Menge an Elementaroperationen wird während der Bewegung des Menschen beobachtet. Anhand dieser Operationen werden Aktivitäten definiert. Sie können also mit geeigneten Klassifikatoren aufgrund von Bewegungen erkannt werden. Über den Abstraktionsgrad von Aktivitäten wird keine Einschränkung gemacht; zur Testmenge der Aktivitäten gehören beispielsweise Zeigegesten, Winken, Verbeugen, Gehen, Stehen und Sitzen (siehe Klassifikation in [Vacek 05]).
Die Menge der elementaren Operationen wird dabei direkt von der Bewegung des menschlichen Körpers abgeleitet. Diese Operationen oder Merkmale werden aus der Bewegung extrahiert, die mit Hilfe eines geometrischen Modells für den menschlichen Körper aufgezeichnet wurde.

4.6 Zusammenfassung

Die verwendete Architektur zur Ausführung von Handlungswissen ist eine verteilte, hybride, ereignisbasierte Architektur. In dieser Architektur ist das Handlungswissen in Form von Flexiblen Programmen repräsentiert. Flexible Programme sind hierarchisch geordnete, modulare Roboterprogramme, die sowohl bedingte Verzweigungen als auch parallele Teilzweige zulassen und modellieren können. Bedingungen können hierfür geschlossen dargestellt werden.
Zur Beobachtung und Interpretation der Bewegungen eines Menschen durch den Roboter werden menschliche Aktivitäten anhand der Aktivitätstheorie modelliert. Diese definiert Aktivitäten durch ein Ziel(objekt) und eine Intention, die mit Hilfe von Operationen umgesetzt werden. Zur Beobachtung dieser Operationen wird der menschliche Körper mit Hilfe von verallgemeinerten Zylindern geometrisch und topologisch modelliert. Dieses Modell wird im Weiteren verwendet, um tatsächliche Operationen zu verfolgen, zu beobachten und zu interpretieren.

Kapitel 5

Abbildung von Handlungswissen auf Flexible Roboterprogramme

Handlungswissen, das mit Hilfe des Programmierens durch Vormachen akquiriert wurde, soll mit geeigneten Verfahren so umgeformt werden, dass ein Roboter dieses ausführen kann. Ein Makrooperator soll also auf ein Flexibles Programm derart abgebildet werden, dass dieses von einem Zielroboter ausgeführt werden kann.
Dieses Kapitel geht auf den vorgestellten Abbildungsprozess ein. Dazu werden zuerst die Eigenschaften von Makrooperatoren und Flexiblen Programmen gegenübergestellt. Daraus werden dann die notwendigen Schritte zur Abbildung abgeleitet und detailliert dargestellt.

5.1 Vergleich von Makrooperatoren und Flexiblen Programmen

Sowohl Makrooperatoren als auch Roboterprogramme basieren auf *Primitiven*, aus denen Aktionen aufgebaut sind. Betrachtet man beide Darstellungen, ergeben sich Unterschiede bei der Struktur und dem Aufbau der Aktionen, in der Menge der betrachteten Primitive und auch in den einzelnen Primitiven.
Das Ziel eines PdV-Systems ist es, aus einer Vorführung Aktionen und Bedingungen zu segmentieren und extrahieren, die zu dem Gesamtziel der beobachteten Handlung beitragen. Dieses wird aus den beobachteten Weltzuständen vor und nach der Vorführung extrahiert. Ein Makrooperator enthält also nur Aktionen, die für das Ziel während der Vorführung durch den Menschen relevant sind. Die Ausführungsbeschreibung für den Roboter enthält dagegen eine wesentliche Menge zusätzlicher Aktionen, die notwendig sind, um die Teilziele und das Gesamtziel erfüllen zu können. Dies sind beispielsweise zusätzliche Bewegungskommandos, um Manipulationen ausführen zu können.
Der Lernprozess zur Erzeugung von Makrooperatoren basiert grundsätzlich nur auf erfolgreichen Vorführungen. Vorführungen, die nicht zu dem gewünschten Ergebnis führen, werden vom Benutzer aus der Trainingsmenge entfernt. Das gelernte Handlungswissen enthält also weder Wissen über mögliche Fehlerfälle, noch über die Behandlung derselben. Im Gegensatz dazu soll die Ausführung von Aufgaben mit einem Serviceroboter fehlertolerant und robust sein. Mögliche Fehler und Ausnahmen sollen also erkennbar und behandelbar sein. Das

Wissen darüber muss deswegen als Hintergrundwissen während der Abbildung hinzugefügt werden.
Die Ziele und Bedingungen werden während der Vorführung aufgrund des Umweltmodells und seiner Veränderungen extrahiert. Ein PdV-System basiert auf der Annahme der Weltabgeschlossenheit (engl.: *Closed-World Assumption*), es wird also die Vorraussetzung getroffen, dass alle relevanten Elemente der Umwelt modelliert und beobachtbar sind. Während der Vorführung wird direkt auf eine geometrische, semantische oder topologische Beschreibung der Umwelt zugegriffen, es sind keine zusätzlichen sensorischen Aktionen notwendig. Diese werden nicht beobachtet. Dies ist ersichtlich am Beispiel der Lokalisation eines Objekts durch den Demonstrator, die nicht erfasst werden kann. Ein Makrooperator enthält diese Aktionen folglich nicht. Während der Ausführung durch den Roboter muss die Umwelterfassung jedoch durchgeführt werden. Bei der Abbildung müssen also Aktionen eingeführt werden, um relevante Zustände der Umwelt erfassen zu können.
Da die Vorführung von einem Menschen durchgeführt wird, sind alle Bewegungen und Aktionen notwendigerweise im Aktions- und Arbeitsraum des Menschen gegeben. Die dem Menschen mögliche Menge an Aktionen entspricht aber nicht unbedingt der des Roboters. Zusätzlich sind der Arbeits- und Konfigurationsraum bei Mensch und Roboter unterschiedlich. Beides muss bei der Abbildung von Makrooperator auf Roboterprogramm berücksichtigt werden.

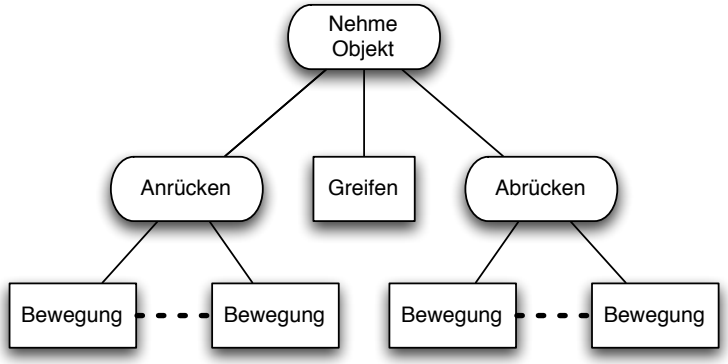


Abbildung 5.1: Schematisches Beispiel eines durch PdV akquirierten Makrooperators. Dargestellt ist ein Teil einer Pick&Place Operation. Aus der Vorführung wurde die Bewegungstrajektorie zum Anrücken, der Greifvorgang sowie die Abrückbewegung extrahiert. Die Position des Objekts, die Verwendung von Arm und Hand sowie die erfolgreiche Ausführung werden als gegeben vorausgesetzt.

Zur Verdeutlichung zeigt Abbildung 5.1 schematisch den Aufbau eines Makrooperators, der aus einer Vorführung extrahiert wurde. In diesem Fall wurde vom Demonstrator ein Objekt gegriffen. Die Erkennung segmentiert dabei das Anrücken, den Greifvorgang selbst und das Abrücken. Abbildung 5.2 zeigt dagegen, wie ein entsprechendes Flexibles Programm aussehen könnte. Während der Ausführung mit einem Roboter sind zusätzliche sensorische Aktionen notwendig, sowie in der Vorführung nicht enthaltene weitere Aktionen. Auch die interne Verwaltung (in diesem Fall Ressourcenverwaltung) muss im Roboterprogramm berücksichtigt werden.
Aus diesem Vergleich sowie den aufgezeigten Beispielen lässt sich ein wichtiger Unterschied zwischen Makrooperatoren und Flexiblen Programmen erkennen: Makrooperatoren, die von

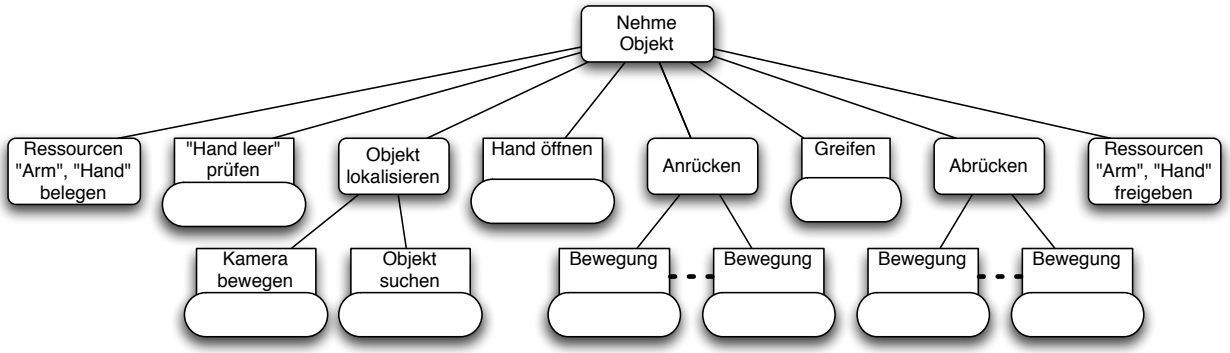


Abbildung 5.2: Schematische Darstellung eines Flexiblen Programms zum Greifen eines Objekts. Runde Operatoren beinhalten physische Aktionen des Roboters. Zusätzlich zu den Operatoren, die im Makrooperator enthalten sind, sind hier Ressourcenanfragen (belegen/freigeben), sensorische Aktionen (Objekt lokalisieren) und zusätzliche Bewegungen (Hand öffnen) eingefügt. Alle physischen Aktionen können zudem fehlschlagen, so dass zu jeder dieser Aktionen eine Fehlerbehandlung beinhaltet sein muss (nicht dargestellt).

einem Sensorsystem aus einer oder mehreren Vorführungen extrahiert wurden, sind nur von der Vorführung, der Sensorik und den zu Grunde liegenden Beobachtungsverfahren abhängig. Im Gegensatz dazu ist ein Flexibles Programm notwendigerweise abhängig von den Eigenschaften der Zielplattform, also dem ausführenden Roboter. Dies ergibt sich direkt aus der Forderung, dass Roboterprogramme darauf ausführbar sein sollen; sie müssen damit explizit oder implizit Wissen über die Möglichkeiten und Einschränkungen des Roboters beinhalten, sind also roboterspezifisch. Daraus folgt wiederum unmittelbar, dass im Abbildungsprozess roboterspezifisches Wissen vonnöten ist und dieser Prozess also von den Eigenschaften der Zielplattform (des Roboters) abhängt.

Tabelle 5.1 fasst den Vergleich zwischen Makrooperator und Flexiblem Programm zusammen. Dabei werden die Eigenschaften des Aktionsumfangs, der Fehlerbehandlung, der sensorischen Aktionen und des Arbeitsraums betrachtet.

Im Folgenden wird detailliert auf die Darstellung und das Format von Makrooperatoren und Flexiblen Programmen eingegangen. Diese bilden die Grundlage für den Abbildungsprozess.

	Makrooperator	Flexibles Programm
Aktionsumfang	Für das Ziel relevante Aktionen	Vollständige Roboterbefehlskette
Fehlerbehandlung	Nicht vorhanden	Muss enthalten sein
Sensorische Aktionen	Nicht vorhanden	Muss enthalten sein
Definitionsraum, Arbeitsraum	Mensch	Roboter

Tabelle 5.1: Gegenüberstellung von Makrooperatoren und Flexiblen Programmen

5.1.1 Format von Makrooperatoren

Die Organisation des Handlungswissens in Form von Bäumen wurde bereits in Kapitel 3.1.1 vorgestellt. Dabei wird die Reihenfolge der auszuführenden Aktionen durch eine Tiefensuche innerhalb des Baums von links nach rechts bestimmt (siehe auch Anhang B). Die Reihenfolge spielt also eine essentielle Rolle bei der Darstellung einer Handlung.
In der hierarchischen XML-Beschreibung des Makrooperators werden die Nachfolgeelemente (Kindknoten) eines Operators durch ein gesondertes Element festgelegt. Die Ausführungsreihenfolge innerhalb dieser Nachfolgeoperatoren wird durch ein weiteres Element beschrieben. Dieses enthält Vorrangrelationen (Präzedenzen) zwischen den angegebenen Nachfolgern (siehe Kapitel 3.1.2).
Die Effekte eines Makrooperators werden durch seine jeweiligen Vor- und Nachbedingungen beschrieben (siehe Kapitel 3.1.1). Dabei wird eine Vor- oder Nachbedingung B als Disjunktion von verschiedenen möglichen Umweltsituationen (*Kontexten*) K_i aufgefasst. Jeder Kontext wiederum wird beschrieben durch die konjunktive Verknüpfung einer Menge an Relationen R_{ij}. Es gilt also:

$$B = K_1 \vee K_2 \vee \ldots \vee K_n \tag{5.1}$$

$$K = R_1 \wedge R_2 \wedge \ldots \wedge R_m \tag{5.2}$$

Eine spezifizierte Vorbedingung für einen Makrooperator entspricht der schwächsten Bedingung, die erfüllt sein muss, damit der Operator ausführbar ist. Die Nachbedingung stellt die Effekte des Operators in der Umwelt dar. Sie kann dargestellt werden durch die Menge der Relationen, die zum Endzeitpunkt gelten, ohne diejenigen Relationen, die schon am Startzeitpunkt gültig waren:

$$\text{Effekt} = \text{Zustand}_{\text{Ende}} \setminus (\text{Zustand}_{\text{Anfang}} \cap \text{Zustand}_{\text{Ende}}) \tag{5.3}$$

Für die elementaren Operatoren beinhaltet der Makrooperator die aus der Vorführung extrahierten Parameter. Dies sind insbesondere Informationen über die Trajektorien und Griffe. Während der Erzeugung von Makrooperatoren werden alle Trajektorien relativ zu den jeweils beteiligten Objekten gespeichert. Dies ermöglicht die Anwendung der jeweiligen Datensätze auch in veränderten Situationen, wenn sich Objekte an anderen Positionen befinden.

5.1.2 Format von Flexiblen Programmen

Das Konzept und die Struktur Flexibler Programme ist in Kapitel 4.2.3 ausführlich dargestellt. Gespeichert werden Flexible Programme in einem XML-Format, das entsprechend der Struktur des Handlungswissens aufgebaut wird. Da sowohl Flexible Programme als auch XML selbst streng hierarchisch aufgebaut sind, eignet sich diese Darstellung hinreichend.
An dieser Stelle soll nun der Unterschied zur Darstellung der Makrooperatoren in der Ordnung der Teilhandlungen demonstriert werden. Hierauf gründet sich der in Kapitel 5.2 vorgestellte Abbildungsprozess.
Handlungsalternativen sind im Makrooperator implizit durch Vorrangrelationen dargestellt. Im Gegensatz dazu werden sie in Flexiblen Programmen als explizite Alternativlösungen dargestellt, um eine direkte Auswahl zu erlauben. Dabei wird der Unterschied zu Makrooperatoren deutlich: Jede mögliche Reihenfolge ist explizit kodiert. Dadurch ist es während der

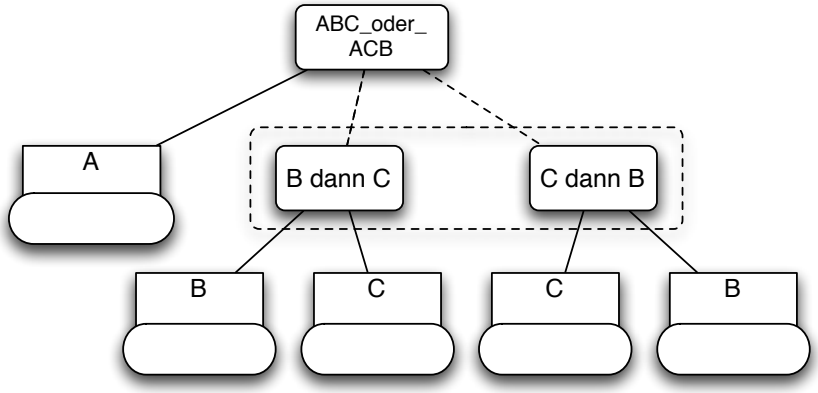


Abbildung 5.3: Darstellung eines FPs mit unterschiedlichen Ordnungmöglichkeiten zweier Operatoren. Alle Möglichkeiten sind explizit enthalten.

Ausführung möglich, zwischen Alternativen zu entscheiden. Abbildung 5.3 zeigt ein solches Beispiel.
Die Vor- Während- und Nachbedingungen sind in Flexiblen Programmen entsprechend der Modellierung in Kapitel 4.3.2 in einem XML-Format abgelegt. Dieses ist der Darstellung von Bedingungen in Makrooperatoren sehr ähnlich.
Die Abbildung von Makrooperatoren auf Flexible Programme sollte demnach einen in XML gegebenen Makrooperator in ein ebenfalls in XML definiertes FP konvertieren können. Dieser Prozess wird in den nächsten Kapiteln vorgestellt.

5.2 Der Abbildungsprozess im Überblick

Die Erstellung ausführbarer Roboterprogramme aus Makrooperatoren wird durch eine Abbildung mit der Einführung zusätzlichen Hintergrundwissens über das Zielsystem dargestellt. Dabei werden die in Kapitel 5.1 vorgestellten Eigenschaften der jeweiligen Repräsentation zu Grunde gelegt. Die Abbildung wird dabei in mehrere Schritte zerlegt. Abbildung 5.4 skizziert den Ablauf. Im Folgenden werden die einzelnen Schritte näher erläutert (siehe auch [Knoop 07a, Knoop 05b, Andersen 05]).

1. Die *strukturelle Abbildung* bildet den Inhalt des Makrooperators auf die Struktur der Zielbeschreibung ab. Dabei wird eine vorläufige Beschreibung der Handlung aufgebaut, die die Elemente des Makrooperators in der Struktur eines Flexiblen Programms beinhaltet. Insbesondere werden dabei die Informationen über Nachfolgeknoten, Präzedenzrelationen, sowie explizit im Makrooperator vorgegebene Parallelitäten verarbeitet. Die resultierende Beschreibung $\widehat{\mathcal{P}}'$ ist vorläufig und damit nicht durch einen Roboter ausführbar.

2. In einem zweiten Schritt wird die *Abbildung der Elementaroperatoren* durchgeführt. In der vorläufigen Handlungsbeschreibung $\widehat{\mathcal{P}}'$ werden dabei die im Makrooperator enthaltenen Elementaroperatoren durch Flexible Programme ersetzt. Diese können entweder aus einer einzelnen Aktion oder aus einem zusammengesetzten Handlungsbaum bestehen. Während dieses Schritts werden die Parameter der Elementaroperatoren in

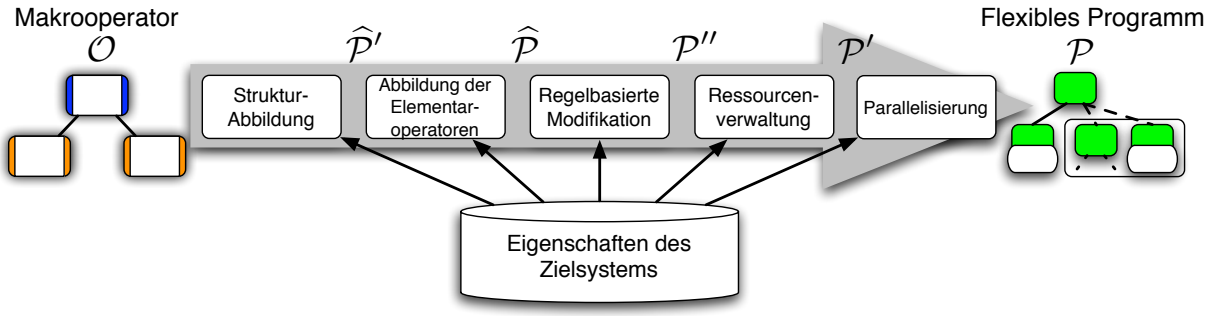


Abbildung 5.4: Die einzelnen Schritte des Abbildungsprozesses. $\mathcal{O}$ stellt den Makrooperator dar, $\mathcal{P}$ das endgültige FP, alle mit Dach bezeichneten Programme sind vorläufig und damit nicht oder nicht sinnvoll ausführbar. $\mathcal{P}''$, $\mathcal{P}'$ und $\mathcal{P}$ sind syntaktisch und inhaltlich sinnvolle Versionen des Programms in verschiedenen Optimierungsstadien.

eine ausführbare Form gebracht; dies betrifft Trajektorien, Positionen und Relationen. Ergebnis ist eine weitere vorläufige Handlungsbeschreibung $\widehat{\mathcal{P}}$. Diese enthält durch den Roboter ausführbare Aktionen in der Struktur eines Flexiblen Programms; die Ausführung ist jedoch im Allgemeinen nicht möglich, da $\widehat{\mathcal{P}}$ nur aus der Beobachtung extrahierte Elemente enthält.

3. Den Kern der Abbildung bildet die *regelbasierte Modifikation* des Handlungswissens. Dabei werden Knoten der hierarchischen Handlungsbeschreibung verändert oder hinzugefügt. Dieser Schritt soll die Ausführbarkeit der Handlung mit dem Roboter gewährleisten, es werden also sowohl sensorische als auch aktorische Handlungsteile hinzugefügt und modifiziert. Dabei werden die verwendeten Regeln unterteilt in die Gruppe der *notwendigen Regeln*, die für eine fehlerfreie Ausführung angewendet werden müssen, und in *zusätzliche* oder *optionale Regeln*, deren Anwendung nicht zwingend ist. Die optionalen Regeln erweitern die Ausführungsbeschreibung vor allem um Ausgaben für den Benutzer. Das resultierende Flexible Programm $\mathcal{P}''$ enthält alle zur Ausführung notwendigen Aktionsprimitive, ist folglich ein vollständiges Roboterprogramm.

4. Die Ausführungsumgebung ist mit einer Ressourcenverwaltung ausgestattet, die Kollisionen bei der Nutzung der Sensorik und Aktorik vermeiden hilft (siehe Kapitel 4.1.2). Entsprechend wird das erzeugte Handlungswissen auf *verwendete Ressourcen* untersucht, und es werden Elemente zur Belegung und Freigabe der Ressourcen eingefügt. Dadurch wird die reibungslose Ausführung mehrerer Flexibler Programme gleichzeitig gewährleistet, indem kritische Ressourcen jeweils nur einem der aktiven Programme zugeteilt werden. Das Flexible Programm $\mathcal{P}'$ ist voll funktionsfähig und ausführbar.

5. Der letzte Schritt der Abbildung stellt einen reinen Optimierungsschritt dar. Durch *Parallelisierung* von Teilprogrammen, die gleichzeitig ausgeführt werden können, wird die benötigte Zeit für die Ausführung reduziert und es werden unnötige Totzeiten entfernt. Hierzu wird das Programm $\mathcal{P}'$ analysiert, um mögliche Parallelisierungen zu finden und entsprechend zu formatieren.

Die einzelnen Schritte dieser Abbildung werden im Folgenden näher erläutert. Dabei wird in der Reihenfolge der Abbildung vorgegangen. Das für die Abbildung verwendete Wissen über das Zielsystem wird anschließend zusammengefasst.

5.3 Abbildung der Handlungsstruktur

Für den Abbildungsprozess wird zunächst eine Beschreibung formal definiert, in der sich die temporären Programme $\widehat{\mathcal{P}}'$ und $\widehat{\mathcal{P}}$ darstellen lassen. Die Definition lehnt sich die der Flexiblen Programme an, und ein Knoten stellt sich folgendermaßen dar (siehe auch Kapitel 4.2.3):

$$\widehat{\mathcal{P}} = \{Id, Cat, C_{pre}, C_{post}, C_{rt}, P, A, Res, Par\} \tag{5.4}$$

Dabei stellt *Id* einen eindeutigen Namen für den Knoten dar, *Cat* gibt die Klasse (Kategorie) der Handlung des Knotens an, und $C_{pre}, C_{post}, C_{rt}$ sind die Vor-, Nach- und Währendbedingungen. Analog zur Definition in Gleichung (4.1) sind P der Prospekt und A die Aktion. Hier gilt $A = \emptyset \Leftrightarrow P \neq \emptyset$. *Res* enthält die Liste der benötigten Ressourcen für die gegebene Teilhandlung, und *Par* enthält explizit zu parallelisierende Kindknoten. Hier gilt also $A \neq \emptyset \Rightarrow Par = \emptyset$. Existiert eine parallele Gruppe Par_i mit $(u, v) \in Par_i$, kann man auch $u \| v$ schreiben.

5.3.1 Aufbau des Prospekts

Zur Abbildung der Struktur einer Handlung wird nun der Makrooperator anhand einer Tiefensuche durchlaufen. Dabei wird für jeden Knoten im Makrooperator zunächst ein korrespondierender Knoten k in $\widehat{\mathcal{P}}'$ erzeugt. Für jeden Knoten k werden die Nachfolgerliste V, die Präzedenzen M und explizite Parallelitäten extrahiert. Anhand dieser Information kann der Prospekt (Definition: Siehe Kapitel 4.2.3) von k aufgebaut werden.
Zu erzeugen sind für den aktuellen Knoten k die Liste der Sitze $P = \{s_1, \ldots, s_n\}$ und der jeweiligen Kandidaten. Dabei kann ein Nachfolger v_i nur dann für den aktuellen Sitz s_j als Kandidat ausgewählt werden, wenn er in keiner derjenigen Präzedenzrelationen auf der rechten Seite vorkommt, deren linker Operator noch nicht in einem der schon konstruierten Sitze $\bar{P} = \{s_1, \ldots, s_{j-1}\}$ positioniert ist. Alle Präzedenzen, deren linker Operator nicht in $\bar{P}$ vorkommt, werden deshalb zu diesem Zeitpunkt *gültige Präzedenzen* genannt.
Sind zusätzlich folgende Forderungen an einen Nachfolger v_i erfüllt, so wird dieser als Kandidat in den aktuell bearbeiteten Sitz s_j aufgenommen:

- v_i ist nicht in $\bar{P}$ vertreten.
- Kein bisher ermittelter Kandidat für den aktuellen Knoten s_j darf sich in einer parallelen Gruppe mit v_i befinden.
- Ist v_i in einer parallelen Gruppe enthalten, so darf kein anderer Nachfolger aus der selben Gruppe auf der rechten Seite einer gültigen Präzedenz stehen.

Die erste Anforderung besagt, dass v_i bisher nicht verarbeitet wurde. Dies stellt sicher, dass jeder Nachfolger genau einmal in der Handlung existiert. Dazu wird gefordert, dass als parallel spezifizierte Knoten in unterschiedlichen Sitzen platziert werden. Dadurch wird

```
   Eingabe : k (Knoten, dessen Sitze zu berechnen sind), W (Liste der noch nicht
             verarbeiteten Nachfolger), M (Präzedenzenliste)
   Ausgabe : C (Liste mit Kandidaten für den aktuellen Sitz)
 1 Beginn
 2     C ← ∅
 3     foreach Nachfolger w_i ∈ W do
 4         Kandidat ← true
 5         foreach Präzedenzpaar (p_1, p_2) ∈ M do
 6             if w_i ≠ p_2 then continue
 7             if p_1 ∈ W then                          /* s = p_2 und (p_1, p_2) gültig */
 8                 Kandidat ← false
 9                 break
10             endif
11         endfch
12         if Kandidat = true then              /* Überprüfe parallele Gruppen. */
13             foreach parallele Gruppe G ∈ Par(k) do
14                 if w_i ∈ G then
15                     foreach Mitglied g ∈ G, g ≠ w_i do
                           /* Paralleler Knoten Kandidat für aktuellen Sitz */
16                         if g ∈C then
17                             Kandidat ← false
18                             break
19                         endif
20                         foreach Präzedenz (p_1, p_2) ∈ M do
21                             if p_2 = g and p_1 ∈ W then
22                                 Kandidat ← false
23                                 break
24                             endif
25                         endfch
26                         if Kandidat = false then break
27                     endfch
28                 endif
29                 if Kandidat = false then break
30             endfch
31         endif
32         if Kandidat = true then C ← C ∪ {s}          /* Kandidat hinzufügen */
33     endfch
34     return C
35 Ende
```

Algorithmus 5.1 : Aufbau der Kandidatenliste: `KandidatenAufbau`

sichergestellt, dass alle Knoten einer parallelen Gruppe ausgeführt werden. Die letzte Forderung schließt folgenden Fall aus: Gäbe es weitere Nachfolger $w \notin \bar{P}$ und u, sowie eine gültige Präzedenz $(u, w) \in M$, müsste wegen $v_i \| w$ auch die Präzedenz $(u, v_i) \in M$ gelten. Damit kann v_i nicht Kandidat des aktuellen Sitzes sein. Algorithmus 5.1 fasst den Prozess der Kandidatenbestimmung zusammen.
An dieser Stelle sei angemerkt, dass bedingt durch die Abarbeitung entlang einer Tiefensuche bei der Einfügung von Kandidaten diese noch nicht vollständig parametriert zur Verfügung stehen. Aus diesem Grund werden zunächst Platzhalter eingefügt, die im späteren Verlauf der strukturellen Abbildung durch erzeugte Knoten ersetzt werden.

Nach der Bestimmung der Kandidatenliste für einen Sitz wird nun der Sitz mit Kandidaten belegt. Dabei können folgende drei Fälle auftreten:

- Für den aktuellen Sitz wurde genau ein Kandidat ermittelt, der keiner parallelen Gruppe angehört. Dieser kann sofort eingefügt werden.
- Wurde für den aktuellen Sitz ein Kandidat ermittelt, der einer parallelen Gruppe angehört, wird diese Gruppe vollständig eingefügt. Dazu wird für jedes Element ein eigener Sitz erzeugt, und diese werden als parallel markiert.
- Die Kandidatenliste für den aktuellen Sitz enthält $n > 1$ Kandidaten. Diese dürfen nicht als Kandidaten zum aktuellen Sitz hinzugefügt werden, um zu verhindern, dass nur einer davon ausgeführt wird. Dieser Fall bedeutet, dass die Reihenfolge im Makrooperator nicht eindeutig festgelegt ist und mehrere Möglichkeiten existieren. Um diese verschiedenen Möglichkeiten auch im Flexiblen Programm abzubilden, wird für jeden Knoten c_i in der Liste ein zusätzlicher Verzweigungsknoten v_i^b erzeugt und als eigentlicher Kandidat dem aktuellen Sitz s_j hinzugefügt. v_i^b wird dann ein Sitz mit genau einem Kandidaten c_i hinzugefügt, und die weitere Belegung von v_i^b wird regulär anhand der beschriebenen Methode bestimmt.

Theoretisch kann auch der Fall auftreten, dass kein Kandidat gefunden wurde. Das kann allerdings ausgeschlossen werden, wenn von korrekt aufgebauten Präzedenzen im Makrooperator ausgegangen wird. Algorithmus 5.2 fasst den Aufbau der Sitze zusammen.
Nachdem der aktuelle Knoten k vollständig aufgestellt worden ist, muss er noch im eigentlichen Handlungsbaum $\widehat{\mathcal{P}}'$ verankert werden, der bis zu diesem Zeitpunkt nur einen Platzhalter enthält. Zwei Fälle können hier auftreten: Ist der aufgestellte Teilbaum der erste der Handlung, so wird $\widehat{\mathcal{P}}'$ mit k initialisiert. Ist dies nicht der Fall, wird $\widehat{\mathcal{P}}'$ vollständig durchsucht und jeder Platzhalter, der für k steht, durch k ersetzt. Das Verfahren ist in Algorithmus 5.3 dargestellt.
Die vorgestellten Algorithmen erlauben es nun, die Abbildung von der Struktur eines Makrooperators auf die der Flexiblen Programme durchzuführen. Dabei wird der Makrooperator von der Wurzel aus rekursiv traversiert. Algorithmus 5.4 gibt den vollständigen Ablauf wieder.

5.3.2 Abbildung der Bedingungsausdrücke

Aufgrund der Ähnlichkeit zwischen der Repräsentation der Bedingungen in Makrooperator und Flexiblen Programmen ist die Abbildung dieser Ausdrücke einfach. Die Abbildung

```
   Eingabe : k (Knoten, dessen Sitze aufgebaut werden), V (Nachfolgerliste),
             M (Präzedenzenliste), C (Kandidatenliste)
   Ausgabe : -
 1 Beginn
 2     n ← 0
 3     V' ← V        /* Interne Kopie. Wird für rekursive Aufrufe benötigt */
 4     while V' ≠ ∅ do
 5         if |C| = 1, C = {c} then     /* Ein Kandidat für den aktuellen Sitz */
 6             Füge c in den n-ten Sitz von k ein
 7             V' ← V' \ {c}
 8             foreach parallele Gruppe G ∈ Par(v) do
 9                 if c ∈ G then
10                     foreach Mitglied g ∈ G, g ≠ c do
11                         n ← n + 1
12                         Füge g in den n-ten Sitz von k ein
13                         V' ← V' \ {m}
14                     endfch
15                 endif
16             endfch
17             n ← n + 1
18             C ← KandidatenAufbau(k, V', M)              /* Algorithmus 5.1 */
19         else                                  /* Zwei oder mehr Kandidaten */
20             foreach Kandidat c ∈ C do
21                 Erzeuge Verzweigungsknoten v_c^b
22                 Füge v_c^b in den n-ten Sitz von k ein
23                 SitzeBelegen(v_c^b, V', M, {c})                  /* Rekursion */
24             endfch
25             V' ← ∅
26         endif
27     endw
28 Ende
```

Algorithmus 5.2 : Aufbau der Sitze eines Knotens aus den im Makrooperator vorgegebenen Einschränkungen: `SitzeBelegen`

```
   Eingabe : k (Wurzel des durch die Algorithmen 5.1 und 5.2 aufgebauten Teilbaums),
             r (Wurzel von P̂'. Muss beim ersten Aufruf ein leerer Platzhalter sein)
   Ausgabe : -
 1 begin
 2     if V(r) = ∅ and r ist Platzhalter then
                                  /* r ist noch leer. Dies ist der erste Aufruf */
 3         r ← k
 4         return
 5     endif
 6     foreach Sitz v ∈ V(w) do
 7         foreach Kandidat c ∈ C(v) do
 8             if c ist Platzhalter and Id(c) = Id(k) then
 9                 Erzeuge Kopie k' von k
10                 c ← k'
11             endif
12         endfch
13     endfch
14     foreach Sitz v ∈ V(w) do
15         foreach Kandidat c ∈ C(v) do
16             TeilBaumEinfügen(k, c)                          /* Rekursion */
17         endfch
18     endfch
19 end
```

Algorithmus 5.3 : Einfügen von aufgebauten Teilbäumen in $\widehat{\mathcal{P}}'$: `TeilBaumEinfügen`

entspricht einer syntaktischen Konvertierung, wobei zwischen unären, binären und ternären Bedingungen unterschieden werden muss.
In den Bedingungen des Makrooperators sind nur die Operatoren *Konjunktion*, *Disjunktion* und *Negation* enthalten. Diese werden von der Bedingungsbeschreibung der Flexiblen Programme unterstützt. Die Verknüpfung der Operatoren kann also vollständig übernommen werden.

5.4 Abbildung der Elementaroperationen

Elementaroperationen (EOs) des Makrooperators enthalten beobachtete elementare Bewegungen oder andere Aktionen des Menschen. Eine entsprechende Roboteraktion ist nicht notwendigerweise elementar; sie kann aus mehreren Einzelaktionen zusammengesetzt sein.
Dieser Schritt wird durch eine explizite Abbildung von Elementaroperatoren auf bekannte Flexible Programme gelöst. Für jeden Elementaroperator gibt es also ein Flexibles Programm auf der Seite der Ausführung, das an dieser Stelle in $\widehat{\mathcal{P}}'$ eingefügt wird, um $\widehat{\mathcal{P}}$ zu erhalten.
Diese Einschränkung kann deshalb getroffen werden, weil die Menge der EOs klein (Griffe und Bewegungen) ist, und sich dadurch zusätzlich ein Vorteil ergibt: Die vorgegebenen Roboterprogramme können auf anderem Wege optimiert oder gelernt werden. Durch die Re-

```
   Eingabe : O (Makrooperator)
   Ausgabe : k_O (vorläufiges Flexibles Programm P̂')
 1 begin
 2     Erzeuge Knoten k_O mit Id(k_O) = Name von O
 3     Konstruiere Nachfolgerliste V und Präzedenzenliste M aus O
 4     if explizite Parallelisierungsinformationen in O then
 5         Belege Par(k_O) entsprechend
 6     else Par(k_O) ← ∅
 7     C ← KandidatenAufbau(k_O, V, M)                    /* Algorithmus 5.1 */
 8     SitzeBelegen(k_O, V, M, C)                         /* Algorithmus 5.2 */
 9     if P̂' = ∅ then
10         Erzeuge P̂' mit k_O als Wurzel
11     else TeilBaumEinfügen(k_O, Wurzel von P̂')          /* Algorithmus 5.3 */
12     if V = ∅ then return k_O
13     foreach Nachfolger v ∈ V do MakrooperatorAbbildung(v)
                                                  /* Rekursiver Aufruf */
14     return k_O
15 end
```

Algorithmus 5.4 : Vollständige Strukturabbildung unter Verwendung der Algorithmen 5.1, 5.2 und 5.3

ferenzierung der FPs untereinander wird die Optimierung eines solchen Basis-FPs für alle Programme übernommen, in denen es eingebunden ist.
Um $\widehat{\mathcal{P}}'$ zu erhalten, wird also jeder Elementaroperator auf ein Teilprogramm im Flexiblen Programm abgebildet. Wie erwähnt, muss diese Abbildung grundsätzlich a priori bekannt sein. Dabei können zwei Fälle auftreten:

1. Ein EO wird auf eine primitive Aktion im FP abgebildet, die durch ein Blatt im Handlungsbaum repräsentiert wird. Die korrespondierende Handlung kann direkt für die Ausführungsbeschreibung erzeugt werden.

2. Der EO wird durch eine zusammengesetzte Teilhandlung auf Ausführungsseite repräsentiert. Diese Teilhandlung ist aus mehreren verknüpften primitiven Roboteraktionen aufgebaut. In diesem Fall wird der EO durch den vollständigen Teilbaum ersetzt.

In beiden Fällen ist die Abbildung der Knotentypen a priori gegeben. Die Abbildung der *Parameter* aus dem Makrooperator wird in Kapitel 5.4.1 diskutiert.
Dieser Abbildungsschritt basiert auf der Annahme, dass jeder EO im Makrooperator auf eine entsprechende Roboteraktion abgebildet werden *kann*. Ist für einen Elementaroperator keine Abbildung definierbar, ist eine Handlung, die diesen EO enthält, mit dem Zielsystem aufgrund seiner Eigenschaften nicht durchführbar. Dies stellt dann keine Einschränkung der Abbildung, sondern eine Einschränkung des Zielsystems dar. Die Art dieser Abbildung wird nicht eingeschränkt; $1:1$ Abbildungen sind genauso möglich wie Abbildungen mit $1:n$. Abbildungen, die mehrere EOs auf eine primitive Roboteraktion abbilden, werden hier

nicht betrachtet. In realen Szenarien ist es unwahrscheinlich, dass beobachtbare Handlungen feinkörniger sind als die für die Ausführung benötigte Granularität.

5.4.1 Abbildung der Parameter

Die in den Elementaroperatoren enthaltenen Parameter werden durch eine Abbildung in den Raum der Ausführung transformiert und in den Blättern des Flexiblen Programms $\widehat{\mathcal{P}}'$ abgelegt. Zusätzlich zur Ordnung und Hierarchisierung der Handlung enthält der Makrooperator folgende Daten:

- Trajektorien sind als geometrische Pfade in Bewegungsoperationen gespeichert. Diese werden während der Vorführung linearisiert und in Form von linearen Bewegungen, Splines oder anderen parametrisierten Kurven gespeichert (siehe Abbildung 5.5).

- Die Beobachtungseinheit differenziert zwischen 16 verschiedenen Griffen, die nach der Cutcosky-Hierarchie geordnet sind (siehe [Cutkosky 89]). Der Typ und die Parameter jedes Griffs werden während der Vorführung gespeichert. Die Parameter bestehen aus allen Fingerwinkeln, die mit Hilfe der Datenhandschuhe beobachtet wurden.

Für die Abbildung der Griffe wird an dieser Stelle nur der erforderliche Grifftyp verwendet. Aufgrund der großen Unterschiede zwischen menschlicher Hand und Robotergreifer ist es in vielen Fällen sinnvoll, auf Ausführungsseite eine erneute Greifplanung vorzunehmen. Die in der Vorführung angewendeten Griffe können dabei als Eingabeparameter dienen. Eine detaillierte Untersuchung dieses Problems findet sich in [Rogalla 02].

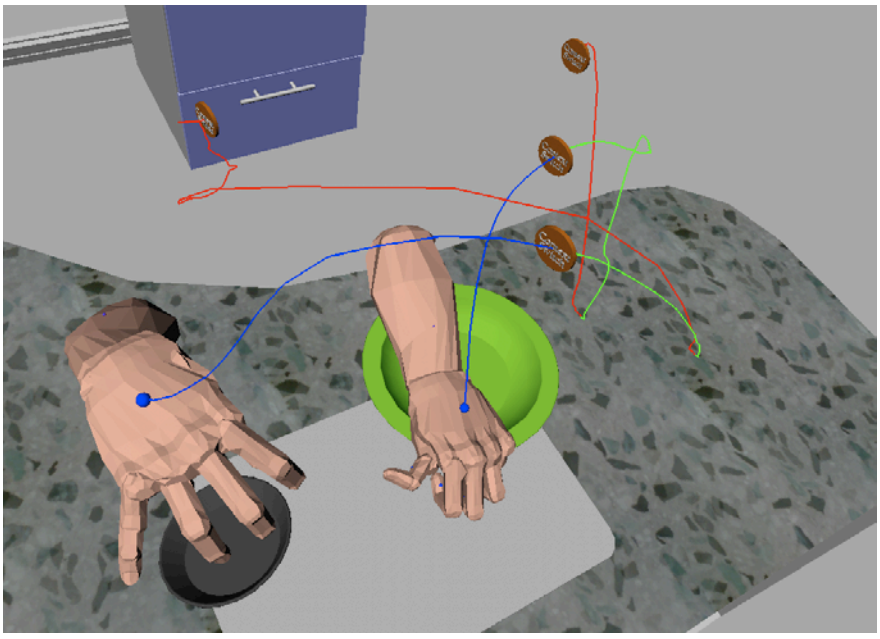


Abbildung 5.5: Trajektorien der Hände, aufgezeichnet während einer Benutzerdemonstration. Gut zu erkennen sind die Kontextwechsel, die immer auch einen Wechsel des Referenzkoordinatensystems bedeuten, da alle Bewegungen relativ zu Objekt- und Ablagepositionen gespeichert werden.

Geometrische Bewegungspfade (Trajektorien) sind in Makrooperatoren immer relativ angegeben. Sie beziehen sich auf das jeweils betroffene Objekt, das gegriffen oder abgelegt wird. Die Koordinaten sind jeweils kartesisch im Koordinatensystem der Objektaufnahme- bzw. Ablageposition angegeben (siehe Abbildung 5.5).
Während einer Bewegung wechselt also der Bewegungskontext (siehe [Schwarz 98]) von der Aufnahme- zur Ablageposition (Transport) oder von einem Objekt zum nächsten (freie Bewegung zwischen zwei Transportoperationen). Der Roboter muss also für die Ausführung die Koordinaten der Objekte kennen, um die Bewegungen ausführen zu können.
Bei der Abbildung der Trajektorien können die Positionen und Bewegungen also nur relativ zu Objektkoordinaten angegeben werden, die zur Übersetzungszeit unbekannt sind. Die Lokalisierung des jeweiligen Objekts ist als sensorische Aktion im Makrooperator nicht enthalten; dies wird in Kapitel 5.5 betrachtet.

5.5 Regelbasierte Modifikation

Das vorläufige Roboterprogramm $\widehat{\mathcal{P}}$ enthält alle Information, die aus dem zugehörigen Makrooperator extrahiert werden kann. Trotzdem ist $\widehat{\mathcal{P}}$ im Normalfall nicht erfolgreich ausführbar aus den in Kapitel 5.1 genannten Gründen: Es fehlen sowohl sensorische Aktionen zur Erkennung der Szene und Reduktion der Unsicherheiten als auch weitere physische Aktionen. Deshalb wird das aus dem Makrooperator erzeugte Programm $\widehat{\mathcal{P}}$ weiter verarbeitet. Dazu wird $\widehat{\mathcal{P}}$ regelbasiert manipuliert, es werden also Programmteile verändert und erweitert. Dies geschieht anhand festgelegter Regeln, die als Hintergrundwissen über das ausführende System zur Verfügung stehen.
Ein einsichtiges Beispiel zeigt dies anhand der Greifersteuerung. Abbildung 5.6 zeigt eine vereinfachte Darstellung einer Greifaktion, wie sie aus der Vorführung extrahiert wurde und in $\widehat{\mathcal{P}}'$ dargestellt ist. Die Sequenz *Anrücken* $\rightarrow$ *Greifen* $\rightarrow$ *Abrücken* ist für die Analyse einer Handlung ausreichend. Die Ausführung jedoch benötigt eine Reihe weiterer Aktionen. Dazu gehört, die Hand vor dem Greifen zu öffnen, wie Abbildung 5.7 zeigt. Dabei ist es wichtig, dass die Hand vor der Anrückphase in eine offene Stellung gebracht wird, um Kollisionen zu vermeiden.
Abhängigkeiten dieser Art sind nicht aus einer Vorführung extrahierbar und werden deshalb wissensbasiert in das Handlungswissen hineingebracht.

Formal gesehen ist $\widehat{\mathcal{P}}$ ein *attributierter Baum* (siehe Anhang B). Dies legt die Werkzeuge der Graphentheorie nahe. Regeln zur Manipulation von $\widehat{\mathcal{P}}$ werden damit wie folgt definiert:

Definition 5.1 (Regel zur Baummodifikation) $\widehat{\mathcal{P}}$ ist ein attributierter Baum. Dann wird eine Regel $\mathcal{R}$ auf $\widehat{\mathcal{P}}$ definiert durch das 5-tupel (N, P, I, C, A). Dabei gilt:

- N ist ein eindeutiger Bezeichner.
- $P = \{P_1, \ldots, P_n\}$ ist eine Menge an attributierten Bäumen, die auch als Muster (engl.: *Patterns*) dienen. Die Regel $\mathcal{R}$ soll angewendet werden, sobald ein Muster P_i in $\widehat{\mathcal{P}}$ gefunden wird.
- $I = \{i_1, \ldots, i_n\}$ bezeichnet jeweils einen Knoten aus P_i, der als Einstiegsknoten für die Baummanipulation dient. Es gilt also für $j = 1 \ldots n$: $i_j \in P_j$.

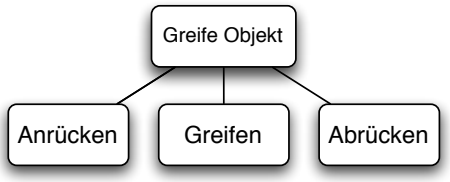


Abbildung 5.6: Vereinfachte Darstellung einer Greifaktion in $\widehat{\mathcal{P}}$

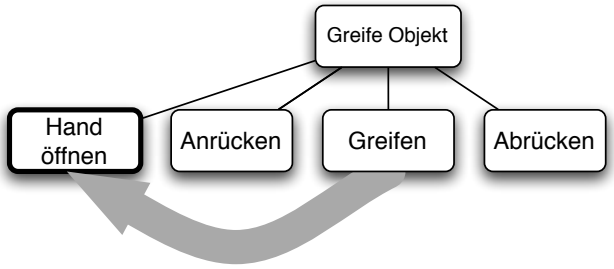


Abbildung 5.7: Regelbasierte Erweiterung der Greifaktion in $\mathcal{P}''$. Durch das vorherige Öffnen des Greifers wird sichergestellt, dass zum Zeitpunkt des Griffs die Hand sich in einer dazu geeigneten Pose befindet. Die Abhängigkeit der Aktionen ist mit dem grauen Pfeil markiert.

- Die Bedingung C (engl.: *Condition*) stellt eine zusätzliche Ausführungsbedingung für die Regel dar, die erfüllt sein muss, damit $\mathcal{R}$ ausgeführt wird. In vielen Fällen gilt allerdings $C = \texttt{true}$.
- A ist die eigentliche Aktion der Regel zur Manipulation des Handlungsbaums. A ist definiert als eine Folge von elementaren Baummanipulationen. Dies sind Löschen und Hinzufügen von Knoten zum Baum $\widehat{\mathcal{P}}$. A kann auch Kontrollstrukturen beinhalten.

Bei der Anwendung mehrerer Regeln auf denselben Graphen stellt sich die Frage nach der Anwendungsreihenfolge. Graphenmanipulationssysteme, bei denen das Ergebnis unabhängig von der Reihenfolge der Manipulationen ist und damit nur vom Eingabegraphen abhängt, werden als ungeordnete Graphenmanipulation bezeichnet. Dieser Fall ist für das vorliegende Problem nicht gegeben; die Abhängigkeiten zwischen den aufgestellten Regeln müssen also betrachtet werden.
Offensichtlich besteht eine Abhängigkeit zwischen zwei Regeln $\mathcal{R}_1$ und $\mathcal{R}_2$ dann, wenn bei gleichem Eingabebaum die Betrachtung der Reihenfolge $\mathcal{R}_1\mathcal{R}_2$ und der Reihenfolge $\mathcal{R}_2\mathcal{R}_1$ zu unterschiedlichen Ergebnissen führt. Dabei kann einer der drei folgenden Fälle auftreten:

1. $\mathcal{R}_1$ und $\mathcal{R}_2$ sind auf dem Eingabebaum ausführbar. Die Ausführung in unterschiedlichen Reihenfolgen führt zu unterschiedlichen Ergebnissen.
2. Auf dem Eingabebaum ist zunächst nur $\mathcal{R}_1$ ausführbar. Durch diese Ausführung wird jedoch auch $\mathcal{R}_2$ anwendbar. $\mathcal{R}_1$ *aktiviert* $\mathcal{R}_2$.
3. Zunächst sind beide Regeln ausführbar. Die Anwendung von $\mathcal{R}_1$ führt jedoch dazu, dass $\mathcal{R}_2$ nicht mehr ausführbar ist. $\mathcal{R}_1$ *deaktiviert* also $\mathcal{R}_2$.

Um die Abhängigkeit zwischen Regeln definieren zu können, soll zunächst noch der *Beitrag* einer Regel definiert werden:

Definition 5.2 Der Beitrag einer Regel $\mathcal{R}$ wird beschrieben durch den gewünschten Effekt in der Handlungsbeschreibung. Der Beitrag beschreibt die Eigenschaften, die im Sinne der erfolgreichen Ausführung der Handlung erwünscht sind.

Für die vorliegende Fragestellung ist es nicht notwendig, die Eindeutigkeit des Handlungsbaums nach Anwendung der Regeln in verschiedenen Reihenfolgen zu gewährleisten. Es ist ausreichend zu gewährleisten, dass die *Beiträge* der Regeln erfüllt sind. Damit kann die Abhängigkeit zweier Regeln definiert werden.

Definition 5.3 Eine Abhängigkeit zwischen zwei Regeln $\mathcal{R}_1$ und $\mathcal{R}_2$ besteht genau dann, wenn eine der folgenden Bedingungen erfüllt ist:

- $\mathcal{R}_1$ aktiviert $\mathcal{R}_2$ oder umgekehrt,
- $\mathcal{R}_1$ deaktiviert $\mathcal{R}_2$ oder umgekehrt,
- Bei Anwendung der Reihenfolge $\mathcal{R}_1\mathcal{R}_2$ zerstört $\mathcal{R}_2$ den Beitrag von $\mathcal{R}_1$ oder umgekehrt.

Augenscheinlich ist klar, dass jede anwendbare Regel auf einem Handlungsbaum auch angewendet werden soll, um eine korrekte Handlungsbeschreibung $\mathcal{P}''$ zu erhalten. Dies wird durch folgende Direktive erreicht:

> Alle zu einem beliebigen Zeitpunkt potentiell anwendbaren Regeln auf einem Handlungsbaum sollen auch angewendet werden. Potentiell anwendbar sind Regeln, die entweder direkt ausführbar sind oder von einer anderen potentiell anwendbaren Regel aktiviert werden.

Mit dieser Forderung können die Regeln für die Anwendung in eine Reihenfolge gebracht werden, die folgenden Anforderungen entspricht:

- $\mathcal{R}_1$ aktiviert $\mathcal{R}_2$: $\mathcal{R}_1$ muss vor $\mathcal{R}_2$ ausgeführt werden.
- $\mathcal{R}_1$ deaktiviert $\mathcal{R}_2$: $\mathcal{R}_2$ muss vor $\mathcal{R}_1$ ausgeführt werden.
- $\mathcal{R}_1$ zerstört den Beitrag von $\mathcal{R}_2$: $\mathcal{R}_1$ muss vor $\mathcal{R}_2$ ausgeführt werden.

Zwischen den existierenden Regeln können auf diese Weise alle Ordnungsbeziehungen aufgestellt werden. Dadurch ergibt sich für die Ausführung der Regeln eine Teilordnung, die mit Hilfe eines Vorranggraphen ausgedrückt werden kann. Dabei ist zu beachten, dass es eine sinnvolle Reihenfolge der Regelausführung nur dann gibt, wenn der Vorranggraph keine Zyklen enthält (siehe Anhang B). Dies kann durch geeignete Verfahren überprüft werden (siehe [Andersen 05]).

5.5.1 Anwendung von Regeln

Ist eine geeignete Reihenfolge der anzuwendenden Regeln gefunden, können diese auf den Handlungsbaum $\widehat{\mathcal{P}}$ angewendet werden. Dabei wird für jede Regel zuerst nach einem *auslösenden Muster* gesucht. Dieses Muster stellt einen Handlungsbaum dar, der aus keinem, einem oder mehreren Knoten bestehen kann. An allen Stellen des Handlungsbaums, die das auslösende Muster beinhalten, wird weiter überprüft, ob die Regel angewendet werden soll. Dafür wird die zusätzliche Ausführungsbedingung überprüft. Ist diese WAHR, so wird der in der Regel enthaltene Einstiegsknoten I (dieser muss Teil des auslösenden Musters sein) aufgesucht und von dort aus der Aktionsteil A der Regel ausgeführt. Abbildung 5.8 verdeutlicht diesen Ablauf.

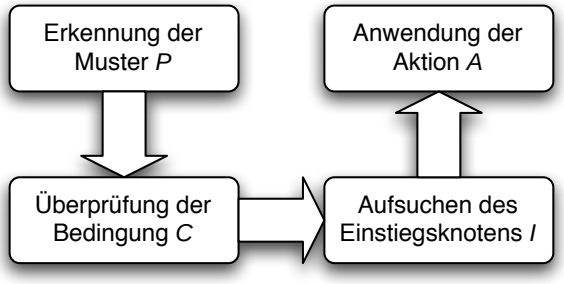


Abbildung 5.8: Ablauf bei der Anwendung von Manipulationsregeln auf einen Handlungsbaum

Der Baum $\widehat{\mathcal{P}}$ wird von einem Suchalgorithmus traversiert. Dabei werden Teilbäume gesucht, deren Wurzel mit der eines auslösenden Musters der anzuwendenden Regel übereinstimmt. Ein Teilbaum von $\widehat{\mathcal{P}}$ wird genau dann als übereinstimmend mit dem Muster P_i gewertet, wenn die Wurzel l_0 von P_i einem gefundenen Knoten a in $\widehat{\mathcal{P}}$ entspricht und es für jeden Knoten $l_j \in P_i$ einen Knoten $b_j \in \widehat{\mathcal{P}}$ gibt, so dass die Beziehung zwischen l_j und l_0 der Beziehung zwischen b_j und a entspricht (siehe Algorithmus 5.5).
Bei allen gefundenen Teilbäumen wird in einem weiteren Schritt die Ausführungsbedingung überprüft. Diese muss zusätzlich als WAHR evaluiert werden, um die Regel anwenden zu können. Der Inhalt dieser Bedingung ist nicht weiter eingeschränkt, und kann von verschiedenen Parametern abhängen. Beispiele sind die momentane Baumtiefe, die Entscheidung ob der aktuelle Knoten die Wurzel, ein innerer Knoten oder ein Blatt ist, oder die Kategorie des Knotens. Kapitel 5.5.2 zeigt konkrete Beispiele anhand der verwendeten Regeln.

5.5.2 Verwendete Regeln

Die entwickelten Regeln folgen Definition 5.1. Danach besteht eine Regel aus einem Bezeichner N, den auslösenden Mustern P_i, jeweils einem zugehörigen Einstiegsknoten I, einer zusätzlichen Bedingung C und der Aktion A zur Manipulation des Handlungsbaums. Im Folgenden werden die eingesetzten Regeln aufgezeigt. Sie teilen sich, wie bereits in Kapitel 5.2 beschrieben, in *notwendige* und *optionale* Regeln. Erstere sind für eine Ausführung des Flexiblen Programms notwendig; letztere dienen einer besseren Kommentierung des Programms und fügen hauptsächlich Ausgaben für den Benutzer ein.

```
   Eingabe : R (Regel), Eingabebaum P̂
   Ausgabe : I (Menge von Einstiegsknoten je gefundenem Muster)
 1 begin
 2     I ← ∅
 3     foreach Knoten v ∈ P̂ do                                   /* Baumsuche */
 4         foreach Muster P ∈ P(R) do          /* Vergleich mit allen Mustern */
 5             match ← false
 6             r_P ← Wurzel von P
 7             i_P ← Einstiegsknoten von P                        /* i_P ∈ I(R) */
 8             if Cat(r_P) = Cat(v) then
 9                 match ← true
10                 foreach Blatt l von P do
11                     if not ∃ a ∈ P̂ : a in P̂ entspricht l in P then
                                          /* Kindknoten stimmen nicht überein */
12                         match ← false
13                         break
14                     endif
15                 endfch
16                 if match = true then
17                     I ← I ∪ {m(i_P)}                          /* Muster gefunden */
18                 endif
19             endif
20         endfch
21     endfch
22     return I
23 end
```

Algorithmus 5.5 : Mustererkennung für die Regeln

Die Schreibweise *Knotentyp A* $\longrightarrow$ *Knotentyp B* bedeutet, dass B ein Kind von A ist. Entsprechend gilt bei $A \longrightarrow (B, C)$, dass B und C Kinder von A sind.

Öffnen der Hand vor Greifoperationen

Abbildung 5.6 und 5.7 haben bereits schematisch die Notwendigkeit einer solchen Regel gezeigt. Vor einer Greifoperation soll die Hand entsprechend geöffnet werden. Es genügt allerdings nicht, das Öffnen direkt vor der Greifoperation durchzuführen; in einem solchen Fall wäre der Robotergreifer bereits in unmittelbarer Nähe zum Zielobjekt und würde so mit hoher Wahrscheinlichkeit eine Kollision auslösen. Die Aktion zum Öffnen muss also *vor* dem Anrücken eingefügt werden. Dabei sind verschiedene Fälle zu berücksichtigen, wie in der Aktion deutlich wird. Dies ist eine *notwendige* Regel.

N Öffnen der Hand vor Greifoperationen
P Knoten vom Typ *greifen*
I Knoten vom Typ *greifen*
C WAHR
A Folgende Fälle werden berücksichtigt:

1. Der Knoten *greifen* hat keinen linken Geschwisterknoten, oder dieser ist eine Aktion der Roboterhand. Füge *Hand öffnen* direkt als linken Nachbarn von *greifen* ein.

2. Der linke Nachbar von *greifen* enthält eine Bewegungsaktion, die nicht die Roboterhand verwendet. Füge *Hand öffnen* direkt als linken Nachbarn dieses Knotens ein.

3. Der linke Nachbar von *greifen* ist kein Blatt, und der darunterliegende Teilbaum enthält eine Aktion der Hand. Füge *Hand öffnen* als rechten Nachbarn des letzten Blatts in den Teilbaum ein, das die Hand verwendet.

Beitrag Vor der auslösenden Aktion *greifen* und nach der letzten vorherigen Aktion der Hand wird diese geöffnet.

Schließen der Hand nach Ablage eines Objekts

Diese Regel ist analog zur vorhergehenden aufgebaut, und dient dazu, Kollisionen während einer Bewegung des Arms oder des gesamten Roboters zu vermeiden. Dies ist eine *notwendige* Regel.

N Schließen der Hand nach Objektablage
P Muster *ablegen* $\longrightarrow$ *zurückziehen* (Die englischen Originalnamen sind *place* und *retreat*)
I Knoten vom Typ *ablegen*
C WAHR
A Füge direkt nach *zurückziehen* einen Knoten *Hand schließen* ein.
Beitrag Direkt nach dem Öffnen und Zurückziehen wird die Hand geschlossen.

Nach Ablegen eines Objekts den Arm in eine sichere Position bewegen

Nach einer abgeschlossenen Manipulation soll der Arm in die Ausgangsposition bewegt werden. Dies dient vor allem der Sicherheit während weiterer Bewegungen, wie zum Beispiel dem Verfahren der Plattform. Dies ist eine *notwendige* Regel.

N Sicherheitsposition für den Arm
P Muster *ablegen* $\longrightarrow$ *Hand schließen*
I Knoten vom Typ *ablegen*
C WAHR
A Füge direkt nach *Hand schließen* einen Knoten *Arm bewegen* ein mit einer vorgegebenen Endposition.
Beitrag Nach einer abgeschlossenen Manipulation wird der Arm in eine definierte Position gebracht.

Lokalisierung von Objekten

Wie bereits in Kapitel 5.1 beschrieben, enthält ein Makrooperator keine Aktionen zur Feststellung des Umweltzustands. Diese Aktionen werden regelbasiert eingefügt. Dabei werden alle Aktionen gesucht, die Objektpositionen benötigen. Dies sind Aktionen vom Typ *nehmen* (engl.: *pick*) und *transportieren* (engl.: *transport*), wobei beim ersteren eine Objektposition, beim letzteren eine Ablageposition benötigt wird. Diese wird durch den eingefügten Knoten erzeugt und an entsprechender Stelle für die nachfolgende Aktion abgelegt. Dies ist eine *notwendige* Regel.

N Objekt finden und lokalisieren
P Muster *nehmen*, *transportieren*
I Knoten vom Typ *nehmen*, *transportieren*
C WAHR
A Füge direkt vor *nehmen* bzw. *transportieren* einen Knoten *Objekt lokalisieren* ein
Beitrag Jeder *nehmen* und jeder *transportieren* Knoten haben einen linken Nachbarn *Objekt lokalisieren*.

Bewegen des Roboters

Es kann nicht eindeutig und sicher geschlossen werden, dass ein zu greifendes Objekt sich tatsächlich im Arbeitsraum des Roboterarms befindet. Deshalb baut diese Regel auf der *Lokalisierung von Objekten* auf und bestimmt mit Hilfe der Armfunktionalität zum Simulieren von Bewegungen, ob die Position angefahren werden kann. Ist dies nicht der Fall, wird heuristisch eine bessere Position für den Roboter bezüglich des Objekts bestimmt und diese mit der beweglichen Plattform angefahren. Dies ist eine *notwendige* Regel.

N Bewegen des Roboters für den Greifvorgang
P Muster *Objekt nehmen und ablegen*(engl.: *pick and place*) $\longrightarrow$ *(Objekt lokalisieren, nehmen)*
I Knoten vom Typ *Objekt nehmen und ablegen*
C WAHR

A Füge zwischen *Objekt lokalisieren* und *Objekt nehmen* einen Knoten *Überprüfen der Position und gegebenenfalls fahren* ein.

Beitrag Der Knoten *Objekt nehmen und ablegen* besitzt die Kindknoten *Objekt lokalisieren*, *Überprüfen der Position und gegebenenfalls fahren* und *nehmen* in dieser Reihenfolge.

Vor dem Fahren den Arm in eine sichere Position bringen

Diese Regel berücksichtigt erneut Sicherheitsaspekte, und soll sicherstellen, dass der Arm sich in einer definierten Position befindet, sobald sich die Plattform bewegt. Diese Regel ist *notwendig*, da sie die Funktionalität des Programms selbst sicherstellt. Sie muss aus Sicherheitsgründen immer angewendet werden.

N Bewegen des Arms in eine sichere Position

P Muster *fahren*

I Knoten vom Typ *fahren*

C WAHR

A Füge vor *fahren* einen Knoten *Arm bewegen* ein, der den Arm in eine definierte Position bewegt.

Beitrag Jede Operation *fahren* hat einen linken Nachbarn vom Typ *Arm bewegen*.

Einfügen von Sprachausgaben

Es ist sinnvoll, dass der Roboter dem Menschen während der Ausführung regelmäßig Informationen gibt, in welchem Zustand er sich befindet und welche Aktion ausgeführt wird. Dies kann erreicht werden, indem vor den entsprechenden Knoten eine Sprachausgabe eingefügt wird, die die jeweilige Aktion meldet. Dabei wird eine Grenze heuristisch festgelegt, welche Granularität der Handlung noch sprachlich ausgegeben wird. Dies verhindert, dass jedes einzelne Blatt im Handlungsbaum mit einer Ausgabe versehen wird. Sprachausgaben werden zudem nur für nicht-elementare Handlungen erzeugt. Dies ist eine *optionale* Regel.

N Sprachausgabe

P Muster $\star$ (jeder Knoten wird betrachtet)

I Knoten vom Typ $\star$

C Die Granularität der Handlung wird von der Tiefe des Handlungsbaums abhängig gemacht. Sprachausgaben werden nur bis zu einer maximalen Baumtiefe eingefügt. Es muss für den Knoten v also gelten: $\mathit{Tiefe}(v) \leq K \wedge \mathit{Prospekt}\ P(v) \neq \emptyset$

A Füge einen Knoten *Sprachausgabe* als erstes Kind des Einstiegsknotens ein. Der auszugebende Text wird aus dem Name des Einstiegsknotens erzeugt.

Beitrag Jede nicht-elementare Operation bis zu einer maximalen Tiefe im Baum wird vom Roboter angekündigt.

Visuelle Ausgabe des internen Zustands

Der auf dem Roboter angebrachte Monitor ist in der Lage, den internen Zustand des Programmablaufs graphisch zu visualisieren (siehe Kapitel 7.1). Es existieren unter anderem Visualisierungen der Zustände *wartend, beschäftigt, suchend, Erfolg* und *Fehlschlag*. Die Ansteuerung dieser Ausgabe wird entsprechend in das Programm eingefügt. Diese Regel ist *optional*.

N	Visualisierung des internen Zustands
P	Muster $\star$ (jeder Knoten wird betrachtet)
I	Knoten vom Typ $\star$
C	Für den Knoten v muss gelten: v ist Wurzel der Handlung $\vee Cat(v) =$*Objekt finden*
A	Füge einen Knoten *Visualisierung* entsprechend der Situation ein.
Beitrag	Die Visualisierung des internen Zustands wird an den passenden Stellen eingefügt. Insbesondere wechselt der Zustand beim Beginn eines Programms zu *beschäftigt*, während er nach Beendigung zu *wartend* zurückkehrt.

Ausnahmebehandlung

Makrooperatoren und somit auch die daraus erzeugten Flexiblen Programme beinhalten das Handlungswissen aus erfolgreichen Demonstrationen. Daraus folgt unmittelbar, dass ein so erzeugtes FP keine Ausnahmebehandlung besitzt für den Fall, dass während der Ausführung des FPs Fehlschläge von Teilaufgaben auftreten. Da dies aber in realistischen Szenarien der Fall sein kann, wird die Ausnahmebehandlung als Hintergrundwissen durch eine Regel hinzugefügt.
Die Ausnahmebehandlung besteht aus zwei Schritten: Zuerst muss das System einen Ausnahmezustand erkennen. Daraufhin kann die Ausnahme bearbeitet werden.
Die Erkennung einer Ausnahme im Sinne des Programmablaufs geschieht über die Prüfung der Vorbedingungen von Knoten. Besitzt die reguläre Teilhandlung V den Ausdruck A als Vorbedingung, so tritt eine Ausnahme an dieser Stelle (*vor* Ausführung von V) genau dann auf, wenn $\neg A$ erfüllt ist. Aus der Darstellung der Bedingungen (siehe Kapitel 5.1.1) kann $\neg A$ mit Hilfe der DeMorgan'schen Gesetze

$$\neg(A \wedge B) = \neg A \vee \neg B \tag{5.5}$$
$$\neg(A \vee B) = \neg A \wedge \neg B \tag{5.6}$$

erzeugt werden.

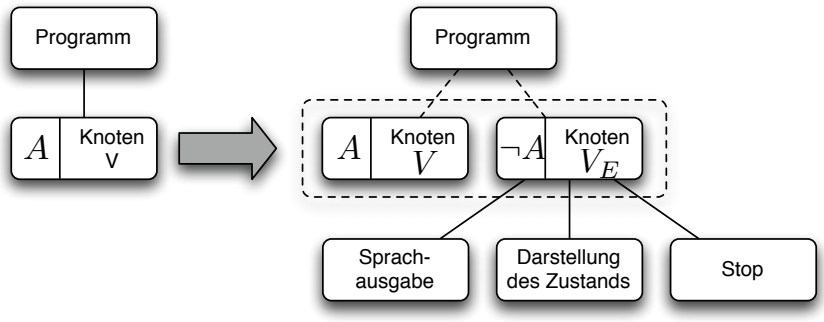


Abbildung 5.9: Einfügen der Ausnahmebehandlung für einen Knoten mit existierender Vorbedingung

Mit den so generierten Vorbedingungen kann für den Sitz des Knotens V ein weiterer Kandidat V_E für die Ausnahmebehandlung von V eingefügt werden (siehe Abbildung 5.9). Das

Teilprogramm, das von V_E beschrieben wird, enthält die Funktionen zur Behandlung der Ausnahme. Dies ist allgemein eine entsprechende Sprachausgabe, die korrekte Darstellung des Fehlerzustands in der Visualisierung, und ein kontrollierter Abbruch des FPs. Wird ein FP so beendet, befinden sich Roboter und Roboterprogramm in einem sicheren und definierten Zustand. Es können danach also weitere Programme gestartet und ausgeführt werden.

N	Ausnahmebehandlung
P	Muster $\star$ (jeder Knoten wird betrachtet)
I	Knoten vom Typ $\star$
C	WAHR
A	Erzeuge einen Knoten *Ausnahmebehandlung* und füge ihn als neuen Kandidaten im selben Sitz des Einstiegsknotens ein. Die Vorbedingung ist dabei das Komplement der Vorbedingung des Einstiegsknotens. Die Ausnahmebehandlung wird mit drei Sitzen mit jeweils einem Kandidaten gefüllt. Dies sind eine *Sprachausgabe* mit entsprechendem Text, ein Zustandswechsel der *Visualisierung* zum Zustand "Fehler", sowie einer *Stop*-Operation für die FP-Bearbeitung.
Beitrag	Jeder Knoten mit Vorbedingung besitzt einen alternativen Kandidaten, der ausgeführt wird, wenn die Vorbedingung nicht erfüllt ist. Dieser enthält die Ausnahmebehandlung in Form von Rückmeldung und kontrolliertem Stoppen.

5.5.3 Vorranggraph für die Regelanwendung

Mit der angegebenen Regelmenge können nun die Abhängigkeiten der Regeln modelliert werden. Aus diesen Abhängigkeiten wird der Vorranggraph aufgebaut, der wiederum zur Bestimmung der Reihenfolge der Regelausführung dient (siehe Kapitel 5.5).
Abbildung 5.10 zeigt den Vorranggraphen für die in Kapitel 5.5.2 angegebenen Regeln. Die Abhängigkeiten wurden dabei manuell erstellt, was aufgrund der informellen Definition der Aktion einer Regel unumgänglich ist.

5.6 Bestimmung verwendeter Ressourcen

Die Architektur, die Ausführungsumgebung sowie die Ausführungsbeschreibung selbst sind so konzipiert, dass sie mit der Ausführung mehrerer Flexibler Programme zur gleichen Zeit umgehen können. Die Ausführung mehrerer Roboterprogramme ist notwendig, sobald der Roboter mehr als eine Aufgabe zur gleichen Zeit bewältigen soll; beispielsweise kann dies ein Hol-und-Bring-Dienst sein, während parallel ein weiteres Flexibles Programm auf externe Signale (z.B. Türklingel) wartet, um Handlungen durchzuführen (beispielsweise die Tür zu öffnen).
Um Kollisionen zwischen den verschiedenen Flexiblen Programmen bei der Verwendung der Roboterkomponenten zu vermeiden, werden die Komponenten als Ressourcen betrachtet und müssen von den Flexiblen Programmen belegt und freigegeben werden. Ohne die Belegung einer Ressource kann diese nicht verwendet werden (siehe Kapitel 4.1.2).
Die Belegung und Freigabe von Ressourcen muss also in den Programmablauf integriert werden (Abbildung 5.4), um $\mathcal{P}'$ zu erzeugen. Belegung und Freigabe werde durch Aktionen erreicht, die als Blätter im Handlungsbaum integriert werden.

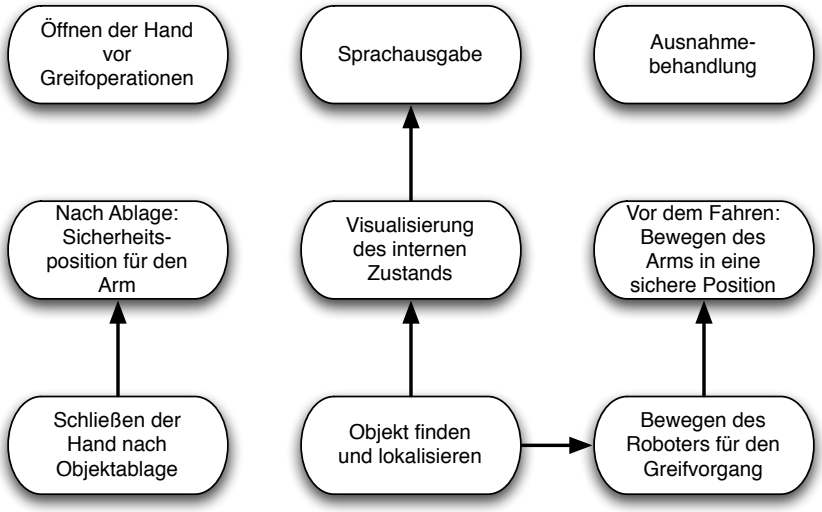


Abbildung 5.10: Vorranggraph für die angegebenen Regeln. Der Pfeil weist in Richtung der Abhängigkeit. Beispielsweise muss die Visualisierung des internen Zustands vor der Regel zur Lokalisierung von Objekten hinzugefügt werden.

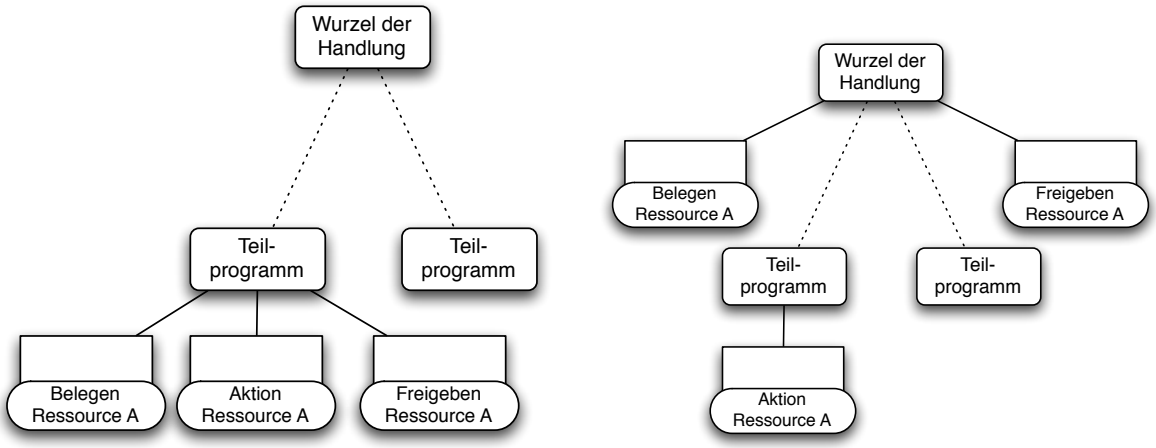
Die Belegung der Ressourcen innerhalb eines Flexiblen Programms kann nach verschiedenen Strategien durchgeführt werden. Diese unterscheiden sich darin, ob Ressourcen *lokal*, *global* oder *flexibel* belegt werden. Im Folgenden werden diese Strategien näher erläutert.

Eine *lokale Strategie* zur Belegung von Ressourcen delegiert die Belegung und Ressourcen so weit wie möglich in Richtung der Aktionen im Handlungsbaum (Abbildung 5.11, links). Es wird also jeweils direkt vor der Verwendung einer Ressource deren Belegung, und direkt danach deren Freigabe durchgeführt. Dieser Ansatz ermöglicht maximale Flexibilität und eine starke Verwebung verschiedener FPs während der Ausführung. Allerdings entsteht dadurch auch das Risiko, dass Handlungen sich gegenseitig stören, indem zusammengehörende Handlungsteile getrennt werden. So darf während einer *Pick&Place*-Operation die Ressource `Hand` von keinem anderen FP verwendet werden, um zu verhindern, dass das transportierte Objekt fallengelassen wird.

Im Gegensatz dazu steht die *globale Strategie* (Abbildung 5.11 rechts). Dabei werden alle im FP verwendeten Ressourcen direkt bei Beginn der Handlung belegt und für die Dauer der Handlung behalten. Durch dieses Vorgehen werden Kollisionen mit anderen FPs ausgeschlossen. Allerdings ist dieser Ansatz sehr starr im Hinblick auf die parallele Ausführung mehrerer FPs; das Teilen von Ressourcen zwischen verschiedenen FPs wird dadurch nahezu unmöglich.

Zur Lösung wird ein *flexibler Ansatz* vorgeschlagen. Dabei werden Ressourcen abhängig von der Handlung lokal, global oder auf einer dazwischenliegenden Ebene belegt. So kann sichergestellt werden, dass einerseits bei einer *Pick&Place*-Operation die verwendeten Ressourcen durchgängig belegt bleiben, andererseits aber die Ressource der Sprachausgabe nicht unnötig lange belegt wird.

Die dazu nötige Wissensbasis assoziiert jeweils einen Knotentyp v mit den Ressourcen, die *für diese Teilhandlung global* zu belegen sind. Die jeweiligen Ressourcen werden dann für diese



(a) (b)

Abbildung 5.11: (a) Lokale Strategie zur Belegung von Ressourcen. Jede Ressource wird direkt vor bzw. nach der Verwendung belegt und freigegeben. (b) Globale Belegung von Ressourcen. Alle im FP verwendeten Ressourcen werden direkt bei Beginn der Handlung belegt, und erst am Ende der Handlung wieder freigegeben.

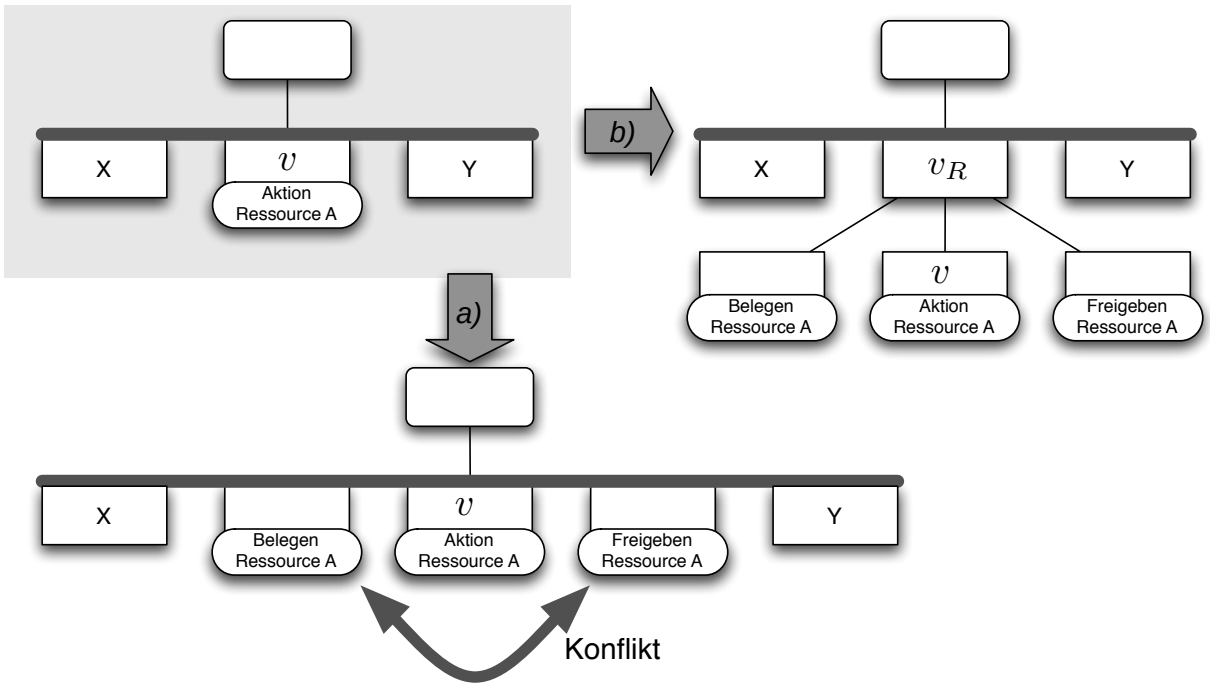


Abbildung 5.12: Einfügen der Ressourcenverwaltung für ein Blatt im Handlungsbaum. Oben links die Ausgangslage: Die Aktion v befindet sich in einer parallelen Gruppe mit anderen Aktionen. Wird nun die Ressourcenbelegung direkt davor und dahinter eingefügt, ergibt sich ein Konflikt. Dieser besteht darin, dass Belegung und Freigabe einerseits vor bzw. nach v aufgerufen werden müssen, andererseits sich aber in der gleichen parallelen Ausführungseinheit befinden (Fall a). Zur Lösung wird eine Hierarchieebene eingefügt (Fall b).

Teilhandlung global belegt, wenn sie nicht schon von einem hierarchisch höheren Knoten in der Handlung belegt wurden. Es können zwei Fälle auftreten:

- v ist ein Verzweigungsknoten. Die Aktionen zum Belegen und Freigeben werden als erstes und als letztes Kind von v in eigene Sitze eingefügt.
- v ist ein Blatt. In diesem Fall wird ein zusätzlicher Knoten v_R generiert, der den Sitz von v einnimmt und v als Kind besitzt. Die Hierarchie wird also erweitert. Die Aktionen zur Belegung und Freigabe der Ressourcen werden als global für v_R betrachtet und entsprechend Fall 1 eingefügt.

Die Notwendigkeit des in Fall 2 beschriebenen Vorgehens ergibt sich aus dem Fall, dass v Teil einer parallelen Gruppe ist, und ist in Abbildung 5.12 illustriert: Werden direkt vor und hinter v die Operationen zum Belegen und Freigeben hinzugefügt, so wird die Parallelität aufgebrochen. Die Operationen vor und hinter der parallelen Gruppe einzufügen, wäre nicht günstig: Die Ressourcen wären für die Dauer von $\mathrm{Max}\,(\mathit{Dauer}(X), \mathit{Dauer}(v), \mathit{Dauer}(Y))$ belegt. Deshalb wird das beschriebene Verfahren angewendet.

Das resultierende Flexible Programm $\mathcal{P}'$ (siehe Abbildung 5.4) beinhaltet sowohl alles aus der Vorführung extrahierte Handlungswissen als auch die während der Übersetzung hinzugefügten Handlungselemente. $\mathcal{P}'$ ist damit vollständig und sicher mit dem Roboter ausführbar.

5.7 Parallelisierung von Teilaufgaben

Der letzte Schritt des Übersetzungsprozesses beinhaltet die Parallelisierung von Teilaufgaben in $\mathcal{P}'$. Programmteile, die laut der Beschreibung im Makrooperator zu parallelisieren sind, wurden bereits während der Strukturübersetzung berücksichtigt (siehe Kapitel 5.3). Dieser letzte Schritt stellt also eine Optimierung dar, die hauptsächlich unnötige Totzeiten vermeidet. Diese können insbesondere bei durch Regelanwendung eingefügten Handlungsteilen auftreten.
Zur Bestimmung von parallelisierbaren Teilprogrammen werden a priori definierte Relationen verwendet. Diese sind entsprechend

$$A \Big\| \big(\{B, C, \ldots\} \setminus \{D, E, \ldots\}\big)$$

spezifiziert und können Platzhalter beinhalten. Verwendete Parallelisierungsregeln sind unter anderem

$$\begin{aligned}
\textit{(Sprachausgabe)} &\Big\| \big(\star \setminus \{\textit{(Belegung) (Freigabe) (Stop)}\}\big) \\
\textit{(Zustandsdarstellung)} &\Big\| \big(\star \setminus \{\textit{(Belegung) (Freigabe) (Stop)}\}\big) \\
\textit{(Hand schließen)} &\Big\| \big(\{\textit{(Arm bewegen) (Sprachausgabe) (Zustandsdarstellung)}\}\big).
\end{aligned}$$

Sprachausgaben und Zustandsdarstellungen sind unkritisch, da sie keine physischen Aktionen beinhalten. Sie sind also mit allen anderen parallelisierbar, abgesehen von Operationen

zur Belegung und Freigabe von Ressourcen. Genauso kann das Schließen des Robotergreifers mit der zugehörigen Armbewegung parallelisiert werden. Im Gegensatz zum Fall des Greifens eines Objekts ist es hier erlaubt, Hand und Arm gleichzeitig zu bewegen.

Nach der Parallelisierung erhält man das endgültige Flexible Programm $\mathcal{P}$ zum Makrooperator $\mathcal{O}$. Die Abbildung ist damit vollständig.

5.8 Aufwandsbetrachtung

Wie auch in [Knoop 07a] gezeigt wird, kann der Aufwand für das Abbildungsverfahren durch $O(n)$ dargestellt werden, wobei n die Anzahl der Knoten im Makrooperator darstellt. Dies ergibt sich aus der Betrachtung der Einzelschritte der Abbildung:
Während der Strukturabbildung wird jeder Knoten des ursprünglichen Makrooperators genau einmal verarbeitet. Die Verarbeitung eines solchen Knotens wird durch eine feste Anzahl von Schritten realisiert, so dass der Aufwand hier linear mit der Anzahl der Knoten ist. Die Abbildung der Elementaroperationen besteht aus der sequentiellen Verarbeitung jeder Elementaroperation, so dass dieser Schritt linearen Aufwand besitzt, abhängig von der Anzahl der Elementaroperationen.
Die Aufwandsbetrachtung der regelbasierten Modifikation gestaltet sich dahingehend schwieriger. Hierfür muss jeder Schritt und jede verwendete Regel separat betrachtet werden. Zunächst werden die auslösenden Muster gesucht. Dies geschieht für jede anzuwendende Regel und ist als Mustererkennung in $O(n)$. Die Anzahl der angewendeten Regeln ist fest und finit, so dass die gesamte regelbasierte Modifikation in $O(n)$ liegt, vorrausgesetzt der Aufwand jeder Operation der Regeln ist fest. Dies ist für die in Kapitel 5.5.2 aufgeführten Regeln der Fall, muss allerdings für jede weiter hinzugefügte Regel überprüft werden.
Die übrigen Schritte der Ressourcenverwaltung und Parallelisierung stellen jeweils einen linearen Durchlauf des Handlungsbaums dar und besitzen somit auch linearen Aufwand mit der Anzahl der Knoten.

5.9 Verwendetes Hintergrundwissen

Der Abbildungsprozess besteht aus fünf Schritten, die jeweils Wissen aus dem Makrooperator und Wissen über das Zielsystem in die ausführbare Handlungsbeschreibung integrieren. Dabei ist die verwendete Information über das Zielsystem von besonderer Bedeutung. Basierend darauf lässt sich die Abbildung für abstraktes Handlungswissen definieren und ausführen.
In diesem Kapitel wird deshalb das für die Abbildung notwendige Hintergrundwissen über den Roboter untersucht und zusammengefasst. Implizit sind diese Eigenschaften in jeder symbolischen Handlungsbeschreibung für autonome Serviceroboter enthalten, werden aber nicht explizit definiert. Dies stellt allerdings eine Grundvoraussetzung dafür dar, dass Handlungswissen auf verschiedene Roboter übertragen werden kann.
Das während der Abbildung verwendete Wissen über den Roboter besteht im Einzelnen aus den folgenden Komponenten:

- Die *Struktur der Handlungsbeschreibung* für das ausführende System muss bekannt sein. Diese Struktur kann hierarchisch, sequentiell, oder in Form eines endlichen Au-

tomaten aufgebaut sein. Weitere Beispiele sind Petri-Netze und Vorranggraphen. Für die Strukturen des Makrooperators muss jeweils eine Abbildungsstrategie existieren; dies bedeutet auch, dass die Zielbeschreibung mindestens die gleiche Mächtigkeit wie der Makrooperator besitzen muss.

- Die *Zuordnung von Elementaroperatoren* zu Roboterprogrammen sowie die Abbildung der Parameter zwischen den verschiedenen Arbeitsräumen muss existieren.

  Dies ist ein momentan äußerst aktives Forschungsgebiet, und Ansätze des Imitationslernens beschäftigen sich mit diesem Problem. Es existieren verschiedene Lösungen zur Abbildung einzelner Aktionen; eine detaillierte Betrachtung findet sich in Kapitel 2.4.

- Die Menge der *Manipulationsregeln* ist der wichtigste Bestandteil des Übersetzungsprozesses. Die Regeln vervollständigen das Handlungswissen für die Ausführung. Es gibt zwei verschiedene Klassen von Regeln: Notwendige und optionale Regeln. Abhängigkeiten zwischen den Regeln müssen modelliert werden, um die Anwendungsreihenfolge zu bestimmen.

- Die *Belegung von Ressourcen* kann sicherheitsorientiert oder bezüglich größtmöglicher Flexibilität in der Ausführung geschehen. Für einzelne Teilhandlungstypen ist deshalb spezifiziert, welche Ressourcen jeweils für die Dauer der Handlung zu belegen sind.

- Der letzte Abbildungsschritt stützt sich auf Vorgaben zur Parallelisierung. Diese sind ebenfalls a priori gegeben, und können für unterschiedliche Robotersysteme unterschiedliche Ausprägungen besitzen. Dies hängt beispielsweise auch von der Zahl der Manipulatoren des Roboters ab: Je mehr unabhängige Manipulatoren dieser besitzt, umso mehr Handhabungen können parallelisiert werden.

Wird diese Information über ein Robotersystem explizit definiert, stellt dies einen wichtigen Schritt für die automatische Abbildung von abstraktem Handlungswissen dar.

5.10 Zusammenfassung

Die Abbildung von abstraktem Handlungswissen auf dedizierte Robotersysteme kann durch einen Prozess mit den fünf Schritten *Strukturabbildung*, *Abbildung der Elementaroperatoren*, *Regelbasierte Modifikation*, *Einfügen der Ressourcenverwaltung* und *Parallelisierung* geschehen. Dabei werden das Handlungswissen und die Parameter der abstrakten Quellbeschreibung verwendet, sowie a priori zu definierendes Wissen über das Zielsystem. Wird dieses Wissen explizit für einen Roboter und die entsprechende Ausführungsbeschreibung definiert, kann die Abbildung für Handlungswissen auf den Roboter damit beschrieben werden.
Die Abbildung wird am Beispiel von Makrooperatoren und Flexiblen Programmen gezeigt. Die Struktur der Beschreibung, Bedingungen, sowie im Makrooperator enthaltene Parameter werden für die Ausführung abgebildet. Verschiedene Regeln zur Vervollständigung der Handlung werden vorgestellt.

Kapitel 6

Dynamische Ausführung Flexibler Roboterprogramme

Die Modellierung von Handlungswissen in Form von Flexiblen Programmen wurde in Kapitel 4.2 vorgestellt. Dieses Kapitel beschreibt die zur Ausführung dieses Handlungswissens notwendigen Komponenten und Vorgänge. Dazu gehört insbesondere die *Handlungsauswahl*, also die Aktivierung Flexibler Programme abhängig von Kontext und Eingaben. Desweiteren werden die zur Ausführung Flexibler Programme notwendigen Prozesse beschrieben. Dies umfasst die dynamische Handhabung von Handlungsbäumen, die Überprüfung von Vor- und Nachbedingungen sowie die Darstellung und Visualisierung von Flexiblen Programmen.

6.1 Auswahl von Handlungen

Ein Flexibles Programm stellt zunächst die Fähigkeit dar, eine bestimmte Handlung auszuführen (eine *Verhaltensdisposition*). Diese Verhaltensdispositionen können aktiviert und deaktiviert werden; eine aktivierte Verhaltensdisposition stellt die Möglichkeit dar, die zugehörige Handlung auszuführen (siehe auch [Au 06]).

Die Menge der aktivierten Verhaltensdispositionen stellt also die Menge der zu diesem Zeitpunkt verfügbaren Handlungen dar. Eine verfügbare Handlung wird durch entsprechende Nachrichten angestoßen. Dies sind üblicherweise externe *Ereignisse* (engl.: *Events*) wie z.B. Kommandos des Benutzers, und werden im Rahmen der Ausführungsarchitektur (siehe Kapitel 4.1.2) von den Sensorkomponenten in das System eingebracht. Typische Ereignisse, die Handlungen anstoßen, sind Spracheingaben oder andere direkte Benutzereingaben.

Die Aufgabe der Handlungsauswahl besteht nun darin, Flexible Programme zu aktivieren, zu deaktivieren, und zu priorisieren (siehe Abbildung 4.2). Die Priorität eines Flexiblen Programms entscheidet insbesondere bei der Nachrichtenverteilung über die Adressierung der Nachrichten.

Um die notwendige Menge der aktivierten FPs zu bestimmen, wird der momentane Zustand des Roboters und der Umwelt betrachtet. Die verwendete Definition und Bestimmung dieses *Kontexts* wird in Kapitel 6.1.1 erläutert. Die darauf basierende Auswahl von Handlungen ist in Kapitel 6.1.2 dargestellt.

6.1.1 Bestimmung des Kontexts

Der Kontext beschreibt im Allgemeinen Faktoren der Umwelt und des Agenten selbst, die die Entscheidungen des Agenten beeinflussen können. In [Dey 01] wird der Kontext folgendermaßen definiert:

Definition 6.1 Der Kontext wird beschrieben durch sämtliche Information, die zur Charakterisierung der Situation einer Entität verwendet werden kann. Eine Entität ist eine Person, ein Ort, oder ein Objekt, das als relevant für die Interaktion zwischen einem Benutzer und einer Applikation gelten kann. Dies beinhaltet den Benutzer und die Applikation selbst.[1]

Diese Definition findet im Weiteren Verwendung. Dabei wird der betrachtete Kontext eingeschränkt auf wenige beobachtbare Zustände des Roboters und der Umgebung, die zur Handlungsauswahl herangezogen werden.

Interne Zustände des Roboters

Die Menge der für die Handlungsauswahl relevanten Zustände des Roboters wird der Verwaltung der Flexiblen Programme sowie der Nachrichtenverteilung entnommen. Die relevanten internen Zustände werden auf die folgende Menge eingegrenzt:

- Die Menge der *aktivierten* Flexiblen Programme stellt die Menge der momentanen Verhaltensdispositionen dar. Diese wird von der Handlungsauswahl überwacht und gesteuert.
- Alle aktivierten Flexiblen Programme besitzen eine *Priorität*, die von der Handlungsauswahl vergeben und geändert werden kann.
- Aktivierte Flexible Programme werden durch entsprechende Ereignisse angestoßen, die Handlung also ausgelöst. Jedes aktivierte FP ist entweder *bereit* oder *laufend*.
- Die *Historie* der Verarbeitung wird aufgezeichnet. Basierend darauf wird die Priorisierung und damit die Zuteilung von Nachrichten gesteuert.

Zustände der Umgebung

Die Menge der externen Zustände, die zur Handlungsauswahl betrachtet werden, kann beliebig aus allen beobachtbaren Parametern gewählt werden. Für die vorliegende Arbeit werden drei Parameter ausgewählt, die zur Bestimmung des Kontexts dienen:

- Der *aktuelle Aufenthaltsort* des Roboters ist ein wesentlicher Faktor für die Auswahl von Verhalten. Viele Verhalten sind ortsgebunden; so soll sich ein Serviceroboter eventuell in der Küche anders verhalten als im Wohnzimmer.
- *Datum und Uhrzeit* können dazu genutzt werden, bestimmte Verhalten zu festgelegten Tageszeiten zu aktivieren, oder an festgelegten Wochentagen.

[1] Englische Originaldefinition nach [Dey 01]: Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

```
   Eingabe : R̄ mit r = {N, S} und S = s_0 ... s_n (Menge aller Handlungsrahmen),
   K (aktueller Kontext)
   Ausgabe : R̄* (Menge der gültigen Handlungsrahmen)
 1 begin
 2     R̄* ← ∅
 3     foreach r ∈ R̄ do
 4         g ←true
 5         foreach s ∈ S(r) do
 6             if s ∉ K then
 7                 g ←false
 8                 break                                  /* r nicht gültig */
 9             endif
10             else
11                 continue                                   /* s_i gültig */
12             endif
13         endfch
14         if g =true then
15             R̄* ← r                  /* Gültigen Handlungsrahmen hinzufügen */
16         endif
17     endfch
18     return R̄*
19 end
```

Algorithmus 6.1 : Bestimmung der gültigen Handlungsrahmen unter Verwendung des Kontexts. `gültigeHandlungsrahmen`

- Die Menge der *anwesenden Personen* und deren Namen, soweit diese identifiziert werden können, bestimmt die Auswahl der FPs. Auf diese Weise können unterschiedliche Personen unterschiedlich behandelt werden.

Mit Hilfe dieser Zustände wird ein *Handlungsrahmen* als Basis für die Handlungsauswahl definiert:

Definition 6.2 Ein Handlungsrahmen legt eine Menge an Situationen fest. Er besteht aus einem eindeutigen Bezeichner $\mathcal{N}$ und einer Situationsbeschreibung $\mathcal{S}$. Die Situationsbeschreibung $\mathcal{S}$ wird durch $n \geq 0$ Zustände der Umgebung beschrieben. Ein Handlungsrahmen wird dann als *gültig* bezeichnet, wenn die angegebenen Parameter eine Untermenge des momentan bekannten Kontexts darstellen.

Ein Handlungsrahmen beschreibt also eine Menge von Situationen. Tabelle 6.1 zeigt zwei exemplarische Handlungsrahmen. Es ist auch möglich, dass sich mehrere Handlungsrahmen überschneiden. Die Menge der *gültigen* Handlungsrahmen wird mit Algorithmus 6.1 bestimmt.
Mit Hilfe der Handlungsrahmen können also Faktoren des Kontexts gruppiert und gebündelt werden, um eine Menge an Situationen zu beschreiben. Dieses Konzept wird nun zur Auswahl von Handlungen genutzt.

Abends mit Bernd	
Datum, Uhrzeit	24. Dezember 20:16
Personen	Bernd

(a)

Frühstück mit Hella	
Ort	Küche
Datum, Uhrzeit	21. Februar 7:58
Personen	Hella

(b)

Abbildung 6.1: Beispiel zweier Handlungsrahmen. Diese können zur Parametrierung der Handlungsauswahl verwendet werden. Für *Abends mit Bernd* (a) können beispielsweise FPs zum Getränke Anbieten und Holen aktiviert werden, während bei *Frühstück mit Hella* (b) andere Flexible Programme aktiviert sind.

6.1.2 Kontextabhängige Handlungsauswahl

Die Aufgabe der Handlungsauswahl besteht zum einen darin, die Menge der Flexiblen Programme kontextabhängig zu aktivieren und zu deaktivieren, zum anderen in der Priorisierung der aktivierten FPs.

Aktivierung und Deaktivierung von Flexiblen Programmen

Abhängig von der Situation sollen Flexible Programme aktiviert werden können. Um die Menge der zu aktivierenden FPs bestimmen zu können, werden alle existierenden Roboterprogramme zunächst folgendermaßen klassifiziert:

- *Obligatorische FPs* sind Flexible Programme, die jederzeit im System verfügbar sein müssen. Dies beinhaltet beispielsweise Sicherheitsaspekte, um jederzeit auf Ausnahmesituationen reagieren zu können.

- *Situationsspezifische FPs* können abhängig vom Kontext aktiviert werden. Diese FPs können mit einer Menge von $n \geq 0$ Handlungsrahmen assoziiert sein.

Damit kann festgelegt werden, welche Flexiblen Programme zu einem Zeitpunkt aktiviert sein müssen. Diese Menge ist abhängig von der Menge der gültigen Handlungsrahmen und wird als *konsistente Menge der FPs* bezeichnet. Sie beinhaltet alle FPs, für die gilt:

- Die Handlung des FPs wird gerade ausgeführt,

- das FP ist mit einem *gültigen Handlungsrahmen* assoziiert,

- oder das FP ist in der Menge der *obligatorischen FPs* enthalten.

Die Konsistenz der aktivierten Flexiblen Programme mit dem Kontext muss zu verschiedenen Zeitpunkten überprüft werden. Dies geschieht bei der Initialisierung des Systems, sobald die Handlung eines FPs abgearbeitet ist, und bei einer Änderung des Kontexts. Damit sind alle Fälle abgedeckt, bei denen sich die Menge der FPs ändern kann. Algorithmus 6.2 gibt die Schritte zur Aktivierung und Deaktivierung der FPs wieder.

```
   Eingabe : R̄ (Menge aller Handlungsrahmen), K (aktueller Kontext), F̄ mit
             f = {r̂, P̄} (Assoziation von Handlungsrahmen r̂ mit einer Menge FPs P̄),
             Ā (Menge aktivierter FPs), L̄ (Menge laufender FPs)
   Ausgabe : -
 1 begin
 2     R̄* ← gültigeHandlungsrahmen                          /* Algorithmus 6.1 */
 3     foreach r ∈ R̄ do
 4         foreach f ∈ F̄ do
 5             if r = r̂ then
 6                 foreach p ∈ P̄ do
 7                     if r ∈ R̄* then                              /* r ist gültig */
 8                         if p ∉ Ā then
 9                             activate(p)              /* Programm p wird aktiviert */
10                         endif
11                     endif
12                     else
13                         if p ∉ L̄ and p ∈ Ā then
                                                /* p ist nicht gültig und läuft nicht */
14                             deactivate(p)          /* Programm p wird deaktiviert */
15                         endif
16                     endif
17                 endfch
18             endif
19         endfch
20     endfch
21 end
```

Algorithmus 6.2 : Aktivierung und Deaktivierung von Flexiblen Programmen

Priorisierung von Flexiblen Programmen

Die Auswirkung der Priorisierung Flexibler Programme ist in Abbildung 6.2 dargestellt. Flexiblen Programmen werden asynchrone, unadressierte Nachrichten der Reihe nach angeboten. Die Reihenfolge wird dabei durch die Priorität bestimmt. FPs mit hoher Priorität wird also jedes Ereignis zur Verarbeitung angeboten, während eine niedrige Priorität nur Ereignisse erhält, die nicht durch höher priorisierte FPs akzeptiert wurden.

Für den vorliegenden Fall sind vier Prioritätsklassen definiert, denen jeweils ein Wert zugeordnet wird. Die höchste Prioritätsklasse mit dem Wert 5 ist Flexiblen Programmen mit Sicherheitsfunktionen vorbehalten. Aktive und laufende Flexible Programme erhalten die Prioritätsklasse *aktiv*, die mit den Werten $2-4$ weiter unterteilt ist. Aktive, nicht laufende FPs besitzen die Bereitschaftpriorität 1, während FPs mit der Ausschlusspriorität 0 bei der Nachrichtenvergabe nicht berücksichtigt werden. Abbildung 6.3 visualisiert die Anordnung der Prioritätsklassen für Flexible Programme.

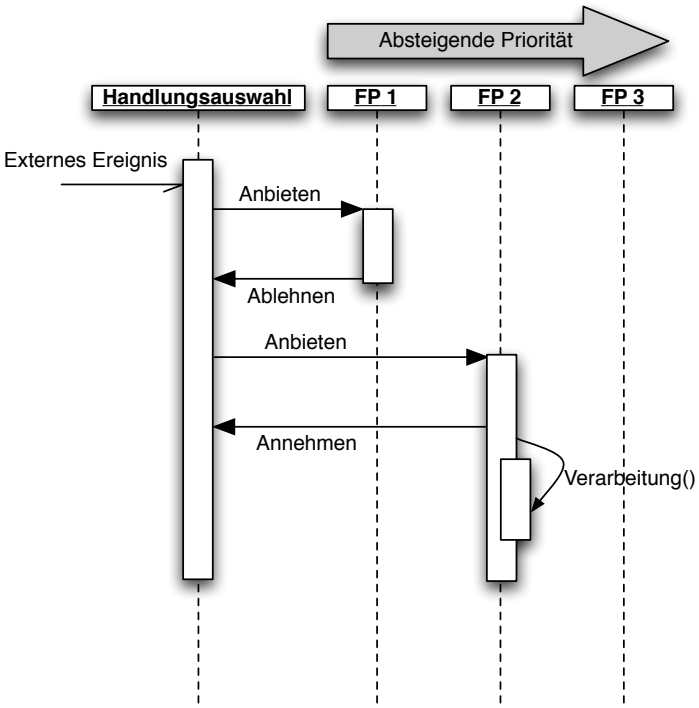


Abbildung 6.2: Auslösen einer aktivierten Handlung durch ein externes Ereignis. Asynchrone, externe Ereignisse werden den aktivierten FPs in absteigender Priorität angeboten. Einmal akzeptierte Ereignisse werden nicht weiter verteilt. FP 3 wird die Nachricht deshalb aufgrund der niedrigeren Priorität nicht angeboten.

Die Priorität für jedes FP wird vor der Vergabe einer Nachricht neu bestimmt. Die dazu verwendeten Parameter sind der Typ der zu verteilenden Nachricht, der momentane Zustand der FPs, sowie die Verarbeitungshistorie vergangener Nachrichten.
Algorithmus 6.3 gibt die Schritte zur Bestimmung der Prioritäten für Flexible Programme wieder. Dazu werden zuerst die gültigen Handlungsrahmen mit Algorithmus 6.1 bestimmt; diese werden dann in Bezug auf deaktivierte, aktivierte und laufende Flexible Programme ausgewertet. Nicht laufende, laut Zuordnung zu den Handlungsrahmen nicht zu aktivierende FPs werden gegebenenfalls deaktiviert; umgekehrt werden FPs zusätzlich aktiviert, wenn der momentane Kontext dies verlangt.

6.2 Verarbeitung von Flexiblen Programmen

Die hierarchische Modellierung von Handlungswissen ist bereits in Kapitel 4.2 vorgestellt worden. Die folgenden Abschnitte beschreiben die Abläufe während der Interpretation und Ausführung Flexibler Programme. Insbesondere wird die Abarbeitung des Handlungsbaums dargestellt, sowie die Handhabung der einzelnen Knoten innerhalb einer vollständigen Handlungsbeschreibung.
Zwei Schichten definieren die Verarbeitung Flexibler Programme: Die Baumverarbeitung operiert auf dem Handlungsbaum entlang der Handlungsmodellierung mit sequentiellen und parallelen Handlungsteilen. Die von diesem Kern jeweils bestimmte Handlungsfolge wird

Eingabe : $\bar{A}$ (Menge aktivierter FPs), $\bar{L}$ (Menge laufender FPs), E (aktuelle Nachricht), H mit $h_i = \{a_h, e_h, t_h\}$ (Verarbeitungshistorie, bestehend aus Ereigniszuteilungen e_h an FP a_h zur Zeit t_h)
Ausgabe : $\bar{A}^\star$ (Liste nach Priorität geordneter FPs)

```
1  begin
2      Ā* ← ∅, Ā_0 ← ∅, Ā_1 ← ∅, Ā_2 ← ∅, Ā_3 ← ∅
3      a_t ← 0, Δ_t ← 0
4      foreach a ∈ Ā do
5          if Typ(E) von a nicht verarbeitbar then
6              Ā_0 ← a                                   /* Priorität 0 für FP */
7              continue
8          endif
9          if a ∉ L̄ then
10             Ā_1 ← a                          /* Priorität 1 für FP (Bereitschaft) */
11         endif
12         else
13             Ā_2 ← a                               /* Priorität 2 für FP (laufend) */
14         endif
15     endfch
           /* Suche das FP in Ā_2 mit der längsten Wartezeit auf Typ(E) */
16     foreach a ∈ Ā_2 do
17         foreach h ∈ H̄ do
18             if a_h = a and e_h = E and (t_now − t_h > Δ_t) then
19                 Δ_t ← (t_now − t_h)
20                 a_t ← a
21             endif
22         endfch
23     endfch
24     foreach a ∈ Ā_2 do
25         if a = a_t then
26             Ā_3 ← a
27             Ā_2 ← Ā_2 \ a
28         endif
29     endfch
30     Ā* ← Ā_0 ← Ā_1 ← Ā_2 ← Ā_3                        /* Aufbau von Ā* */
31     return Ā*
32 end
```

Algorithmus 6.3 : Sortierung der FP-Liste nach Prioritäten. Die Priorität jedes FPs bezüglich einer neuen Nachricht ist abhängig vom eigenen Zustand, der Nachricht selbst, und der Verarbeitungshistorie.

Abbildung 6.3: Prioritäten für Flexible Programme. Programme mit höherer Nummer haben Vorrang bei der Verarbeitung neuer Nachrichten

von der Kommunikationsschicht an die entsprechenden Komponenten übermittelt. Diese empfängt auch die Antworten der Komponenten und gibt sie an den Kern weiter. Dieser Zusammenhang ist in Abbildung 6.4 dargestellt.

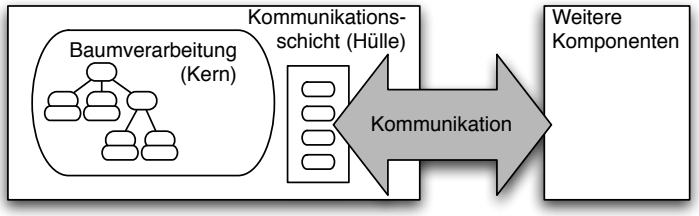


Abbildung 6.4: Baumverarbeitung und Kommunikationsschicht zur Verarbeitung Flexibler Programme. Der Kern operiert auf der hierarchischen Handlungsrepräsentation, während die Hülle die Kommunikation mit den ausführenden Komponenten übernimmt.

Das Verhalten der Kommunikationsschicht wird durch die Definition der umgebenden Architektur und Kommunikationskanäle festgelegt (siehe Kapitel 4.1). Die eigentliche Verarbeitung Flexibler Programme findet im Kern statt.
Mit dieser Grundlage kann der Ablauf während der Verarbeitung Flexibler Programme dargestellt werden. Einzelne Knoten, die gleichzeitig unabhängige Flexible Programme darstellen, besitzen immer einen der drei Zustände *virtuell*, *instantiiert* (oder besucht) oder *abgeschlossen*. Ist der Zustand virtuell, ist der Prospekt noch nicht ausgewertet worden; im instantiierten Zustand wurde der Prospekt ausgewertet und die Kandidatenauswahl getroffen, und ein abgeschlossener Knoten ist vollständig abgearbeitet, alle Kindknoten also wiederum abgeschlossen und damit alle Aktionen abgeschlossen. Abbildung 6.5 veranschaulicht dies.
Die Verarbeitung des Baums wird mit jedem Ereignis e angestoßen, das die Kommunikationsschicht von den ausführenden Komponenten (Agenten) erhält. Ein Verarbeitungsschritt bestimmt alle als nächstes auszuführenden Blätter. Dies geschieht in zwei Schritten:

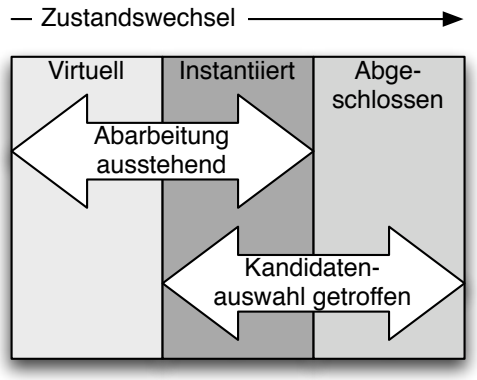


Abbildung 6.5: Zustände eines Flexiblen Programms

- Zunächst wird, ausgehend von dem Blatt, das e ausgelöst hat (und damit von e als abgeschlossen bestätigt wird), der Baum aufwärts nach der letzten nicht abgeschlossenen Teilhandlung durchsucht. Bei diesem Prozess, *Auftauchen* genannt, werden alle Teilbäume, deren Kinder als abgeschlossen markiert sind, ebenfalls so gekennzeichnet. Zwei Fälle führen zum Abbruch des Auftauchens: Entweder wird die Wurzel der Handlung erreicht, die Gesamthandlung ist also abgeschlossen; oder es wird eine Ebene erreicht, deren Kindknoten nicht alle als abgeschlossen markiert sind (siehe Algorithmus 6.4).

- Die Bestimmung der auszuführenden Blätter geschieht während des *Abtauchens*. Ausgehend von der während des Auftauchens gefundenen Ebene werden alle Elementaraktionen gesucht, die zu diesem Zeitpunkt ausführbar sind. Dafür werden gegebenenfalls Knoten instantiiert, und es wird zwischen verschiedenen Kandidaten gewählt. Alle auszuführenden Blätter werden zusammengefasst der Hülle übergeben.

Wenn während des Abtauchens ein Knoten den Zustand von *virtuell* nach *instantiiert* wechselt, wird der Prospekt ausgewertet und gegebenenfalls zwischen mehreren alternativen Handlungen gewählt (vgl. Algorithmus 6.5). Dies geschieht anhand der Vorbedingungen und der Bewertungsfunktion, die in Kapitel 4.2.3 definiert wurden.

6.3 Auswahl von Kandidaten

Jeder Kandidat für einen Knoten im Baum ist für sich genommen ein vollständiges Programm. Bei der Auswahl von Kandidaten wird die auszuführende Handlung also grundsätzlich aus vollständigen Programmen ausgewählt.

Diese Auswahl erfolgt nach Kapitel 4.2.3 zweistufig. Der Prospekt eines Knotens enthält dazu zwei Kriterien: Die *Vorbedingungen* und die *Bewertungsfunktion*. Die notwendigen Vorbedingungen stellen eine binäre Auswahl dar; es wird also für jeden Kandidaten entschieden, ob er prinzipiell anwendbar ist. Im Gegensatz dazu stellt die Bewertungsfunktion ein weiches, kontinuierliches Kriterium dar, das ausgewertet wird, wenn mehrere Kandidaten anhand der Vorbedingungen in Frage kommen.

Eingabe : $\mathcal{P}$ (Flexibles Programm, das sich in der Ausführung befindet), p (Knoten aus $\mathcal{P}$, der mit dem letzten Ereignis als *abgeschlossen* markiert werden kann)
Ausgabe : q (Knoten, der jetzt ausführbare Teilhandlung(en) besitzt)

```
1  begin
2      if p ist Blatt then
3          p als abgeschlossen markieren
4      endif
5      if p kein Blatt und alle Kinder von p abgeschlossen then
6          p als abgeschlossen markieren                    /* Teilhandlung fertig */
7      endif
8      if p abgeschlossen then
9          if alle Mitglieder der parallelen Gruppe von p abgeschlossen then
10             if p = Wurzel then
11                 return 0                                  /* Handlung P fertig */
12             endif
13             else
14                 p ← Vater(p)
15                 Auftauchen(P, p)
16             endif
17         endif
18         else
19             return 0                          /* auf weitere Ereignisse warten */
20         endif
21     endif
22     else
23         return p
24     endif
25 end
```

Algorithmus 6.4 : Auftauchen aus dem Handlungsbaum, um den nächsten Knoten zu finden, dessen Kinder bearbeitet werden müssen: `Auftauchen`

```
   Eingabe : P (Flexibles Programm, das sich in der Ausführung befindet), p (Knoten
             aus P, der nicht abgeschlossene Kindknoten besitzt)
   Ausgabe : P (Liste mit abzuarbeitenden Blättern)
 1 begin
 2     if erstes nicht abgeschlossenes Kind von p ist unbesucht then
 3         if Kinder von p sind virtuell then
 4             instantiiere Kinder(p)                  /* Auswahl aus den Kandidaten */
 5         endif
 6         p ← erstes instantiiertes Kind(p)
 7         Abtauchen(P, p)                                  /* rekursiver Aufruf */
 8     endif
 9     else if p ist Blatt then
10         P ← p                                    /* Hinzufügen des Blatts zu P */
11     endif
12     if rechter paralleler Geschwisterknoten von p existiert then
13         p ← rechter paralleler Geschwisterknoten von p
14     endif
15     else
16         if p = Wurzel then
17             return P                      /* Baum durchlaufen, Auswahl beendet */
18         endif
19         p ← Vater(p)
20         Abtauchen(P, p)                                  /* rekursiver Aufruf */
21     endif
22 end
```

Algorithmus 6.5 : Abtauchen in den Handlungsbaum und Bestimmung aller auszuführenden Blätter (Elementaraktionen): `Abtauchen`

6.3.1 Überprüfung von Bedingungen

Die Modellierung der Bedingungsterme wurde bereits in Kapitel 4.3.2 vorgestellt. Die so modellierten Bedingungen werden zur Laufzeit ausgewertet, um die Anwendbarkeit von Teilprogrammen zu prüfen. Dabei wird WAHR als 1 und FALSCH als 0 kodiert. Die Funktion $\text{val}(T)$ bezeichnet den Wahrheitswert eines Terms T. Die Auswertung der modellierten Operatoren aus Tabelle 4.2 wird im Folgenden vorgestellt.

- Negation:
$$\text{val}(\neg T) = 1 - \text{val}(T)$$

- Konjunktion:
$$\text{val}(T_1 \wedge \cdots \wedge T_n) = \prod_{i=1}^{n} \text{val}(T_i)$$

- Disjunktion:
$$\text{val}(T_1 \vee \cdots \vee T_n) = \max_{i=1}^{n}(\text{val}(T_i))$$

- Allquantifikation:
$$\text{val}(\forall x \in \mathbb{M} : T(x)) = \text{val}(T(x_1) \wedge \cdots \wedge T(x_n)) \qquad \text{mit } \mathbb{M} = \{x_1, \ldots, x_n\}$$

- Existenquantifikation:
$$\text{val}(\exists x \in \mathbb{M} : T(x)) = \text{val}(T(x_1) \vee \cdots \vee T(x_n)) \qquad \text{mit } \mathbb{M} = \{x_1, \ldots, x_n\}$$

Die Quantoren $\forall$ und $\exists$ setzen dabei voraus, dass es bei $\mathbb{M}$ um eine endliche Menge handelt. Da dies bei dem gegebenen Umweltmodell der Fall ist, kann diese Annahme getroffen werden. Allerdings kann die Auswertung quantifizierter Terme viel Zeit und Ressourcen in Anspruch nehmen, da für die Auswertung jeder Quantor durch die in Frage kommende Menge an Relationen ersetzt werden muss. Dies sollte bei der Wahl von Bedingungstermen berücksichtigt werden. Das Verfahren zur Prüfung von Bedingungen ist in Algorithmus 6.6 dargestellt.

6.3.2 Auswahl anhand der Bewertungsfunktion

Die Bewertungsfunktion ist analog zur Vorbedingung aufgebaut. Dabei können alle Relationen verwendet werden, die im Umweltmodell bekannt sind. Durch die Auswertung mit den Werten WAHR, FALSCH und `unbekannt` können sich beliebige Werte zwischen 0 und 1 ergeben. So wird zwischen verschiedenen Kandidaten derjenige mit dem höchsten Bewertungsmaß gewählt.
In die Menge der Relationen können zusätzlich weitere Funktionen aufgenommen werden, die zur Bewertung der Situation dienen können. Diese können dann zur Auswahl zwischen verschiedenen Kandidaten herangezogen werden.

```
   Eingabe : 𝒯 (vollständiger Bedingungsterm), t (zu prüfender Knoten in 𝒯)
   Ausgabe : w (Wahrheitsmaß für den gegebenen Bedingungsterm t)
 1 begin
 2     if k = ∧ then                                          /* 'und' Bedingung */
 3         v = 1
 4         foreach Kind u von t in 𝒯 do
 5             if Bedingungsprüfung(𝒯, u) = 0 then return 0
 6             v = v·Bedingungsprüfung(𝒯, u)
 7         endfch
 8         return v
 9     endif
10     else if k = ∨ then                                     /* 'oder' Bedingung */
11         v = 0
12         foreach Kind u von t in 𝒯 do
13             if Bedingungsprüfung(𝒯, u) = 1 then return 1
14             else if Bedingungsprüfung(𝒯, u) > v then
15                 v =Bedingungsprüfung(𝒯, u)
16             endif
17         endfch
18         return v
19     endif
20     else if k = ¬ then                                     /* 'nicht' Bedingung */
21         return 1−Bedingungsprüfung(𝒯, Kind(k))
22     endif
23     else if k = ∀ then                                     /* 'für alle' Bedingung */
24         Erzeuge neuen Knoten k* vom Typ ∧
25         foreach Werte a vom Typ Kind(k) do
26             Ersetze Variable in Kind(k) durch a, füge Kind(k) als Kind zu k* hinzu
27         endfch
28         return Bedingungsprüfung(𝒯, k*)
29     endif
30     else if k = ∃ then                                     /* 'existiert' Bedingung */
31         Erzeuge neuen Knoten k* vom Typ ∨
32         foreach Werte a vom Typ Kind(k) do
33             Ersetze Variable in Kind(k) durch a, füge Kind(k) als Kind zu k* hinzu
34         endfch
35         return Bedingungsprüfung(𝒯, k*)
36     endif
37     else if Atomarer Prüfer f für k existiert then return f(k)
38     ;                                                      /* elementare Bedingung */
39     else return 0.5
40     ;                                                      /* unbekanntes Ergebnis */
41 end
```

Algorithmus 6.6 : Rekursive Überprüfung der Bedingungsterme: `Bedingungsprüfung`

6.4 Asynchrone Teilhandlungen

Ist eine Teilhandlung abgearbeitet, die Wurzel der Teilhandlung also als *abgeschlossen* markiert, so wird im Handlungsbaum der nach den Abarbeitungsregeln nächste Schritt bearbeitet. Damit können im Ablauf modellierte Handlungen ausgeführt werden. Asynchrone Ereignisse, die optional sind und nicht aus der Handlung direkt resultieren, sind jedoch so nicht verarbeitbar. Um diese Ereignisse in der Handlungsmodellierung berücksichtigen zu können, besteht die Möglichkeit, Teilhandlungen als *asynchron* zu markieren.
Eine als asynchron markierte Teilhandlung wird bei der Verarbeitung des Handlungsbaums nicht als mandatorisch gesehen. Der asynchrone Handlungszweig wird betreten, es wird jedoch nicht die Fertigstellung abgewartet (siehe Abbildung 6.6).

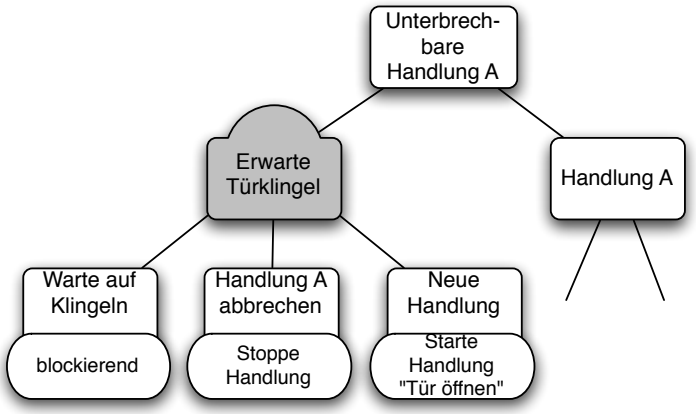


Abbildung 6.6: Asynchrone Verarbeitung eines Teilbaums. Die grau markierte Teilhandlung ist als asynchron markiert. Damit wird sie betreten, aber die Fertigstellung nicht abgewartet. So kann auf optional auftretende Ereignisse reagiert werden, ohne dass der Handlungsbaum blockiert wird.

Dies unterscheidet sich wesentlich von der *parallelen* Ausführung zweier Teilhandlungen. Bei der parallelen Ausführung wird der nächsthöhere Knoten erst dann als *fertig* markiert, wenn alle parallelen Zweige abgearbeitet sind. Eine komplette Handlung, bei der Teile nicht als *fertig* markiert sind, kann dann nicht insgesamt als fertig angesehen werden. Dieses wird durch die *asynchrone* Behandlung von Teilhandlungen ermöglicht.

6.5 Graphische Darstellung der Roboterprogramme

Eine der Stärken hierarchischer Handlungsbeschreibungen im Gegensatz zu linearen Beschreibungen (beispielsweise eines klassischen KI-Planers) besteht in der Übersichtlichkeit und Lesbarkeit der Programme. Diese Lesbarkeit kann weiter unterstützt werden durch eine geeignete Darstellung der Programme, wie sie beispielsweise auch für PRS (siehe [Ingrand 96]) vorgestellt wird.
Zur Darstellung des Ablaufs Flexibler Programme sowie zum direkten Editieren wurde deshalb eine graphische Benutzerschnittstelle geschaffen. Die Funktionalität der Darstellung zur Laufzeit unterscheidet sich dabei von der Editorfunktion:

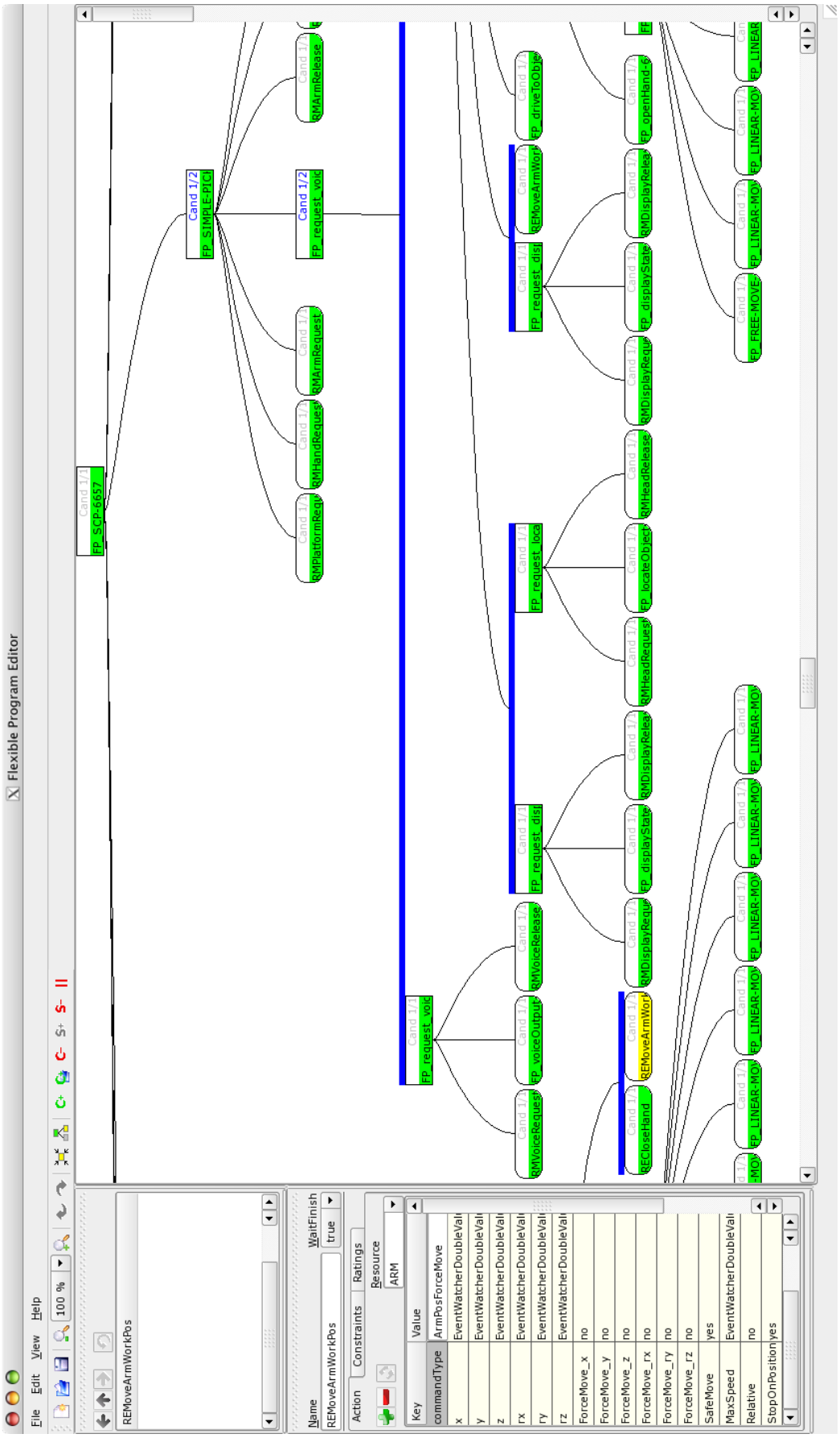


Abbildung 6.7: Vollständiger Editor für Flexible Programme. Es können sowohl existierende Programme verändert als auch neue geschaffen werden. Die Wurzel der Handlung befindet sich oben, die Sitze und Kandidaten werden unter jedem Knoten angeordnet. Innere Knoten werden eckig, Blätter rund angezeigt. Blaue Querbalken zeigen parallele Ausführungsgruppen an. Auf der linken Seite werden oben der Prospekt und unten die Aktion des markierten Knotens detailliert. Bei der Darstellung zur Laufzeit werden die Editierfunktionen ausgeblendet, und es wird der aktuell existierende Handlungsbaum im Hauptfenster visualisiert.

- Zum Editieren eines Flexiblen Programms ist es notwendig, alle möglichen Alternativen des Programms anzuzeigen. Für jeden Sitz wird jeweils ein Kandidat angezeigt, der aus allen möglichen gewählt werden kann. So ist zu jedem Zeitpunkt genau eine mögliche Variante des FPs zu sehen. Abbildung 6.7 zeigt den graphischen Editor für Flexible Programme.
- Zur Laufzeit ist es nicht sinnvoll, einen von mehreren Kandidaten anzuzeigen. Deshalb werden virtuelle Knoten nicht angezeigt. Der dargestellte Baum besteht aus instantiierten und abgeschlossenen Knoten.

Diese Darstellung des Programmablaufs ist während der Ausführung besonders für den Entwickler wertvoll. Ein Endanwender wird mit dieser Darstellung üblicherweise nicht konfrontiert; geeignete Verfahren hierfür werden in Kapitel 7.1 betrachtet. Es ist allerdings denkbar, diese Darstellung als Grundlage für Fortschritts- und Fehlermeldungen zu verwenden.

6.6 Zusammenfassung

Der Umfang der möglichen Handlungen (die aktivierten Verhaltensdispositionen) wird dynamisch anhand des Kontexts festgelegt. Dafür werden *Handlungsrahmen* definiert, die eine Untermenge der verfügbaren Kontextvariablen definieren und zu jedem Zeitpunkt als *gültig* oder *ungültig* gelten. Alle mit den gültigen Handlungsrahmen assoziierten Verhaltensdispositionen werden aktiviert. Ein weiterer Kontrollmechanismus wird durch Priorisierung der FPs erreicht. Dieser Mechanismus ist abhängig vom inneren Zustand der Flexiblen Programme. Zur Laufzeit wird der Handlungsbaum aufgebaut und instantiiert. Dabei werden Entscheidungen zwischen mehreren Kandidaten für dieselbe Handlung aufgrund der Vorbedingungen und der Bewertungsfunktion getroffen. Flexible Programme können sowohl zur Laufzeit als auch zum Bearbeiten graphisch dargestellt werden. Dies ist allerdings nicht für den Anwender gedacht, sondern dient vor allem dem Entwickler.

Kapitel 7

Komponenten zur Interaktion

Dem Benutzer eines Serviceroboters soll eine möglichst intuitive Schnittstelle zur Verfügung gestellt werden. Diese Schnittstelle dient zum einen der Kommandierung des Roboters, und zum anderen der Parametrierung und Adaption von Roboteraktionen.
Eine intuitive Schnittstelle setzt sich aus verschiedenen Modalitäten zusammen. Dazu gehören *Sprache* (siehe [Fügen 06, Krum 05]), Gestik und Blickrichtungserkennung (siehe [Stiefelhagen 04]), sowie visuelle Ausgaben. Insbesondere bei der intuitiven Programmierung ist es für den Roboter auch hilfreich, die vollständige Bewegung des Menschen zu erfassen, um Intention und Abläufe möglichst gut zu extrahieren.
Das vorliegende Kapitel beschreibt die im Rahmen dieser Arbeit entwickelten Funktionen zur Beobachtung des Menschen, sowie zur Ausgabe und Darstellung des internen Zustands des Roboters. Dabei wird auf die bereits in Kapitel 4 vorgestellten Modelle und Repräsentationen zurückgegriffen.

7.1 Darstellung des Roboterzustands für den Benutzer

Eine Rückmeldung des aktuellen Zustands des Roboters an den Benutzer ist unumgänglich. Dazu existieren verschiedene Ansätze. Grundsätzlich kann unterschieden werden zwischen

- Visualisierung der Aktivität und
- Darstellung von Emotionen.

Bei der Visualisierung der Aktivität des Roboters wird die momentane Aktion des Roboters wertfrei ausgegeben, wobei die Aktivität beispielsweise durch die jeweils ausgeführte oder geplante Aktion repräsentiert wird. Beispiele sind in [Giesler 05] im Bereich der erweiterten Realität zu finden. Im Gegensatz dazu steht die Darstellung von Emotionen. Dafür wird meist ein eigener *Emotionsraum* definiert, mit verschiedenen Gemütszuständen als Dimensionen. *Kismet* ist ein prominentes Beispiel (siehe [Breazeal 00, Sen 04]). Ähnlich sind die Zustandsdarstellungen der Roboter *Minerva* [Thrun 99a, Thrun 99b] und *Grace* [Simmons 03].
Im Rahmen dieser Arbeit wurde ein wesentlich stärker technisch orientierter Ansatz gewählt. Im Gegensatz zur Visualisierung von Gefühlszuständen wie *ärgerlich, erfreut, gelangweilt* oder *neugierig*, wie es z.B. bei Kismet der Fall ist, soll die aktuelle Aktivität des Roboters dargestellt werden. Dadurch wird der maschinelle Aspekt des Roboters über den Aspekt der

emotionalen Wirkung gestellt. Der Roboter ALBERT II ist dafür mit einem berührungssensitiven Bildschirm ausgestattet, das sowohl für Eingaben als auch für Ausgaben verwendet werden kann. Im Folgenden wird die Zustandsmodellierung sowie die Darstellung dieser Zustände für den Benutzer erläutert.

7.1.1 Zustandsmodellierung des Roboters

Der innere Zustand des Roboters ALBERT II wird zum einen beschrieben durch den Zustand aller Sensorik- und Aktorikkomponenten, zum anderen durch den Zustand der Handlungsrepräsentation. Diese Zustandsmengen können als subsymbolischer und symbolischer Zustand des Systems bezeichnet werden.
Um den Momentanzustand des Systems benutzerfreundlich darzustellen, muss die Darstellung auf einer Abstraktionsebene gewählt werden, die

- abstrakt genug ist, dass ein Benutzer die Aktionen des Roboters intuitiv verstehen kann, und die
- detailliert genug ist, dass Handlungen des Roboters logisch mitverfolgt werden können.

Um diesen Forderungen gerecht zu werden, werden Zustände der symbolischen Handlungsrepräsentation dargestellt. Die entsprechende Zustandsmenge wird wissensbasiert aufgebaut, und beinhaltet Zustände, die für einen Benutzer von erhöhtem Interesse während der Ausführung sind.
Die verwendete Menge an Zuständen des Serviceroboters ALBERT II umfasst die folgenden Zustände (siehe auch [Minar 04]):

- *Bereitschaft:* Ruhezustand des Roboters.
- *Warten:* Der Roboter hat momentan keine Aufgabe und wartet auf Eingaben.
- *Suchen:* Die Sensorik des Roboters wird verwendet, um den Sichtbereich nach Objekten, Positionen, Personen abzusuchen.
- *Beschäftigt:* Der Roboter befindet sich in diesem Zustand, solange er an einer Aufgabe arbeitet. Er befindet sich auch in diesem Zustand, solange er Berechnungen durchführt.
- *Erfolgreicher Abschluss:* Eine Aktion, die aus mehreren Teilaktionen zusammengesetzt sein kann, ist erfolgreich beendet worden.
- *Fehlschlag:* Eine Aktion ist fehlgeschlagen.
- *Fehler:* Ein schwerwiegender Fehler ist aufgetreten, den der Roboter nicht selbst beheben kann.
- *Fragen:* Der Roboter wartet auf eine konkrete Eingabe des Benutzers, z.B. nach eine Rückfrage.
- *Neuen Sachverhalt gelernt:* Ein neues Objekt, eine neue Handlung o.ä. ist erkannt und abgespeichert worden.

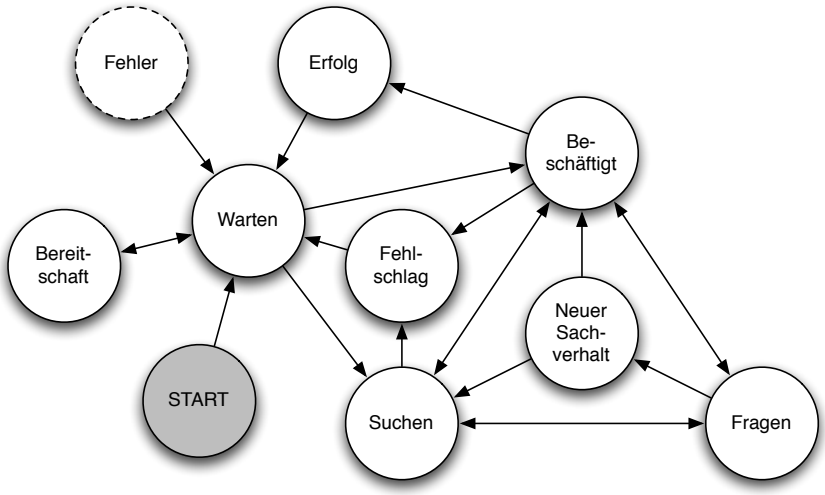


Abbildung 7.1: Zustände und Zustandsübergänge des Roboters für die Darstellung und Visualisierung. Jeder Zustand des Roboters und seiner Programme wird auf einen dieser Zustände abgebildet, und es sind nur die eingezeichneten Übergänge möglich. Der Zustand *START* wird bei der Initalisierung eingenommen, zum Zustand *Fehler* kann von jedem anderen gesprungen werden (der Übersichtlichkeit wegen nicht eingezeichnet).

Mit diesen Zuständen werden alle Abläufe und Aktionen des Roboters modelliert und dargestellt. Dabei sind nicht alle Zustandsübergänge möglich und sinnvoll. Abbildung 7.1 gibt die möglichen Zustände und Zustandsübergänge wieder. Zu beachten sind die Zustände *Fehler* und *START*: Zum Fehlerzustand führt von jedem Zustand eine Kante, die aus Übersichtlichkeitsgründen nicht eingezeichnet sind. Der Startzustand wird bei Initialisierung des Roboters eingenommen [Minar 04].

7.1.2 Darstellung der Roboterzustände

Zur Darstellung des inneren Zustands eines Roboters an der Mensch-Roboter-Schnittstelle ist es sinnvoll, jedem Zustand eine Animationssequenz zuzuordnen, die zur Visualisierung angewandt wird. Eine Animation wird als Pfad im sogenannten *Animationsraum* $\mathbb{A}$ definiert, und besteht aus einer Reihe von Vektoren $\vec{s}$, die die Stützpunkte der Animationssequenz $\mathcal{S}$ darstellen. Jedem Stützpunkt ist zusätzlich eine Dauer und eine Interpolationsart zugeordnet. So kann der Pfad der Animation bestimmt werden. Nicht jeder Stützvektor muss alle Dimensionen des Animationsraums enthalten; ist dies nicht der Fall, definiert die Animation eine *Schar* von Trajektorien. Es gilt also:

$$\mathcal{S} = \vec{s_0} \ldots \vec{s_n} \quad \text{mit} \quad (s_i | i = 1..n) \in \mathbb{A} \tag{7.1}$$

Mit einer Animation werden die Parameter zwischen zwei Zuständen kontinuierlich ineinander überführt. Obwohl die Zustände diskret sind, wechselt also die Visualisierung kontinuierlich und verfolgbar zwischen den Zuständen.
Die eigentliche Darstellung ist davon entkoppelt. Diese geschieht mit verschiedenen *Visualisierungen* $\mathcal{V}$. Diese Visualisierungen stellen graphische Objekte mit dem Parameterraum $\mathbb{V}$ dar. Diese werden mit der jeweils aktiven Animation gekoppelt. Die Schnittmenge der

Visualisierungsparameter mit den Dimensionen des aktuellen Animationsraums $\mathbb{A}$ sind die effektiven Parameter $\vec{v_e}$, die tatsächlich zu einer Änderung des graphischen Objekts führen:

$$\vec{v_e}(\mathcal{V}) = \mathbb{V} \cap \mathbb{A} \tag{7.2}$$

Die den Stützpunkten zugeordnete Interpolationsart ist entweder *Sprung*, *linear* oder *sigmoid* (siehe Abbildung 7.2).

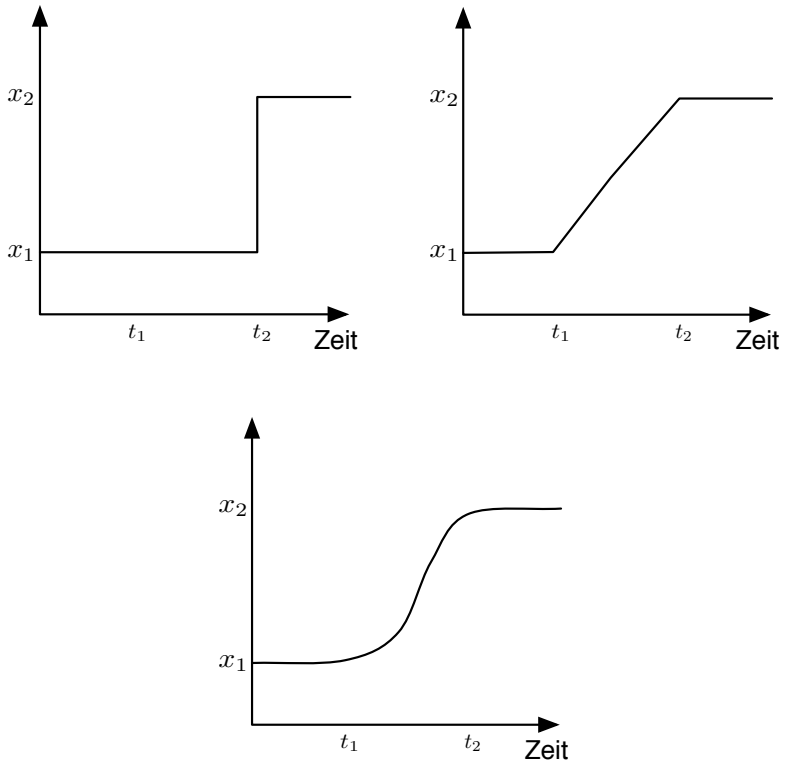


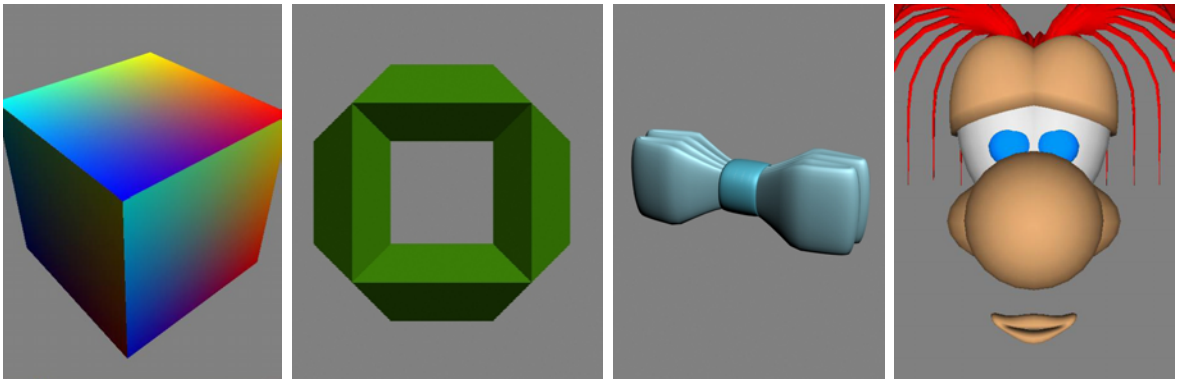
Abbildung 7.2: Verschiedene Interpolationsarten: (a) Sprung, (b) linear und (c) sigmoid

Die so erreichte Trennung zwischen Zustand mit zugehöriger Animation und Visualisierung gibt die Möglichkeit, Visualisierungen zur Laufzeit auszutauschen. Dadurch kann sich der Roboter an Veränderungen der Umwelt anpassen und beispielsweise die Visualisierung abhängig von den Benutzern ändern. Der Zustandsautomat sowie die Animationen werden davon nicht beeinflusst.
Es werden zur Zustandsdarstellung vier verschiedene dreidimensionale Visualisierungen vorgeschlagen (siehe Abbildung 7.3).

- Der Würfel kann sowohl in Größe und Position als auch in Farbe und Muster verändert werden. Änderungen sind leicht nachzuvollziehen, allerdings schwer zu interpretieren.
- Das Logo der Universität Karlsruhe (TH) kann in allen Raumdimensionen bewegt sowie skaliert werden.
- Die Fliege soll den Eindruck eines Butlers beim Menschen verstärken; auch stimmt die Bildschirmposition im Roboter mit der Position einer Fliege überein. Die Fliege kann

bewegt und in Form und Farbe bewegt werden, beispielsweise seitlich zu einem Lächeln gebogen oder signalrot gefärbt werden.

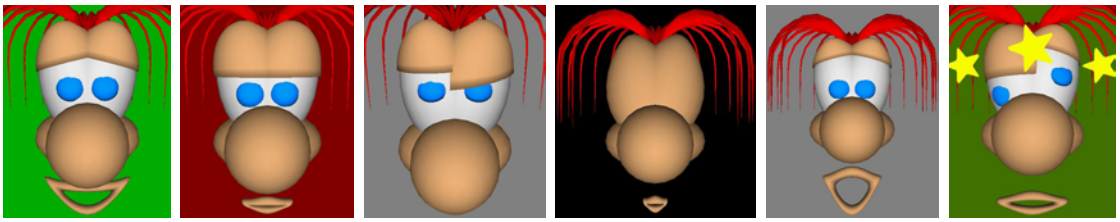
- Das Gesicht stellt einen Kompromiss zwischen einer wertfreien Zustandsdarstellung und der Visualisierung von Emotionen dar. Es kann verschiedene Ausdrücke annehmen, indem es bewegt und verzogen wird.



(a) (b) (c) (d)

Abbildung 7.3: Vier verschiedene Visualisierungen: (a) Abstrakte Box, (b) Logo der Universität, (c) Fliege und (d) Gesicht

Zur Animation des Gesichts wird dessen plakative Erscheinung genutzt. Comics enthalten eine Vielzahl von plakativen Symboliken, die weit verbreitet und so eindeutig einem Zustand zuzuordnen sind. Auch ist es nicht notwendig, Emotionen als Metapher zu visualisieren; vielmehr sollen, wie erwähnt, die Zustände der Abarbeitung betont werden.
Abbildung 7.4 gibt einige der möglichen Ausdrücke wieder. Eine Abfolge von Zuständen, die auch die weichen Übergänge der Animation zeigt, ist in Abbildung 7.5 gegeben.



(a) (b) (c) (d) (e) (f)

Abbildung 7.4: Verschiedene Ausdrücke zur Visualisierung der Systemzustände: (a) Erfolgreiche Ausführung (fröhlich), (b) Fehlschlag (unzufrieden), (c) fragend, (d) ausgeschaltet (schlafend), (e) Ausnahme (überrascht), (f) Fehler (Sterne)

7.1.3 Benutzereingaben über den Bildschirm

Um den Bildschirm nicht nur zur Ausgabe, sondern auch zur Eingabe zu nutzen, kann seine berührungssensitive Eigenschaft verwendet werden. Dabei werden drei verschiedene Arten von Rückfragen unterstützt:

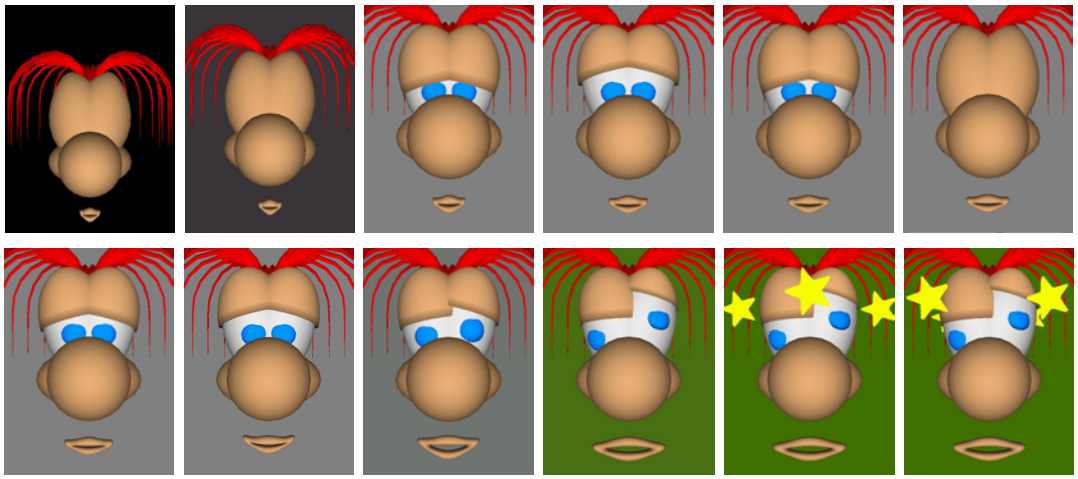


Abbildung 7.5: Einige Zustände während der Animation. Die Zustandsabfolge ist *ausgeschaltet – wartend – Fehler*.

- Bei der Auswahl einer von mehreren Textoptionen wird über der momentan angezeigten Visualisierung ein Auswahlfeld angezeigt. Dieses enthält mehrere Einträge, von denen der Benutzer eins auswählen kann. Tut er dies, wird das Auswahlmenü ausgeblendet und die getroffene Wahl an die rückfragende Komponente (ein Flexibles Programm) gemeldet. So kann beispielsweise die Spracherkennung unterstützt werden, oder die Auswahl verschiedener Handlungsalternativen dem Benutzer überlassen werden.

- Ähnlich wird die Auswahl eines von mehreren Bildern eingesetzt. Dabei werden dem Benutzer mehrere Bilder präsentiert, von denen eins ausgewählt werden kann. Dies kann z.B. bei mehrdeutigen Situationen während einer Manipulation hilfreich sein.

- Die Auswahl eines Bildbereichs, die über die Markierung zweier Eckpunkte eines Rechtecks geschieht, dient vor allem dazu, die Aufmerksamkeit des Roboters auf bestimmte Bereiche zu lenken. Beispielsweise können damit bisher unbekannte Objekte markiert und im Bild ausgeschnitten werden.

Abbildung 7.6 zeigt Beispiele dieser Rückfragen. Der Benutzer muss zur Auswahl den Bildschirm im entsprechenden Bereich berühren, um ein Element auszuwählen.

Mit der beschriebenen Zustandsdarstellung sowie den drei Rückfragemöglichkeiten ist es einfach möglich, über den Bildschirm den internen Zustand des Systems zu vermitteln. Die Zustandsdarstellung agiert dabei als ein Agent (siehe Kapitel 4.1.2), der sowohl Eingaben als auch Ausgaben zur Verfügung stellt. Zustandsänderungen werden durch entsprechende Aktionen in den laufenden Flexiblen Programmen ausgelöst, die bei der automatischen Abbildung von Makrooperatoren eingefügt werden.

7.2 Beobachtung menschlicher Bewegungen

Gestenerkennung ist ein wichtiger Teil der intuitiven Schnittstelle zwischen Mensch und Roboter. Um Gesten und andere Aktivitäten klassifizieren zu können, braucht der Roboter die

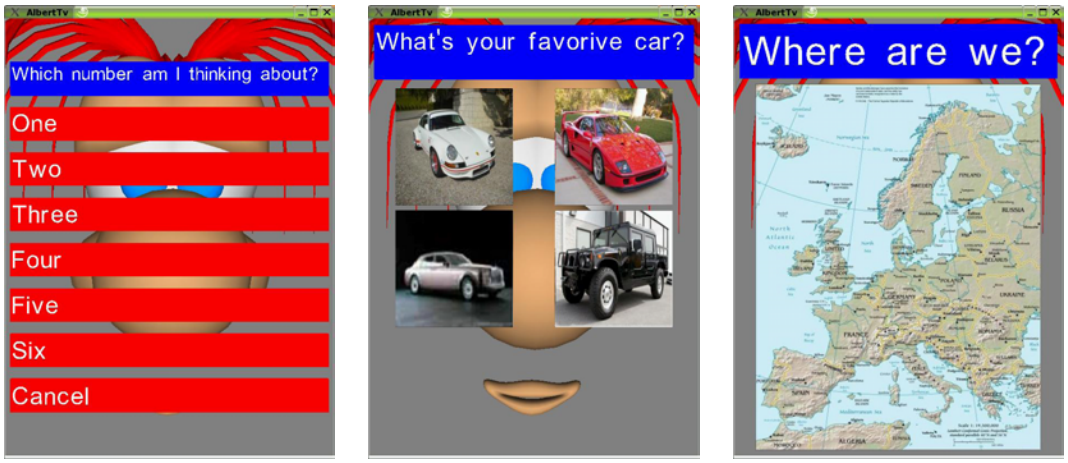


Abbildung 7.6: Beispiele der drei Rückfragetypen: Links eine von mehreren Optionen, Mitte eins von mehreren Bildern, rechts Bereich in einem Bild.

Fähigkeit, den Menschen zu beobachten. Die Schritte der Beobachtung und Klassifikation werden im Folgenden vorgestellt.

Das vorgeschlagene Verfahren beruht in Teilen auf dem Ansatz von Demirdjian[Demirdjian 03]. Dabei wird insbesondere die Verwendung des *Iterative-Closest-Point*-Verfahrens (Abkürzung: *ICP*) für mehrgliedrige Körpermodelle übernommen. Das von Demirdjian vorgestellte Verfahren zur Verfolgung der Körperpose hat allerdings einige in Kapitel 2.5.1 diskutierte Nachteile, insbesondere die fehlende Echtzeitfähigkeit sowie die Trennung zwischen Modellanpassung und dem Sicherstellen der Gelenkeinschränkungen. Dies wird durch die Verwendung eines Körper- und Gelenkmodells gelöst, wie es in Kapitel 4.4 vorgestellt wird.
Grundsätzlich handelt es sich bei dem ICP-Verfahren um ein modellbasiertes Schätzverfahren mit einer Zustandshypothese, wobei die Schätzung als Minimierungsproblem des Abstands zwischen Messung und Modell betrachtet wird. In einem ersten Schritt werden die Korrespondenzen zwischen Messung und Modell generiert, die dann in einem zweiten minimiert werden. Dies kann mit dem ICP Algorithmus nur für jedes starre Körperteil einzeln geschehen (der exakte Algorithmus ist in Anhang C aufgeführt).
Die Erweiterung des ursprünglichen ICPs ist in Abbildung 7.7 dargestellt (siehe auch [Knoop 07b]). Die Schritte der Messung, der Nächste-Punkte-Bestimmung und der Fusion mit Modellwissen werden im Weiteren detailiert betrachtet. Die Abstandsminimierung findet sich in Anhang C.

7.2.1 Zur Messung verwendete Sensorik

Zur Verfolgung der Körperpose wird ausschließlich die Sensorik des Roboters herangezogen. Die Verwendung weiterer, im Raum verteilter Sensoren (beispielsweise Deckenkameras) würde bei geschickter Wahl der Platzierung und der Verfahren die Verfolgung dynamischer Objekte stark vereinfachen; allerdings kann für einen Serviceroboter nicht davon ausgegangen werden, dass sich zusätzliche Sensorik in allen Verfahrbereichen befindet.
Die Sensorik des Roboters, die zur Posenschätzung und -verfolgung eingesetzt wird, umfasst eine Teilmenge der in Kapitel 3.2 beschriebenen Komponenten:

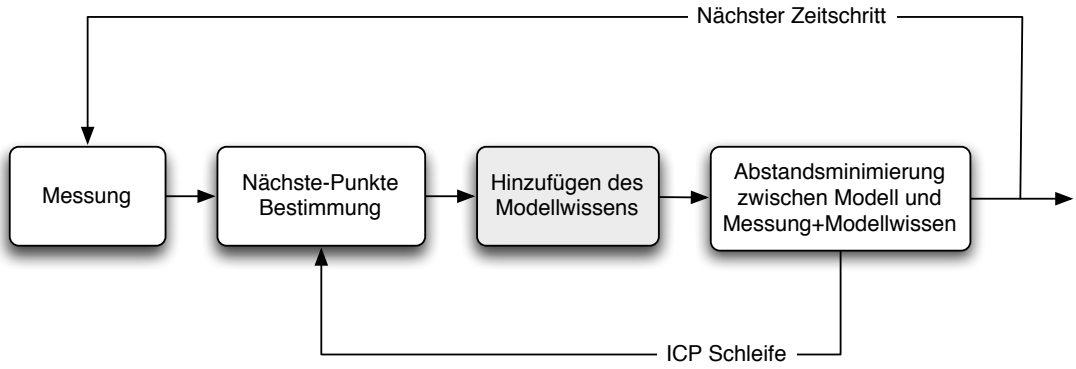


Abbildung 7.7: Modellbasierte Verfolgung der Körperpose mit zusätzlichen Randbedingungen aus Modellwissen. Der grau hinterlegte Teil stellt die Erweiterung gegenüber dem Verfahren in [Demirdjian 03] dar.

- Eine Tiefenbildkamera vom Typ *Swissranger* der Firma *CSEM* [CSEM 06] mit der Auflösung 160×124;
- Eine Stereo-Farbkamera des Herstellers *Videre* [Vid 06], die fest mit der Tiefenbildkamera verbunden ist;
- Einen Laserscanner der Firma *Sick*[SICK 06], der im Roboter zu Navigationszwecken genutzt wird und parallel zur Bodenfläche in ca. 20 cm Höhe die Umgebung vermisst.

Die Daten und Sensorspezifikationen sind in Anhang A aufgeführt.
Für die spätere Fusion der einzelnen Daten sind die Sensoren aufeinander kalibriert, die relativen Positionen also bekannt. So können beispielsweise Punktwolken der Tiefenbildkamera und des Laserscanners fusioniert werden.

7.2.2 Berechnung der Punktkorrespondenzen

Für die Berechnung der Transformation des Modells werden zunächst für die Messpunkte korrespondierende Modellpunkte berechnet. Nach Kapitel 4.4 besteht das Körpermodell aus degenerierten Zylindern. Es muss also für einen gegebenen Messpunkt der geometrisch nächste Punkt auf der Oberfläche des degenerierten Zylinders berechnet werden.
Abbildung 7.8 zeigt einen degenerierten Zylinder Z sowie einen Messpunkt Q. Gesucht ist der Punkt $\bar{Q}$ auf der Körperoberfläche, dessen Abstand d zu Q minimal ist. Dazu wird zunächst die Höhe h berechnet:

$$h = \left(\vec{q} - \vec{b}\right) \cdot \vec{l} \tag{7.3}$$

Nun kann eine Fallunterscheidung getroffen werden. Für Q existieren drei Fälle: Q kann unterhalb, innerhalb oder oberhalb des Zylinders liegen. Zusätzlich wird die Lage des Zylinders in der kinematischen Kette verwendet:

$$Q \quad \text{wird} \begin{cases} \text{verwendet} & 0 < h < H \quad \text{oder} \quad h > H \ \textit{und} \ Z \text{ ist ein Endzylinder,} \\ \text{verworfen} & \text{sonst.} \end{cases} \tag{7.4}$$

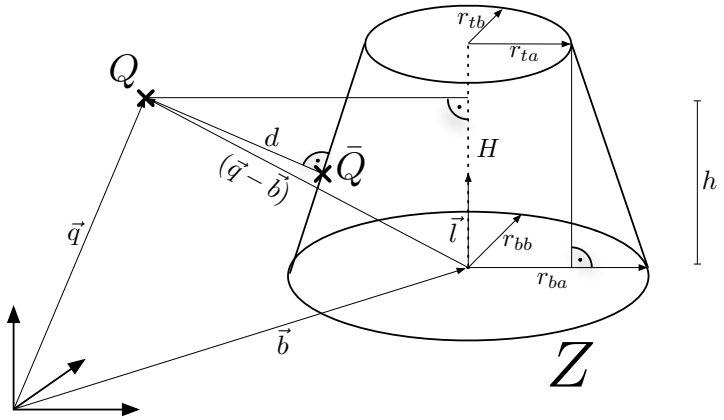


Abbildung 7.8: Berechnung des geometrisch nächsten Punktes $\bar{Q}$ auf der Oberfläche des Zylinders Z für einen gegebenen Messpunkt Q

Wenn Q verwendet wird, wird zur Berechnung von $\bar{Q}$ zunächst die Projektion von $\vec{q}-\vec{b}$ auf die Basisvektoren des Zylinders berechnet:

$$p_{r_{ba}} = (\vec{q}-\vec{b}) \cdot \frac{\vec{r}_{ba}}{|\vec{r}_{ba}|} \tag{7.5}$$

$$p_{r_{bb}} = (\vec{q}-\vec{b}) \cdot \frac{\vec{r}_{bb}}{|\vec{r}_{bb}|} \tag{7.6}$$

Jetzt können die Koordinaten von $\bar{Q}$ im Koordinatensystem von Z berechnet werden. Mit

$$y_0 = \sqrt{\frac{1}{\left(\frac{p_{r_{ba}}/p_{r_{bb}}}{|\vec{r}_{ba}|}\right)^2 + \frac{1}{|\vec{r}_{bb}|^2}}} \tag{7.7}$$

$$\text{gilt} \quad \begin{pmatrix} \bar{q_1}' \\ \bar{q_2}' \\ \bar{q_3}' \end{pmatrix} = \begin{pmatrix} y_0 \cdot \frac{\vec{r}_{ba}}{|\vec{r}_{ba}|} \\ y_0 \\ \text{Min}(H,h) \end{pmatrix}. \tag{7.8}$$

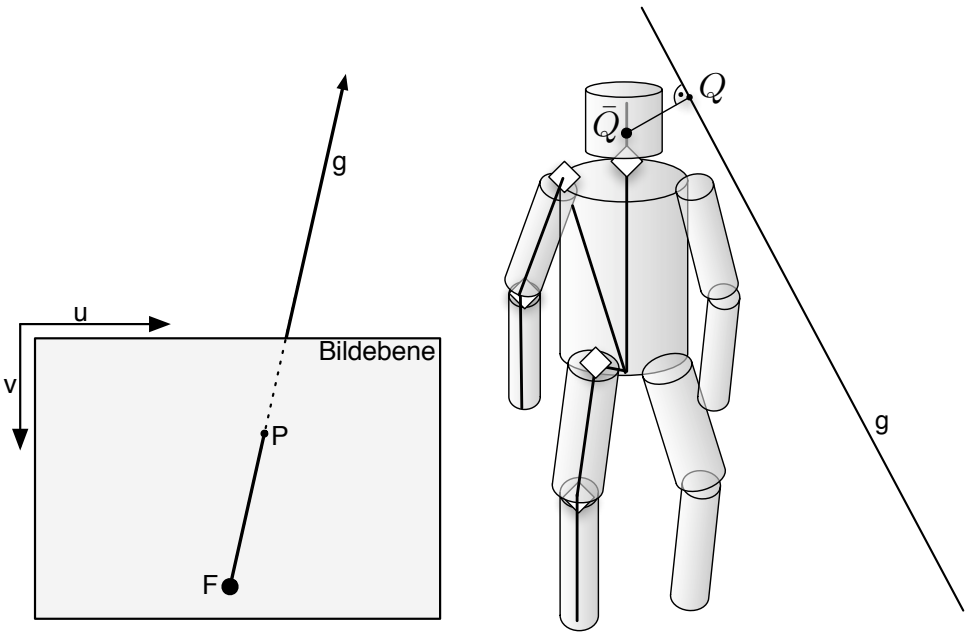
Dafür wird die Näherung $h_{\bar{Q}} \approx h_Q$ gemacht. Dies gilt für $(r_b - r_t \ll H)$ und kann im Allgemeinen angenommen werden. Durch diese Näherung werden rechenintensive Operationen vermieden, was der Gesamtgeschwindigkeit des Verfahrens zugute kommt. Für die Berechnung von $\bar{Q}$ werden die Komponenten noch abhängig von der Höhe und der Steigung der Außenfläche korrigiert:

$$\bar{Q} = \begin{pmatrix} \pm\bar{q_1}' + \frac{|\vec{r}_{ta}-\vec{r}_{ba}|}{H} \\ \pm\bar{q_2}' + \frac{|\vec{r}_{tb}-\vec{r}_{bb}|}{H} \\ \bar{q_3}' \end{pmatrix} \tag{7.9}$$

Das Vorzeichen ist dabei abhängig vom Quadranten, in dem sich Q befindet. $\bar{Q}$ stellt nun den zu Q korrespondierenden Punkt auf Z dar.

7.2.3 Fusion unterschiedlicher Messungen

Die Tiefenbildkamera und der Laserscanner liefern 3D-Punktwolken. Diese können mit den aufgeführten Berechnungen als Punktkorrespondenzen ausgedrückt werden. Desweiteren wird die Kamera verwendet, deren Farbbilder mit Standardverfahren zur Gesichts- und Hautfarberkennung verwendet werden, um Gesicht und Hände zu erkennen und zu verfolgen [Lösch 05, Viola 01]. Dies liefert Positionen im 2d-Bild. Diese können wiederum in Punktkorrespondenzen ausgedrückt werden, um die Messungen direkt in der ICP-Minimierung zu verwenden. Eine solche Messung besteht aus dem gefundenen Merkmal und seiner Lage im Bild, beispielsweise dem Mittelpunkt eines im Bild gefundenen Gesichts an Punkt $P(u_P, v_P)$. Diese Position wird mit Hilfe der inversen Messgleichung aus dem klassischen Lochkameramodell nach 3D in den Raum des Modells zurück projiziert. Daraus ergibt sich eine Halbgerade g, die laut Messung das gefundene Merkmal enthält (Abbildung 7.9(a)). Mit dieser Halbgeraden und dem Modellpunkt kann dann über geometrische Betrachtungen die Punktkorrespondenz aufgebaut werden (Abbildung 7.9(b)).
Auf diese Weise erreicht man eine Punktkorrespondenz, die der Messung entspricht: Die Messung macht eine Aussage über die Position des Merkmals orthogonal zum Sichtstrahl, nicht jedoch in der Tiefe parallel dazu. Die Punktkorrespondenz erreicht genau diese Aussage als Vorgabe für die Abstandsminimierung, da $\overrightarrow{Q\bar{Q}}$ immer senkrecht auf g steht.



(a) (b)

Abbildung 7.9: Projektion eines in P gefundenen Merkmals auf eine Halbgerade im Raum (a). Diese Halbgerade enthält das Merkmal. Die Punktkorrespondenz zwischen Gerade und Modellpunkt kann geometrisch bestimmt werden, indem der dem Modellpunkt $\bar{Q}$ nächste Punkt Q auf der Geraden bestimmt wird (b).

Genauso können Messungen, die Merkmale mit Flächen assoziieren, hinzugefügt werden. Die Korrespondenz besteht dann aus dem Merkmalspunkt $\bar{Q}$ auf dem Modell sowie dem zu $\bar{Q}$

geometrisch nächsten Punkt Q in der Ebene. Messungen dieser Art können beispielsweise durch eindimensionale Richtmikrophone erzeugt werden.
Die verwendete Abbildung der Messung in den Raum des Modells entspricht dabei der inversen Messgleichung. Daraus ergibt sich in großer Vorteil des Verfahrens: Die Messgleichung eines Sensors ist durch das zu Grunde liegende Messprinzip im Allgemeinen bekannt. Die umgekehrte Abbildung lässt sich also ohne Schwierigkeiten aufstellen; je nach Messprinzip enthält die resultierende Abbildung dann Mehrdeutigkeiten, wie dies beispielsweise bei der Abbildung eines Bildpunkts auf den Sichtstrahl der Fall ist.

7.2.4 Randbedingungen aus der Gelenkmodellierung

Abbildung 7.10 zeigt das Verhalten eines Körpermodells ohne Gelenkeinschränkungen über mehrere Sekunden einer Sequenz. Aufgrund der fehlenden Randbedingungen driften die Teile auseinander. Es ist also notwendig, Randbedingungen zu formulieren.

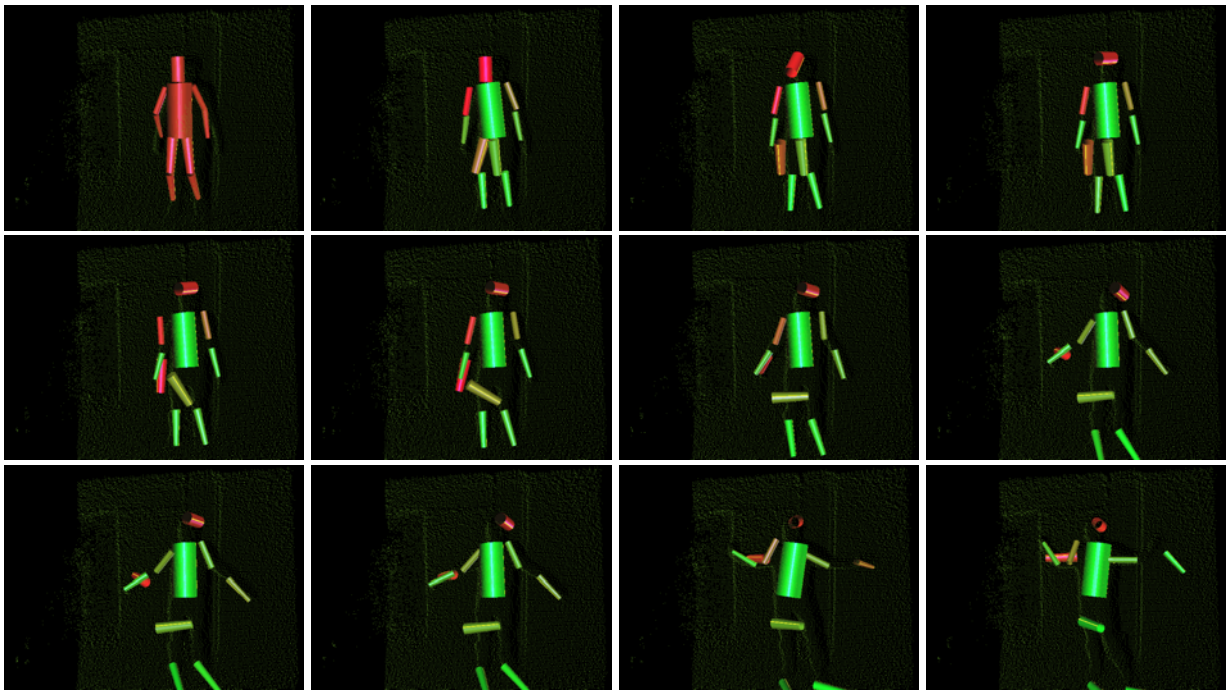


Abbildung 7.10: ICP-Algorithmus ohne explizite Wiederherstellung der Gelenkeinschränkungen. Das Auseinanderdriften der einzelnen Körperteile ist gut zu erkennen.

Um den zusätzlichen Schritt der Wiederherstellung der Gelenkeinschränkungen nach Abbildung 2.19 zu umgehen, werden die in Kapitel 4.4.2 beschriebenen Gelenkeinschränkungen direkt für den ICP aufbereitet. Dafür wird das vorhandene Modellwissen über die Gelenke als Puntkorrespondenzen ausgedrückt und direkt den Korrespondenzen aus der Messung und der Nächste-Punkte-Bestimmung hinzugefügt (siehe Abbildung 7.7).
Dazu sollen zunächst die Eigenschaften der bereits aus Messungen und Körpermodell aufgebauten Punktkorrespondenzen betrachtet werden. Eine Punktkorrespondenz stellt eine Zuordnung eines Messpunkts zu einem Modellpunkt dar. Die gewichtete Summe über die Abstandsquadrate aller Korrespondenzen wird dann minimiert. Die Korrespondenzen stellen

also Kräfte zwischen Welt und Modell dar, die mit elastischen Bindungen verglichen werden können. Die Gesamtenergie über alle elastischen Bindungen wird dann minimiert.
Das in Kapitel 4.4.2 vorgestellte Gelenkmodell zur Zuordnung korrespondierender Punkte auf benachbarten Körperteilen in einem Gelenk kann ebenfalls in diese Repräsentation übertragen werden.
Man betrachte dazu Abbildung 7.11. Die Zuordnung der Modellpunkte erfolgt folgendermaßen: Für L_1 und L_2 werden zunächst die Korrespondenzen aus allen relevanten Messpunkten aufgestellt (siehe Kapitel 7.2.2). Zu dieser Korrespondenzenmenge wird eine weitere hinzugefügt, die aus P_1 als Modellpunkt und P_2 als Korrespondenzpunkt besteht. Für den ICP wird also P_2 als Messpunkt von P_1 modelliert. Dies stellt eine Kraft auf, die L_1 auf L_2 zutreibt. Für L_2 wird die umgekehrte Korrespondenz aufgestellt. Eine weitere Korrespondenzbeziehung mit P_2 als Modellpunkt un P_1 als Welt- bzw. Messpunkt stellt eine Kraft auf, die L_2 auf L_1 zutreibt.

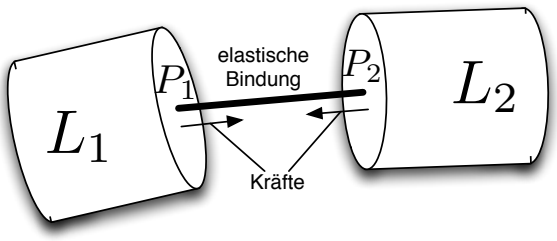


Abbildung 7.11: Gelenkkräfte in der in Kapitel 4.4.2 vorgestellten Gelenkrepräsentation. Der Punkt P_1 des Körperteils L_1 erfährt eine Kraft in Richtung von P_2, während an L_2 eine Kraft in P_2 angreift. Diese zeigt in Richtung P_1.

Die resultierenden Kräfte treiben also die jeweiligen Endpunkte der modellierten elastischen Bänder aufeinander zu. Das vorgeschlagene Modellwissen hat damit dieselbe Form wie die Messungen und wird im selben Schritt vom ICP-Algorithmus einbezogen.
Die Bindungskräfte können über die Gewichtungsparameter eingestellt werden. Eine Gelenkbeziehung muss ein wesentlich höheres Gewicht erhalten als ein einzelner Messpunkt, um die Gelenke dominant gegenüber der Messungen zu halten. Über das Gewicht kann also gewählt werden, wie strikt ein Gelenkmodell eingehalten wird. Es ist sinnvoll, das Gewicht eines Gelenkmodells abhängig von der Summe der Gewichte der Messungen M_k auf dem zugehörigen Körperteil k zu wählen. Sinnvolle Werte für das Gesamtgewicht g_k eines Gelenks liegen bei $0.5 \cdot M_k < g_k < M_k$.
Die in Kapitel 4.4.2 vorgestellte Gelenkmodellierung kann also in Form von Punktkorrespondenzen ausgedrückt werden, die wiederum vom ICP direkt im Minimierungsschritt zusammen mit den Messungen verwendet werden.

7.2.5 Ablauf der Verarbeitung

Abbildung 7.12 zeigt den vollständigen Algorithmus zur Verfolgung der Körperpose. Der Hauptaufwand geht dabei in den Aufbau der Punktkorrespondenzen für den ICP, die für alle Körperteile separat aufgestellt werden. Es muss für jedes Sensordatum entschieden werden, ob es als Messung des Körpers verwendet wird, und zu welchem Körperteil es gezählt

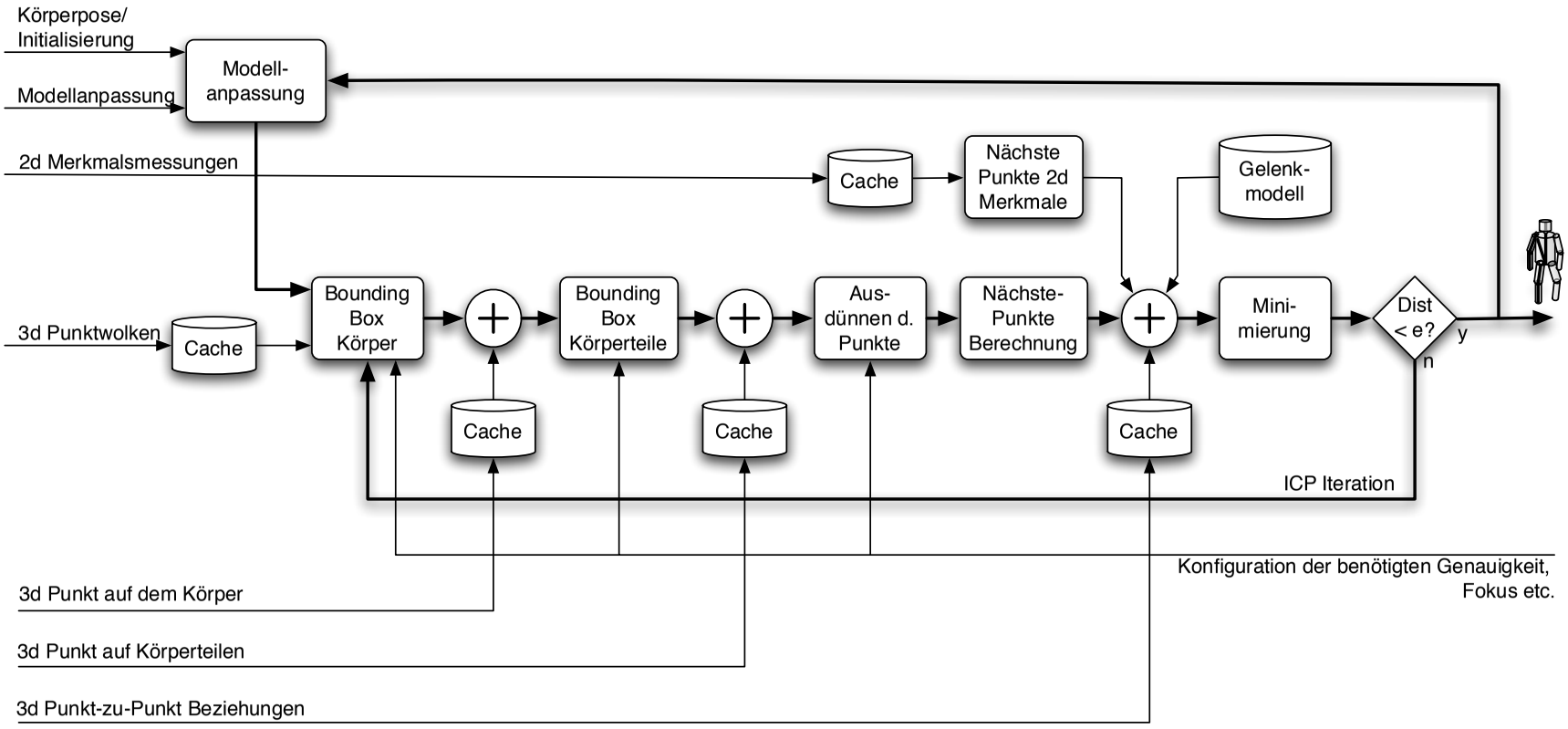


Abbildung 7.12: Ablauf des Algorithmus für die Verfolgung der Körperpose. Auf der linken Seite befinden sich die Eingänge für Sensordaten. Alle Sensordaten werden für den mehrmaligen ICP-Durchlauf zwischengespeichert (*Cache*). Die dickeren Linien bezeichnen zusätzlich zum Datenfluss den zeitlichen Ablauf. Die Hauptschritt sind also: Bewertung der 3D Punktwolken anhand des einhüllenden Quaders (engl.: *Bounding Box*), Bewertung anhand der einhüllenden Quader der Körperteile, Ausdünnen der Punkte, Berechnung der Punktkorrespondenzen, Hinzufügen des Modellwissens und zusätzlicher Messungen, Minimierung.

wird. Punkte, die keinem Körperteil zugeordnet werden können, werden als Hintergrundmessungen betrachtet und verworfen. Messungen, die aufgrund des Messprinzips schon eine Zuordnung mitbringen (z.B. hautfarbenbasierte Verfolgung im Farbbild, die eindeutig Gesicht und Händen zugeordnet werden kann), werden entsprechend aufbereitet und als Punktkorrespondenzen hinzugefügt. Letztendlich erfolgt die Minimierung, die alle Messungen und das Modellwissen gleichzeitig beachtet. Diese Schritte werden iterativ so lange durchgeführt, bis ein Qualitätskriterium erfüllt ist. Die einzelnen Schritte der Verarbeitung werden im Folgenden erläutert.

Vor der Berechnung eines Datensatzes können die Parameter des Modells in der *Modellanpassung* verändert werden. Hier kann beispielsweise die Modellgröße verändert, oder die Körperpose durch eine Intialisierung gesetzt werden. Es können auch Modellteile hinzugefügt oder entfernt werden. Dies ist insbesondere interessant, wenn der Mensch ein Objekt greift, das als weiterer Zylinder an der Hand modelliert werden kann.
Alle Punkte der eingegebenen Punktwolken werden mit dem einhüllenden Quader (engl.: *Bounding Box*) des Gesamtmodells verglichen (Schritt *Bounding Box Körper*). Punkte, die außerhalb des Quaders liegen, werden als Hintergrundmessungen klassifiziert und verworfen. Dieser Schritt basiert auf der Annahme, dass sich die Körperkonfiguration von einem Zeitschritt zum nächsten nur lokal ändert, wobei die Änderung durch eine parametrierbare Vergrößerung des einhüllenden Quaders in Betracht gezogen wird. Die resultierende Punktwolke wird mit etwaigen Punkten konkateniert, die von der Sensorik schon als Messungen des zu verfolgenden Körpers markiert wurden. Ergebnis ist eine Liste von Punkten, die sich nahe am Körpermodell befinden und deshalb als Kandidaten für Messungen des Körpers gelten.
Im nächsten Schritt werden 3D-Punkte einzelnen Körperteilmodellen zugeordnet. Dies geschieht über einen ähnlichen Ansatz: Jeder 3D-Punkt wird mit dem einhüllenden Quader jedes Körperteils verglichen (Schritt *Bounding Box Körperteile*). Die Quader können wiederum mit einem Parameter vergrößert werden, um schnelle Bewegungen verarbeiten zu können. Punkte, die in keinem einhüllenden Quader liegen, werden wiederum verworfen. Punkte, die in mehr als einem Quader liegen (bei Überlappung der einhüllenden Quader), können unterschiedlich behandelt werden:

- Der Punkt wird allen beteiligten Körperteilen anteilig zugeordnet.
- Der Punkt wird exklusiv einem der beteiligten Modellteile zugeordnet.
- Der Punkt wird genau dann anteilig zugeordnet, wenn die beteiligten Modellteile im Modellaufbau benachbart sind.

Im Normalfall wird die letztgenannte Variante verwendet. Dies vermeidet Kollisionen zwischen nicht benachbarten Körperteilmodellen, indem die Punktmengen der Modellteile strikt getrennt werden. Die resultierende Punktmenge enthält Punkte mit Zuordnung zu Körperteilmodellen. Diese wird wiederum mit eventuellen 3D-Punkten konkateniert, die bereits durch die Messung einzelnen Körperteilen zugeordnet sind (z.B. aus hautfarbenbasierter Verfolgung mit einer Stereokamera).
Die Anzahl der Punkte kann in einem weiteren Schritt reduziert werden. Durch *Ausdünnen der Punktwolke* wird der Berechnungaufwand für nachfolgende Schritte, vor allem die Nächste-Punkte-Berechnung, reduziert. Dieser Schritt wird durch drei Parameter beschrieben: Der

Reduktionsfaktor, das Minimum und das Maximum an Punkten pro Modellteil. Durch festlegen eines Minimums und Maximums wird verhindert, dass Modellteile, die durch wenige Messungen beschrieben sind, eine weitere Reduktion erfahren. Auf der anderen Seite werden in der Messung stark vertretene Modellteile auch stark ausgedünnt.
Nun erfolgt die *Nächste-Punkte-Berechnung*. Für jeden Messpunkt in der ausgedünnten Punktwolke wird dabei der nächste Punkt auf dem zugehörigen Modellteil berechnet. Punkte mit einem Abstand zwischen Modell und Messpunkt über einem Schwellwert werden wiederum verworfen. Wenn 3D-Messungen für Merkmale des Modells vorliegen, werden diese zu den berechneten Korrespondenzen hinzugefügt.
Durch Rückprojektion auf Sichtgeraden gewonnene Korrespondenzen werden weiterhin hinzugefügt, genauso wie aus dem Gelenkmodell gewonnene Korrespondenzen. Diese werden abhängig vom modellierten Gelenktyp erzeugt.
Die so erstellte Menge an Korrespondenzen enthält alle Information aus Messungen und Modellierung. Damit kann die Transformation berechnet werden, die auf das Modell angewandt werden muss, um die Summe der Abstandsquadrate zu minimieren. Der Vorgang wird wiederholt, bis das in Anhang C gegebene Qualitätsmaß erfüllt oder eine festgelegte Anzahl an Schritten erreicht ist.

7.2.6 Sensormodell

Jede Datenquelle, die zur Sensorfusion genutzt wird, besitzt ein charakteristisches Rauschverhalten. Diese Unsicherheiten werden durch die Assoziation jeder Messung mit einem Gewicht modelliert.
Dazu wird jedes Datum, das im ICP verwendet wird, mit einer Gewichtung versehen, die die Genauigkeit beschreibt. Diese Gewichte werden im Minimierungsschritt einbezogen. Auf diese Weise wird beispielsweise eine hautfarbenbasierte Verfolgung des Gesichts wesentlich höher bewertet als ein einzelner Datenpunkt einer Punktwolke.
Die Höhe des Gewichts hat dabei keinen Einfluss auf den Rechenaufwand des Algorithmus. Dieser hängt allein von der Anzahl der Messungen, der Anzahl der Körperteile und der Dynamik der Szene ab.

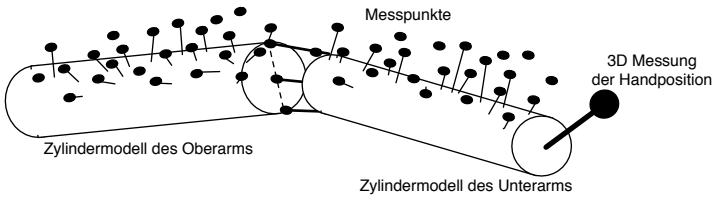


Abbildung 7.13: Verschiedene Gewichte für unterschiedliche Messungen. Die Höhe des Gewichts soll durch die Stärke der Punkte und Linien verdeutlicht werden. Die Punkte aus der 3D-Punktwolke des Tiefenbildsensors werden wesentlich niedriger bewertet als eine 3D-Messung der Handposition. Letztere kann beispielsweise aus einer farbbasierten Erkennung der Hand mit anschließender Stereorekonstruktion berechnet werden.

Abbildung 7.13 zeigt eine Beispielkonfiguration mit Messdaten aus verschiedenen Quellen. Die Punkte der 3D-Punktwolke der Tiefenbildkamera besitzen alle dasselbe Gewicht,

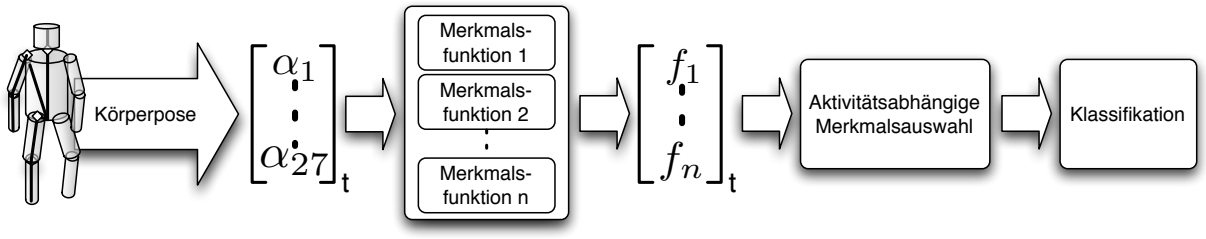


Abbildung 7.14: Ansatz zur Klassifikation von Gesten und Aktivitäten

während eine hypothetische 3D-Messung der Handposition, beispielsweise durch farbbasierte Erkennung und Stereorekonstruktion, ein wesentlich höheres Gewicht besitzt. Genauso werden die künstlich erzeugten Korrespondenzen aus der Gelenkmodellierung (siehe Kapitel 7.2.4) so gewichtet, dass die Gelenkmodelle dominant gegenüber den Messungen sind. Der Effekt der Gewichtung kann wiederum anhand der Analogie der elastischen Bänder erklärt werden: Das Gewicht einer Korrespondenz legt die Zugkraft des zugehörigen Bandes fest. Bei der Minimierung der Gesamtenergie aller elastischen Bänder durch Transformation des Modells wird die unterschiedliche Stärke der einzelnen Bänder berücksichtigt.

7.3 Klassifikation von Aktivitäten

Die Beobachtung der Bewegungen des Menschen kann einerseits genutzt werden, um die Bewegungs- und Manipulationsplanung des Roboter zu parametrieren [Alami 06]. Andererseits kann die Bewegung direkt zur Klassifikation von Aktivitäten verwendet werden. Das vorgeschlagene Körpermodell besitzt 27 Bewegungsfreiheitsgrade. Die Trajektorien in diesem Konfigurationsraum werden zur Klassifikation der beobachteten Aktivitäten genutzt.
Entsprechend der in Kapitel 4.5 vorgestellten Modellierung menschlicher Aktivitäten entspricht eine Aktivität einem Symbol in der Kommunikation, dessen Beobachtung nur eine Ausprägung ist. Eine Aktivität kann also auf verschiedene Arten ausgeführt werden. Grundsätzlich sind Aktivitäten zusammengesetzt aus einer endlichen Menge an Elementaroperationen, die beobachtet werden können.

7.3.1 Ansatz zur Klassifikation

Diese Eigenschaft wird für die Erkennung genutzt. Zunächst wird eine Menge an Merkmalen definiert, die anhand der beobachteten Bewegung bestimmt werden können. Diese Merkmale stellen elementare Operationen ausgeführter Gesten dar. Auf Basis dieser Merkmale werden Aktivitäten klassifiziert.
Dieser Ansatz zur Erkennung primitiver Aktivitäten ist in Abbildung 7.14 verdeutlicht. Die beobachtete Bewegung des Körpermodells wird durch die Freiheitsgrade des Modells $\alpha_1 \dots \alpha_{27}$ ausgedrückt. Aus dieser Bewegung wird mittels verschiedener Funktionen ein Merkmalsvektor $f_1 \dots f_n$ erzeugt. Aus dieser Menge an Merkmalen werden für jede Aktivität die relevanten Merkmale bestimmt und sowohl für das Training als auch den Klassifikationsschritt eines Klassifikators verwendet.

Die Aktivitäten entsprechen in erster Linie Gesten und einfachen Aktionen wie *Hinsetzen*, *Laufen* und *Stehen*. Dabei wird die Menge der Aktivitäten eingeschränkt auf solche, die anhand einer kurzen Bewegungssequenz mit beschränktem Zeitfenster erkennbar sind. Komplexe (also zusammengesetzte) Aktivitäten, für die eine Beobachtung über längere Zeitintervalle notwendig ist, werden an dieser Stelle nicht betrachtet. Handlungen lassen sich mit Hilfe von Bayes-Schätzern (siehe [Schrempf 05]) oder HMMs (Abkürzung für: *Hidden Markov Models*) charakterisieren und klassifizieren, wie in [Kellokumpu 05, Oliver 02]. Diese stellen einen Spezialfall der Modellierung nach Bayes dar und sind geeignet, zeitliche Folgen zu modellieren und zu erkennen.
Die Schritte der Extraktion und Bewertung von Merkmalen sowie der Klassifikation von Aktivitäten werden im Folgenden diskutiert (siehe auch [Brännström 06, Otero 06a]).

7.3.2 Extraktion von Merkmalen

Aus den Daten der beobachteten Bewegung werden im ersten Schritt sinnvolle Merkmale extrahiert. Sinnvolle Merkmale im Sinne der Klassifikation sind solche, die *charakteristisch für mindestens eine, aber nicht alle* Ausprägungen aller Aktivitäten sind, und die für eine zu erkennende Ausprägung *invariant* sind.
Zur Untersuchung der Eigenschaften der Merkmale werden drei Arten von Merkmalen verwendet:

- Als *primitive Merkmale* werden die Trajektorien der Bewegungsfreiheitsgrade des Modells genutzt. Grundsätzlich sind alle Informationen der Bewegung darin enthalten, allerdings oft nicht explizit. Die hohe Kopplung und Abhängigkeit der primitiven Merkmale untereinander macht es in vielen Fällen schwierig, Gesten direkt zu klassifizieren.

- *Kombination von Merkmalen* nach heuristischen Gesichtspunkten wird verwendet, um charakteristische Merkmale für einzelne Aktivitäten zu extrahieren. Beispielsweise kann das Höhenniveau der Hände im Vergleich zur Körpergröße eines Menschen sehr gut verwendet werden, um verschiedene Aktivitäten zu unterscheiden. Zu diesen Merkmalen gehören auch die ersten Ableitungen der Merkmale, also die Geschwindigkeiten der Freiheitsgrade des Modells.

- Eine weitere Menge von Merkmalen wird mit Hilfe *statistischer Methoden* erzeugt. Die verwendeten statistischen Maße umfassen die Berechnung der Kovarianz zwischen verschiedenen primitiven Merkmalen, Hauptkomponentenanalyse (engl.: *Principal Component Analysis, PCA*), sowie die Berechnung von auftretenden Periodizitäten mit Hilfe der Frequenzanalyse.

Dabei zeigt sich, dass Merkmale entweder zeitabhängig oder stationär sein können. Für die Bestimmung der Geschwindigkeiten oder der Periodizitäten muss also ein Zeitfenster definiert werden, das betrachtet wird. Dieses wird anhand der Untersuchungen zu Häufigkeit und Dauer von Aktivitäten in [Otero 06a, Otero 06b] gewählt und umfasst die Messungen der letzten 1.5 Sekunden. Die vollständige Menge an extrahierten Merkmalen umfasst 118 Merkmale und ist in Tabelle 7.1 wiedergegeben.

#	Beschreibung	#	Beschreibung
1-3	ϕ Torso	82-84	$\dot{p}$ Linke Hand
4-6	ϕ Kopf	85-87	$\dot{p}$ Rechter Fuß
7-9	ϕ Linker Oberarm	88-90	$\dot{p}$ Linker Fuß
10-12	ϕ Rechter Oberarm	91	Planarität p Torso
13-15	ϕ Linker Oberschenkel	92	Planarität p Rechte Hand
16-18	ϕ Rechter Oberschenkel	93	Planarität p Linke Hand
19-21	ϕ Linker Unterarm	94	Planarität p Rechter Fuß
22-24	ϕ Rechter Unterarm	95	Planarität p Linker Fuß
25-27	ϕ Linker Unterschenkel	96	Varianz p Rechte Hand
28-30	ϕ Rechter Unterschenkel	97	Varianz p Linke Hand
31-33	p Kopf	98	Kovarianz p Hände
34-36	p Rechte Hand	99	Kovarianz p Füße
37-39	p Linke Hand	100	Varianz p Torso
40-42	p Rechter Fuß	101	Kovarianz p Knie
43-45	p Linker Fuß	102	Periodizität Handabstand
46-48	$\dot{\phi}$ Torso	103	Periodizität Fußabstand
49-51	$\dot{\phi}$ Kopf	104	Periodizität ϕ Rechtes Knie
52-54	$\dot{\phi}$ Linker Oberarm	105	Periodizität ϕ Linkes Knie
55-57	$\dot{\phi}$ Rechter Oberarm	106	Periodizität ϕ Rechter Oberschenkel
58-60	$\dot{\phi}$ Linker Oberarm	107	Periodizität ϕ Linker Oberschenkel
61-63	$\dot{\phi}$ Rechter Oberschenkel	108	Periodizität ϕ Rechter Unterarm
64-66	$\dot{\phi}$ Linker Unterarm	109	Periodizität ϕ Linker Unterarm
67-69	$\dot{\phi}$ Rechter Unterarm	110	Periodizität ϕ Rechter Oberarm
70-72	$\dot{\phi}$ Linker Unterschenkel	111	Periodizität ϕ Linker Oberarm
73-75	$\dot{\phi}$ Rechter Unterschenkel	112	Abstand der Hände
76-78	$\dot{p}$ Torso	113-115	Mittelwert p Rechte Hand
79-81	$\dot{p}$ Rechte Hand	116-118	Mittelwert p Linke Hand

Tabelle 7.1: Verwendete Merkmale aus den Bewegungstrajektorien für die Klassifikation von Aktivitäten. Dabei bezeichnet ϕ einen Winkel, $\dot{\phi}$ die Winkelgeschwindigkeit, p die Position, und $\dot{p}$ die kartesische Geschwindigkeit. Alle Positionen sind relativ zur Körperposition (Torso) betrachtet. Torsopositionen und Geschwindigkeiten werden relativ zum Roboter berechnet.

7.3.3 Bewertung der Merkmale

Zur Klassifikation sollen nur die *relevanten* Merkmale verwendet werden. Dabei ist intuitiv klar, dass die Menge der relevanten Merkmale nicht global festgelegt werden kann, sondern von der Natur der zu klassifizierenden Aktivität abhängt. Beispielsweise werden zur Erkennung von *Winkgesten* andere Merkmale benötigt als für *Laufen*.
Es muss also ein aktivitätsabhängiges Maß zur Bewertung der Relevanz eines Merkmals gefunden werden. Allgemein muss für ein relevantes Merkmal F_i und eine zu klassifizierende Aktivität C für das Auftreten der Klassen- und Merkmalswerte c und f_i gelten:

$$p(c = C|f_i = F_i) \neq p(c = C) \tag{7.10}$$

Die Auftretenswahrscheinlichkeit der Klasse C nach Beobachtung des Merkmals F_i muss sich also unterscheiden von der Auftretenswahrscheinlichkeit von C.
Weiterhin kann das Maß für Relevanz unterteilt werden in *starke* und *schwache Relevanz* (siehe auch [Battiti 94, Yu 04]).
Ein Merkmal F_i wird dann als *stark relevant* bezeichnet, wenn es eine Merkmalsmenge S_i aller Merkmale ohne F_i, also $S_i = \{F_1, \ldots, F_{i-1}, F_{i+1}, \ldots, F_n\}$, sowie Werte f_i, c und s_i mit $p(F_i = f_i, S_i = s_i) > 0$ gibt, so dass gilt

$$p(c = C|f_i = F_i, s_i = S_i) \neq p(c = C|s_i = S_i). \tag{7.11}$$

Im Gegensatz dazu wird ein Merkmal F_i als *schwach relevant* bezeichnet, das nicht stark relevant ist und für das eine Untermenge S_i' von S_i sowie Werte f_i, c und s_i' mit $p(f_i = F_i, s_i' = S_i') > 0$ existieren, so dass

$$p(c = C|f_i = F_i, s_i' = S_i') \neq p(c = C|s_i' = S_i') \tag{7.12}$$

gilt.
Starke Relevanz bedeutet also, dass ein Merkmal nicht entfernt werden kann, ohne Genauigkeit im Ergebnis zu verlieren. Ein schwach relevantes Merkmal kann nicht unter allen Umständen entfernt werden. Schwach relevante Merkmale besitzen also Überlappungen mit mindestens einem anderen Merkmal.
Um für jedes Merkmal und jede Aktivitätsklasse die Relevanz bestimmen zu können, werden diese Abhängigkeiten der einzelnen Merkmale und Klassen bestimmt. Dies kann im Normalfall nicht analytisch geschehen, da die zu Grunde liegenden Prozesse der menschlichen Bewegungen und Aktivitäten nicht exakt beschrieben werden können. Entsprechend wird die Relevanz der Merkmale bezüglich der Aktivitätsklassen aus Beispieldatensätzen bestimmt [Guyon 03].
Grundsätzlich kann die Relevanz eines Merkmals entweder ausschließlich auf Basis der Beispieldaten oder unter zusätzlicher Verwendung eines Klassifikators erfolgen. Verfahren, die ausschließlich auf statistischer Analyse beruhen, werden *Filtermethoden*, Methoden auf Basis eines konkreten Klassifikators *Einschlussmethoden* (engl.: *Wrapper methods*) genannt [Kohavi 97]. Wird eine Einschlussmethode verwendet, ist allerdings das Ergebnis stark abhängig von dem konkret verwendeten Klassifikator und seinen Parametern. Dies ist für ein allgemeines Relevanzmaß nicht erwünscht.
Filtermethoden verwenden statistische Verfahren, um die Relevanz eines Merkmals zu bestimmen. Dazu gehört vor allem die Berechnung der *Transinformation* (engl.: *Mutual Information, Information Gain*).

Die Transinformation ist ein Maß für die Information, die zwei stochastische Variablen teilen; es gibt also die Überlappung zweier Variablen wieder. Die Transinformation I zwischen zwei stochastischen Variablen X und Y wird über die Entropie definiert:

$$I(X,Y) = H(X) - H(X|Y) \tag{7.13}$$

$H(X|Y)$ stellt die bedingte Entropie von X dar, wenn Y gegeben ist, also die Restentropie von X für einen gegebenen Wert von Y. Durch Substitution der Entropie erhält man für die Berechnung von $I(X,Y)$

$$I(X,Y) = \sum_{x,y} P(X=i, Y=j) \log_2 \frac{P(X=i, Y=j)}{P(X=i) \cdot P(Y=j)}. \tag{7.14}$$

Mit diesem Maß können Merkmale bezüglich einer Klasse und untereinander bewertet werden [Long 05]. Auf Basis dieser Bewertung wird eine Auswahl von relevanten Merkmalen getroffen, die zur Klassifikation einer Aktivität verwendet werden. Dabei ist das Ziel, eine minimale Menge von Merkmalen zu finden, die möglichst hohe Klassifikationsgenauigkeit bringt. Da sich diese Ziele im Allgemeinen widersprechen, wird dies umformuliert: Das Ziel der Merkmalsauswahl ist es, eine möglichst kleine Menge von Merkmalen zu finden, so dass die Genauigkeit der Klassifikation eine definierte Mindestanforderung erfüllt.
Im allgemeinen Fall ist dieses Problem NP-hart, wodurch als einziger sicherer Weg die kombinatorische Suche verwendet werden kann. Dies ist allerdings zu aufwendig, da zur Bewertung aller Merkmalskombinationen mit k aus einer Menge von n Merkmalen $\binom{n}{k}$ Bewertungen nötig sind, was für Merkmalsräume üblicher Dimension zu groß wird.
Deshalb wird an dieser Stelle eine Vereinfachung getroffen, indem einmal gewählte Merkmale ausgewählt bleiben (engl.: *Greedy Selection*). Stark relevante Merkmale können so korrekt gefunden werden, während schwach relevante Merkmale eventuell nicht optimal ausgewählt werden.

Mit dieser Vereinfachung und dem Maß der Transinformation kann das Kriterium zur Auswahl von Merkmalen nach [Battiti 94] formuliert werden:

> Bei gegebener Merkmalsmenge F mit n Merkmalen soll die Untermenge $S \subset F$ mit k Merkmalen gefunden werden, die $H(C|S)$ minimiert und damit $I(C,S)$ maximiert.

Mit Gleichung (7.14) kann $I(C,S)$ berechnet werden. Mit einem Gewichtungsfaktor β kann dann nach Kwak und Choi [Kwak 99] das Maß für die Relevanz R eines Merkmals f_i bezüglich einer Klasse C bei einer bereits ausgewählten Merkmalsmenge $\bar{S}$ formuliert werden:

$$R(f_i) = I(C; f_i) - \beta \sum_{f_s \in \bar{S}} \left(\frac{I(f_s; C)}{H(f_s)} I(f_s; f_i) \right). \tag{7.15}$$

Mit diesem Maß werden die Merkmale für die Klassifikation von Aktivitäten bewertet.

7.3.4 Klassifikator

Zur Klassifikation werden zwei Schritte benötigt: Zuerst wird die Menge der n relevantesten Merkmale ausgewählt. Mit diesen Merkmalen wird der Klassifikator für jede Aktivität trainiert und abgefragt.
Für jede zu klassifizierende Aktivität wird dabei ein eigener Klassifikator aufgebaut. Dies hat zwei Gründe:

- Die Menge der aktiven Klassifikatoren kann dynamisch verändert werden, es können also Klassifikatoren einzeln verwendet werden. Dies ist unmöglich, wenn zur Unterscheidung der trainierten Aktivitäten ein gemeinsamer Klassifikator verwendet wird.
- Die Menge der relevanten Merkmale ist abhängig von der zu klassifizierenden Aktivität. Bei der Verwendung eines Klassifikators pro Aktivität kann dieses Wissen explizit genutzt werden, indem dem Klassifikator nur die n relevantesten Merkmale zugeführt werden.

Zur Klassifikation werden Neuronale Netze vom Typ *feedforward* verwendet. Jedes zur Klassifikation einer Aktivität verwendete Neuronale Netz besteht aus drei Schichten mit

- n Eingabeneuronen, wobei n die Anzahl der verwendeten Merkmale darstellt. Typische Werte sind $3 < n < 10$, wie sich durch die Auswertung zeigt. Dies ist in Kapitel 8.4.2 beschrieben.
- einen Ausgabeneuron für einen Schätzwert für die Zugehörigkeit der momentanen Eingabedaten zur zugehörigen Aktivität c_i.
- 10 Neuronen in der Zwischenschicht. Dies ist ein experimentell ermittelter Wert, der auf dem Erfahrungswert $n \leq 10$ beruht. Eine weitere Reduktion der Neuronenanzahl in der Zwischenschicht reduziert die Erkennungsrate, während eine höhere Anzahl wenig Effekt aufzeigt.

Als Transferfunktion zwischen der ersten und der zweiten Schicht wird die klassische *tansig* Funktion verwendet:

$$tansig(n) = \frac{2}{1 + e^{-2n}} - 1 \tag{7.16}$$

Diese Funktion stellt eine Abbildung von $\mathbb{R}$ auf $[-1, 1]$ dar. Die Transferfunktion zwischen der zweiten und dritten Ebene wird durch die *logsig* Funktion gebildet:

$$logsig(n) = \frac{1}{1 + e^{-n}} \tag{7.17}$$

Die *logsig* Funktion besitzt einen Wertebereich $[0, 1]$. Dies lässt die Interpretation des Funktionswerts als Wahrscheinlichkeit zu.
Die Verwendung Neuronaler Netze soll dabei exemplarisch für einen geeigneten Klassifikator stehen und stellt nicht notwendigerweise die beste Lösung an dieser Stelle dar. Die Auswertung (siehe Kapitel 8.4) zeigt allerdings, dass Neuronale Netze für die gegebene Aufgabe hinreichend gut sind, so dass sich die Notwendigkeit einer vergleichenden Betrachtung verschiedener Klassifikatoren im Rahmen dieser Arbeit nicht ergibt. Dennoch sollte dies Ziel weiterer Untersuchungen sein; siehe auch Kapitel 9.

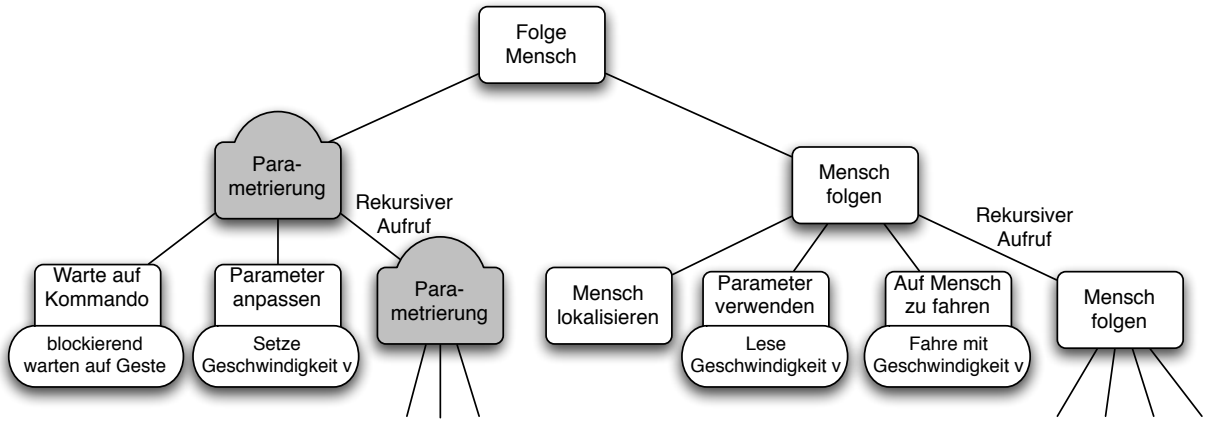


Abbildung 7.15: Beispiel einer Parameteradaption durch eine Geste des Benutzers. Dieses Programm läuft durch den rekursiven Aufruf in einer Endlosschleife, wobei in jedem Aufruf ein Parameter neu gelesen wird, der durch den asynchronen Zweig gesetzt werden kann. Dieser befindet sich genauso in einer Endlosschleife.

7.4 Adaption von Roboterprogrammen

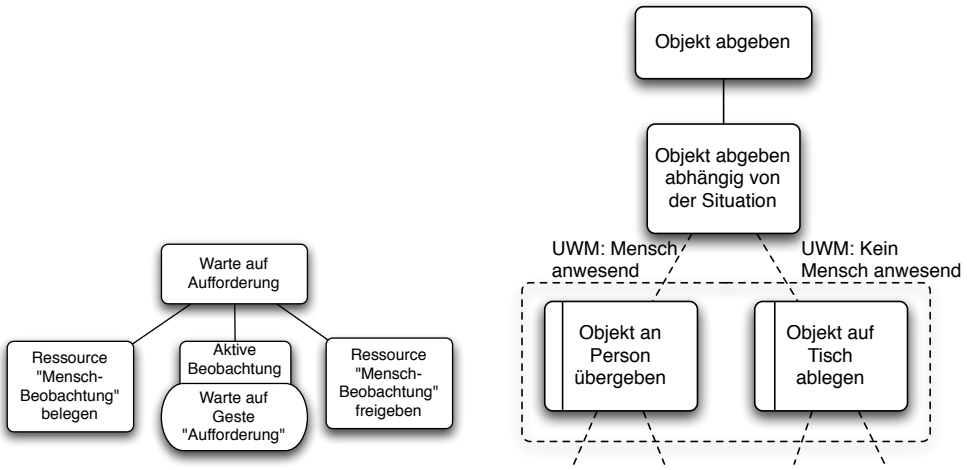
An dieser Stelle stehen nun die Möglichkeiten zur Verfügung, bestehende Roboterprogramme über die beschriebenen Interaktionskanäle zu parametrieren. Dazu werden in einem Flexiblen Programm synchrone oder asynchrone Teilhandlungen eingefügt, die auf Kommandos und Kommentare des Benutzers reagieren und interne Verarbeitungsparameter dementsprechend verändern können. Dies kann auf zwei verschiedene Weisen geschehen:

- Die Interaktionskomponenten werden von einem Flexible Programm als Ressourcen belegt und mit im FP bestimmten Aufgaben belegt. Die Rückgabe der Komponenten wird ausgewertet, und der weitere Verlauf der Handlung des FPs hängt von dieser Auswertung ab. Die Interaktionskomponenten können dafür parametriert und gesteuert werden; beispielsweise kann der Kamerakopf des Roboters aktiv die Bewegungen des Menschen verfolgen, und der berührungssensitive Bildschirm kann mit Anfragen etc. (siehe Kapitel 7.1.3) belegt werden.

- Ist die Ressource zur Beobachtung und Interpretation der Bewegungen des Menschen nicht anderweitig von FPs belegt, so legt diese alle momentan erkannten Aktivitäten im Umweltmodell ab. Dieses Umweltmodell ist allen FPs global verfügbar (siehe Kapitel 4.1.2), so dass die Repräsentation von Bedingungen und Alternativen genutzt werden kann, um aus verschiedenen Varianten eines FPs auszuwählen.

Im Folgenden werden direkte Kommandos sowie Parametrierungen durch den Benutzer betrachtet. Dazu sei das Beispiel aus Abbildung 7.15 herangezogen. Am Anfang eines Flexiblen Programms wird ein asynchroner Zweig eingefügt. Dieser wird von der Verarbeitung betreten, ist jedoch nicht relevant für die Fertigstellung der Handlung. In diesem Zweig kann nun auf ein externes Ereignis gewartet werden, das als Auslöser für weitere Teilhandlungen dient. Im Beispiel ist dies eine Geste zur Veränderung der Geschwindigkeit; diese kann durch die Gesten- und Aktivitätenklassifikation erkannt werden.

Die direkte Kommandierung, also das Auslösen der Ausführung eines FPs durch den Benutzer, kann analog aufgebaut werden. Dazu sind keine asynchronen Handlungszweige notwendig; das blockierende Warten auf ein Kommando wird vor der eigentlichen Handlung im Baum eingefügt, womit die Handlung durch das entsprechende Ereignis ausgelöst wird. Dies gilt ähnlich für die Auswahl von Handlungsalternativen auf Basis der momentanen Situation. Abbildung 7.16 zeigt zwei Beispiele dazu. Entweder wird die Beobachtungskomponente aktiv von einem FP genutzt, um definierte Gesten oder Aktivitäten abzuwarten, oder es wird auf Basis der momentanen Situation zwischen mehreren Handlungsalternativen entschieden. Die momentane Situation wird im aufgezeigten Beispiel durch die Komponente zur Beobachtung und Interpretation menschlicher Bewegungen bestimmt.



(a) (b)

Abbildung 7.16: Beispiele für Flexible Programme mit Einbindung der Interaktionskomponenten. Aktive Beobachtung des Menschen und warten auf eine bestimmte Geste (a). Entscheidung zwischen Handlungsalternativen auf Basis der beobachteten Situation (b).

In Abbildung 7.16 rechts ist besonders die hierarchische Staffelung der Auswahl von Alternativen zu beachten. Wie in Kapitel 6.3 beschrieben, wird die Auswahl zwischen mehreren Kandidaten genau dann getroffen, wenn der Elternknoten betreten wird. Die Wurzel der gezeigten Teilhandlung (*Objekt abgeben*) wird in einem größeren FP gleichzeitig mit allen weiteren Geschwisterknoten auf der gleichen Ebene betreten; dies entspricht nicht unbedingt dem Zeitpunkt der Ausführung der gezeigten Teilhandlung. Um diese Auswahl zum Zeitpunkt der eigentlichen Ausführung zu garantieren, wird eine Hierarchieebene zwischen *Objekt abgeben* und den Alternativen *Objekt an Person übergeben* und *Objekt auf Tisch ablegen* eingeführt. Dies bewirkt, dass die Wurzel dieser Teilhandlung frei in einem größeren Handlungsbaum platziert werden kann.

7.5 Zusammenfassung

In diesem Kapitel wurden die zur Interaktion zwischen Roboter und Mensch entworfenen Verfahren und Komponenten vorgestellt. Dies beinhaltet die Darstellung des internen Verarbeitungszustands sowie die dafür verwendete Modellierung. Diese Zustandsmodellierung

orientiert sich nicht am oft in der Literatur verwendeten Emotionsraum, sondern versucht vielmehr, die Zustände des Roboters als die eines technischen Systems mit intuitiv erfassbaren Mitteln darzustellen. Es wird unterschieden zwischen Zustandsübergängen und Visualisierung; dadurch wird es ermöglicht, Visualisierungen auszutauschen und auf diese Weise an den jeweiligen Menschen anzupassen.
Des Weiteren wurden Verfahren zur Beobachtung und Interpretation menschlicher Bewegung allein mit Robotersensorik vorgestellt. Die Beobachtung geschieht modellbasiert unter Verwendung der in Kapitel 4 vorgestellten Modelle. Das vorgestellte Verfahren verwendet einen erweiterten Nächste-Punkte-Algorithmus und verfolgt eine Hypothese der Körperpose über die Zeit. Verschiedene Messungen des zu verfolgenden Körpers können über die inverse Messgleichung direkt im Raum des Körpermodells fusioniert werden, wodurch aufwendige Berechnungen und Transformationen in höherdimensionale Räume wegfallen, wie dies beispielsweise bei Partikelfiltern der Fall ist. Die so beobachteten Bewegungen werden interpretiert und klassifiziert als Gesten und Aktivitäten, wobei zur Klassifikation eine Menge an Merkmalen aus der Bewegung extrahiert und einem Klassifikator zu Training sowie Erkennung zugeführt wird.
Die vorgestellten Verfahren zur Interaktion können zur Adaption von Parametern in Roboterprogrammen genauso wie zur direkten Kommandierung genutzt werden. Dazu werden vom Roboter die erkannten Gesten und Aktivitäten als Kommando bzw. Kommentar interpretiert und das ausgeführte Programm entsprechend adaptiert.

Kapitel 8

Experimentelle Evaluierung

Nachdem in den vorangegangenen Kapiteln die Ansätze zur Erstellung, Ausführung und Adaption von Handlungswissen erläutert wurden, sollen diese in diesem Kapitel validiert und evaluiert werden. Dazu werden die einzelnen Aspekte näher betrachtet und die Leistungsfähigkeit der Verfahren aufgezeigt.
Zunächst werden Ergebnisse und Beispiele zur Erzeugung von Handlungswissen aus Makrooperatoren vorgestellt. Die Ausführung Flexibler Programme sowohl in Simulationsumgebungen als auch mit dem Roboter wird beleuchtet und bewertet. Des Weiteren wird die Verfolgung und Interpretation menschlicher Bewegungen bewertet sowie das Zusammenspiel aller dieser Aspekte für die Adaption von existierendem Handlungswissen.

8.1 Erzeugung von Handlungswissen

Zwei verschiedene Arten von Experimenten sollen dazu dienen, den Ansatz zur Erzeugung von Handlungswissen aus abstrakten Makrooperatoren zu evaluieren. Zuerst wird der Abbildungsprozess selbst betrachtet. Dazu werden unterschiedlich komplexe Makrooperatoren verarbeitet, um die Performanz und den Rechenaufwand der vorgestellten Methodik zu bewerten. Des Weiteren werden die erzeugten Roboterprogramme in einer Simulationsumgebung mit dem Roboter ausgeführt, um Korrektheit und Vollständigkeit des resultierenden Flexiblen Programms zu bewerten.

8.1.1 Bewertung des Abbildungsprozesses

Um die Leistungsfähigkeit des Abbildungsprozesses selbst zu bewerten, wird eine Menge von 20 Makrooperatoren als Testdaten betrachtet. Diese werden einzeln auf Flexible Programme abgebildet.
Der Abbildungsprozess und das Ergebnis der Abbildung hängen wesentlich vom Aufbau des Makrooperators ab. Dies liegt in erster Linie daran, dass die Anwendung von Regeln zur Modifikation des Handlungswissens während des Abbildungsprozesses (siehe Kapitel 5.5) sehr stark von Aufbau und Inhalt des Makrooperators bestimmt wird. Aus diesem Grund wurden zur Evaluation Makrooperatoren mit unterschiedlichen Inhalten gewählt, die sowohl einfach als auch komplexe beobachtete Handlungen beinhalten. Die Komplexität der Übersetzung wurde in Kapitel 5.8 als linear nachgewiesen. Dies wird sich im Folgenden experimentell bestätigen (siehe auch [Knoop 07a]).

Name des MOs	# Knoten im MO	# Knoten im FP	Rechendauer/sek
Approach	4	19	0.36
Transport	5	30	0.48
Approach	6	21	0.45
Retreat	9	26	0.62
Pick	19	39	1.06
Place	21	42	1.07
Place	24	45	1.16
Pick	27	47	1.25
Pick and place	38	106	2.48
Pick and place	39	117	2.63
Pick and place	40	118	2.64
Pick	42	62	1.84
Place	54	75	2.25
Pick and place	56	124	2.96
Pick and place	57	135	3.16
Pick and place	58	126	3.01
Lay table 3 Objekte	136	389	9.26
Lay table 3 Objekte	139	392	8.95
Lay table 3 Objekte	149	402	9.62
Lay table 6 Objekte	228	721	19.52

Tabelle 8.1: Ergebnis der Abbildung für 20 unterschiedliche Makrooperatoren (MOs), geordnet nach der Anzahl der Knoten im Makrooperator. Man erkennt, dass ähnliche Aufgaben mit unterschiedlicher Anzahl von Knoten existieren; dies liegt an der Segmentierung der Beobachtung in sinnvolle Elementaroperatoren. Die Operatoren beinhalten verschiedene Handlungen wie Aufnehmen und Ablegen von Objekten (engl.: *Pick and Place*), Anrücken (engl.: *Approach*), Abrücken (engl.: *Retreat*), und Transport. Eine komplexe Handlung ist Tisch decken (engl.: *Lay Table*).

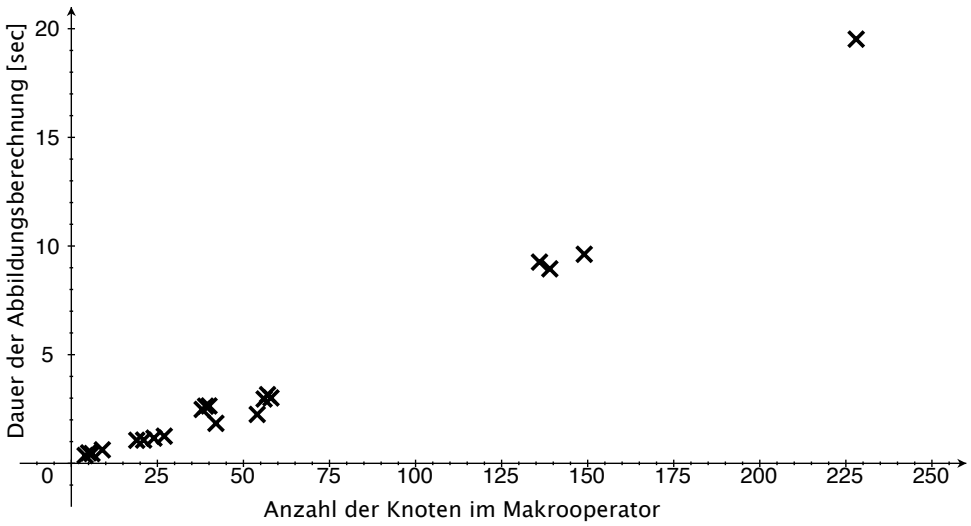


Abbildung 8.1: Berechnungsdauer der Abbildung über der Anzahl der Knoten im Makrooperator. Der Zusammenhang erweist sich als linear, wie in Kapitel 5.8 postuliert.

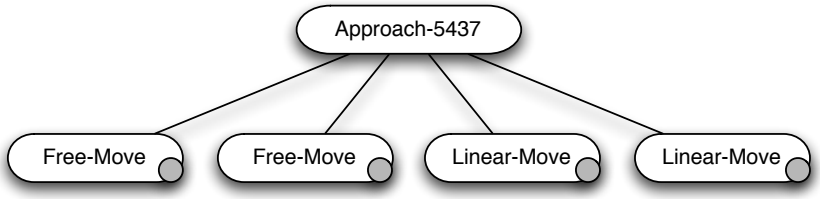


Abbildung 8.2: Makrooperator als Beispiel für die Abbildung. Dieser MO enthält 5 Knoten; mit einem Punkt markierte Knoten enthalten Parameter aus der Vorführung. Die Anzahl der Knoten ist abhängig von der Trajektorie und der Segmentierung.

Tabelle 8.1 gibt die Ergebnisse der Auswertung wieder. Abbildung 8.1 visualisiert noch einmal den Zusammenhang zwischen der Anzahl der Knoten im Makrooperator (zweite Spalte in Tabelle 8.1) und der resultierenden Rechendauer (4. Spalte). Aus Tabelle 8.1 lässt sich erkennen, dass der Zusammenhang zwischen Übersetzungszeit und Anzahl der abzubildenden Knoten tatsächlich linear ist. Des Weiteren ist zu erkennen, dass die Anzahl der Knoten in der Handlungsbeschreibung für den Roboter wesentlich höher ist als die Anzahl der Knoten im Makrooperator. Dies ist damit zu erklären, dass die in Kapitel 5.5 vorgestellten Regeln zur Modifikation roboter- und handlungsspezifisches zusätzliches Wissen einbringen. Die hohe Variabilität zeigt sich auch im Quotienten zwischen der Anzahl der Knoten in FP und MO: Dieser liegt für die 20 vorgestellten Datensätze zwischen 1.3 und 6.0, und beträgt im Mittel 2.85.

Zum Vergleich zwischen MO und resultierendem FP sowie als Beispiel für den Abbildungsprozess soll der Makrooperator in Abbildung 8.2 dienen. Dieser Makrooperator beinhaltet beobachtete und segmentierte Bewegungen der Hand des Demonstrators und besteht aus vier sequentiell geordneten Bewegungen der Typen *Freie Bewegung* (engl.: *Free-Move*) und *Lineare Bewegung* (engl.: *Linear-Move*). Diese vier Bewegungen sind in einem Operator *Anrücken*

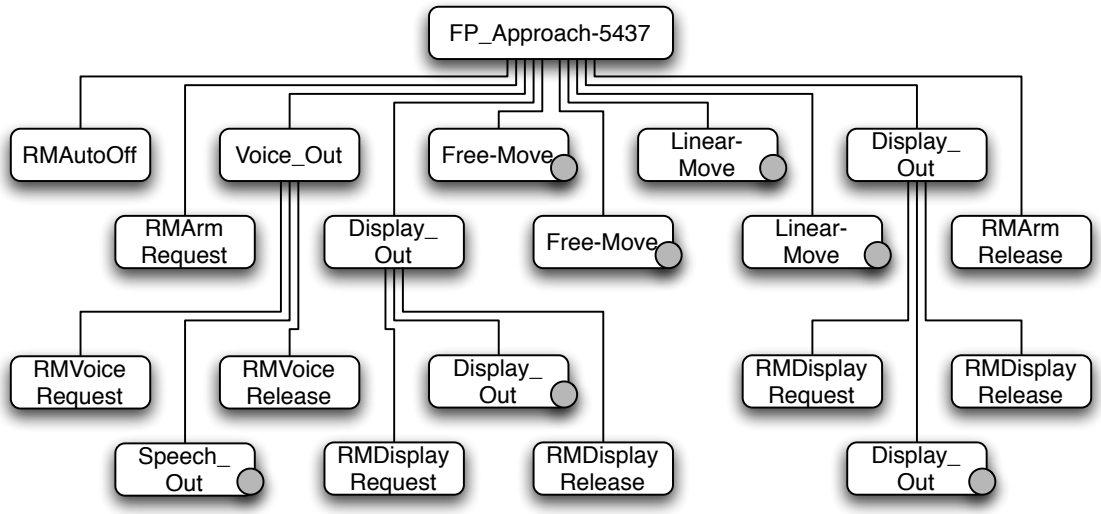


Abbildung 8.3: Aus dem Makrooperator in Abbildung 8.2 resultierendes Flexibles Programm. Es enthält 20 Knoten, und wiederum beinhalten die mit einem Punkt markierten Knoten Parameter für die Ausführung. Dies können Trajektorien, Position oder auch Text sein.

(engl.: *Approach*) zusammengefasst. Dieser stammt beispielsweise aus einer Handlung zum Aufnehmen eines Objekts (engl.: *Pick*).
Dieser einfache Makrooperator wird nun auf ein Flexibles Programm abgebildet, dieses ist in Abbildung 8.3 dargestellt. Die Bewegungsoperationen aus dem Makrooperator sind deutlich zu erkennen. Zusätzliche Teilhandlungen betreffen die Ausgabe von Zuständen über die Visualisierung und die Sprachausgabe (*Display_Out* und *Voice_Out*), sowie das Ressourcenmanagement (Belegen von Ressourcen als *Request*, Freigabe mit *Release*). Das erste Blatt des Handlungsbaums beinhaltet einen Steuerknoten, der das Verhalten zum automatischen Belegen und Freigeben von Ressourcen abschaltet. Dies kann optional durch den Interpreter des Flexiblen Programms geschehen, ist aber selten erwünscht.
Aus den genannten Gründen enthält das resultierende Flexible Programm wesentlich mehr Knoten als die ursprüngliche Beschreibung im Makrooperator. Sowohl die Breite als auch die Tiefe des Handlungsbaums sind durch die Regelanwendung erweitert worden. Dies geschieht während der Abbildung besonders dann, wenn der Makrooperator viele parallele Aktionen beinhaltet.
An dieser Stelle ist der Unterschied zwischen abstraktem und konkretem Handlungswissen gut erkennbar. Der Makrooperator enthält abstrakte Handlungsanweisungen, die nicht auf ein Ausführungssystem zugeschnitten sind. Das Flexible Programm dagegen beinhaltet die Handlungsbeschreibung für einen dedizierten Roboter mit bekannter Handlungsbeschreibungssprache und bekanntem Rahmenwerk.

8.1.2 Bewertung der erzeugten Flexiblen Programme

Um das resultierende Flexible Programm auf Korrektheit und Vollständigkeit zu überprüfen, muss es mit dem Roboter ausgeführt werden. Dafür wird eine Simulationsumgebung verwendet, an die eine 3d-Darstellung des Roboters und seiner Umwelt angeschlossen ist. Die Simulationsumgebung ist in Abbildung 8.4 und 8.5 abgebildet.

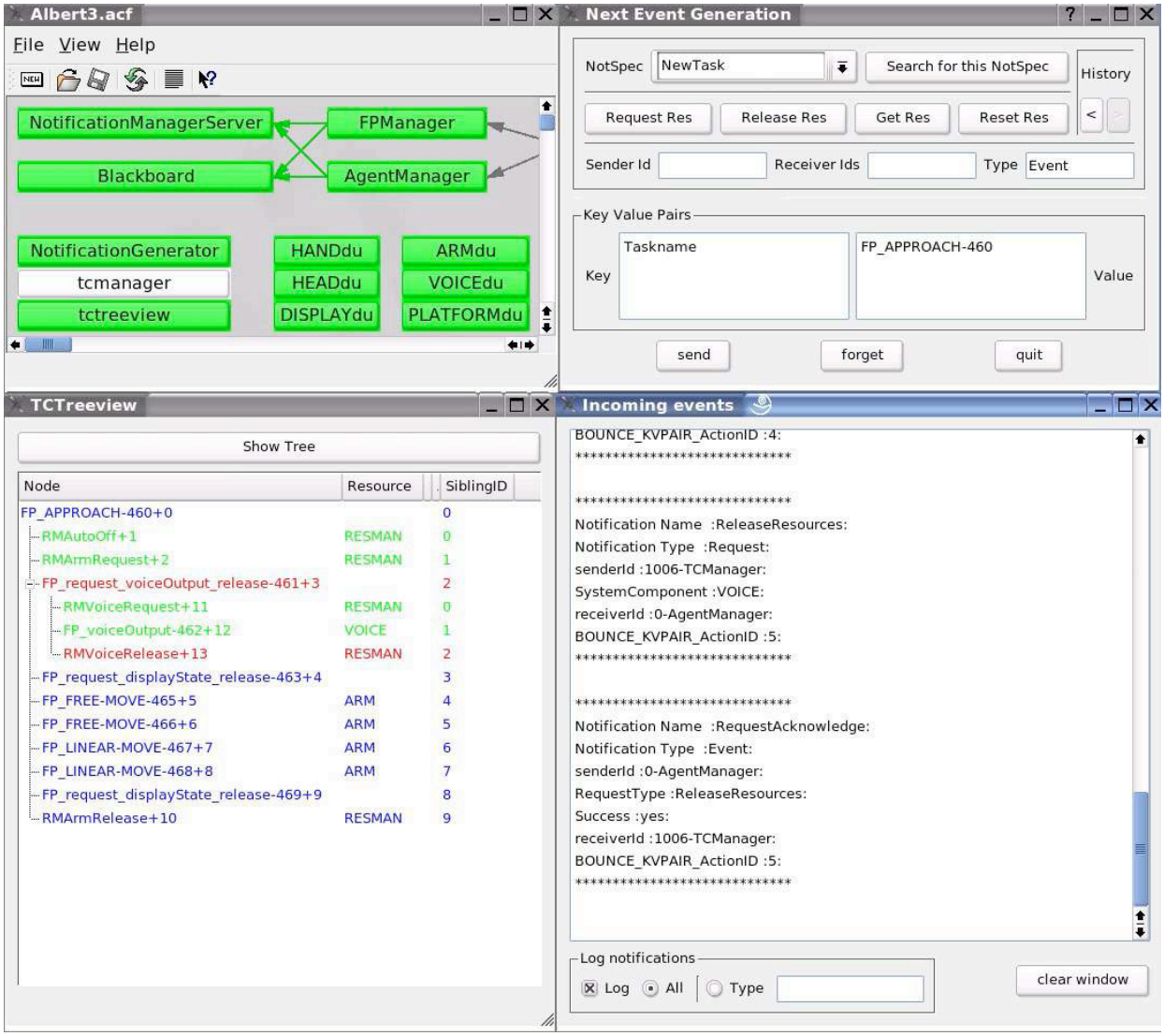


Abbildung 8.4: Steuerung und Simulation zur Validierung von Flexiblen Programmen. Links oben sind die gestarteten Komponenten zu sehen, rechts oben die Eingabemaske für die Ausführung einer Handlung, links unten ein Teilbaum der Handlung, und rechts unten die ereignisbasierte Kommunikation des Systems.

Das Flexible Programm *Lay Table 3 Objekte* wird mit dem vollständigen Ausführungsrahmen sowie der angeschlossenen Simulation und Visualisierung ausgeführt. Der Makrooperator für diese Handlung besteht aus 136 Knoten, das FP besitzt 389 Knoten. Abbildung 8.5 zeigt Bilder aus der Ausführung dieser Handlung.
Aus der erfolgreichen Ausführung können mehrere Schlüsse gezogen werden:

- Die Abbildung von abstraktem Handlungswissen auf dedizierte Roboterhandlungen ist möglich und durchführbar mit dem vorgestellten Ansatz.

- Die Transformation der Parameter von der Vorführung in die Ausführung muss noch weiter betrachtet werden. In Abbildung 8.5 folgt die Ausführung in großen Teilen der Trajektorie der Vorführung, was teilweise zu Kollisionen führen kann. Hier muss ein

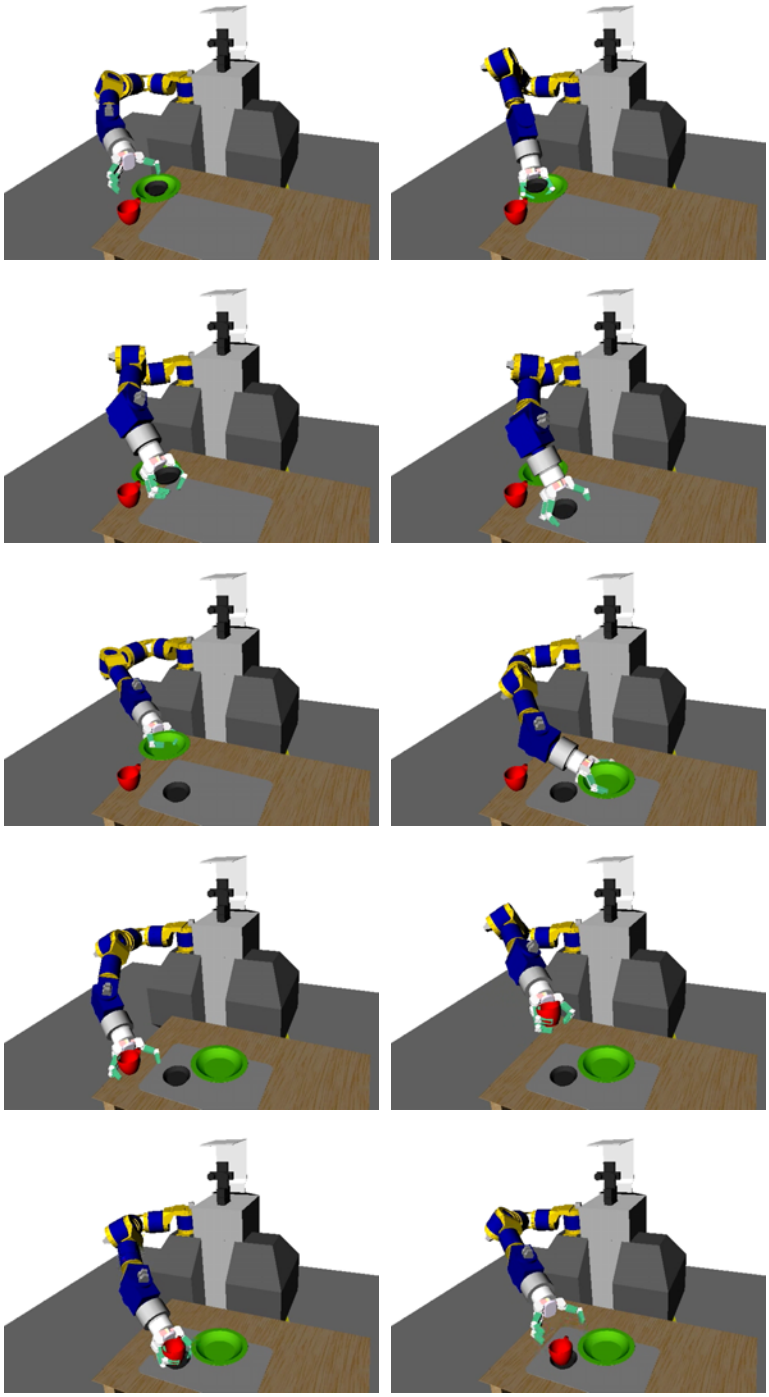


Abbildung 8.5: Ausführung des Flexiblen Programms *Lay Table 3 Objekte* in der Simulation. Der Tisch mit den 3 Objekten befindet sich im Vordergrund. Der Roboter greift zuerst die Untertasse und legt sie an der Zielposition ab (erste 4 Bilder). Dann wird der Teller bewegt (Bild 5 und 6). Zuletzt wird die Tasse auf der Untertasse platziert.

Weg gefunden werden, die gelernten Handlungssequenzen mit klassischen Trajektorienplanern für den Manipulator zu verknüpfen. Hier bieten sich beispielsweise Verfahren nach [Miller 03] für die Greifplanung und [Kramer 06] für die Pfadplanung an.

8.2 Ausführung Flexibler Roboterprogramme

Die in den vorangegangenen Kapiteln vorgestellten Komponenten zur Modellierung und Ausführung von Handlungswissen werden im Folgenden experimentell validiert. Dazu gehört die Überprüfung von Bedingungen auf lokalen und globalen Repräsentationen sowie die Ausführung selbst.

8.2.1 Überprüfung von Bedingungen

Die experimentelle Evaluierung gibt einerseits Aufschluss über die Funktionsweise, andererseits vor allem über benötigte Laufzeiten der Überprüfung. Der hierarchische Aufbau des Umweltmodells aus Abbildung 4.7 wird dabei nach Algorithmus 6.6 genutzt, um atomare Terme der Bedingungsausdrücke auszuwerten.
Programm 8.1 zeigt eine zu überprüfende Bedingung der Form

$$(\texttt{Cup}\ \textit{on}\ \texttt{Box2} \wedge \texttt{Cup/Capacity} \neq 0) \vee (\texttt{Cup}\ \textit{on}\ \texttt{Box1} \wedge \texttt{Cup/Color} = \texttt{red} \wedge \texttt{Cup/Capacity} = 42). \quad (8.1)$$

Der Ablauf der Überprüfung ist in Abbildung 8.6 abgebildet. Zu erkennen ist, dass die Bedingung `Cup/Capacity` $\neq 0$ nicht weiter überprüft wird, da sich der Ausdruck `Cup` *on* `Box2` bereits als FALSCH erwiesen hat und damit die Konjunktion mit FALSCH belegt werden kann.

Von besonderem Interesse ist die Bewertung der Laufzeiten der Überprüfung. Die beteiligten Komponenten sind verteilt und kommunizieren über die in Kapitel 4.1.2 vorgestellten Mechanismen. Zusätzlich findet die Überprüfung geometrischer Bedingungen in der Simulationsumgebung statt. Da die Überprüfung in vielen Situationen zeitkritisch erfolgt, wird dieses untersucht und bewertet.
Der Hauptfaktor für die Dauer der Überprüfung eines Terms liegt in der Anzahl der enthaltenen Atome. Daraus folgt direkt, dass die Überprüfung der quantifizierte Terme $\forall, \exists$ die Laufzeit stark beeinflussen können, da hier mitunter eine große Menge an Termen betrachtet und evaluiert werden muss.
Für die Evaluierung wurden die einzelnen Schichten des Weltmodells auf ihren Einfluss auf die Laufzeit der Überprüfung hin untersucht. Diese Schichten sind die *lokale Repräsentation*, das *globale Blackboard*, sowie die *globale Simulationsumgebung*.
Tabelle 8.2 gibt die Ergebnisse dieser Auswertung wieder. Dabei zeigt sich, dass die Überprüfung von atomaren Termen sowohl im lokalen als auch im globalen assoziativen Speicher (das Blackboard) unkritisch ist. Die Überprüfung von mehreren zehntausend Termen pro Sekunde ist hinreichend schnell. Die Überprüfung von geometrischen Relationen in der Simulationsumgebung ist mit 360 Atomen pro Sekunde wesentlich aufwendiger, aber trotzdem verwendbar. Nichtsdestotrotz sollte bei der Formulierung von Bedingungstermen daran gedacht werden, dass eine geometrische Überprüfung den Aufwand von ~ 100 Gleichheitsbe-

```
<context-list>
<or>
  <and>
    <on>
      <arg const="True">Cup</arg>
      <arg const="True">Box2</arg>
    </on>
    <not>
      <equal>
        <arg type="Long">Cup/capacity</arg>
        <arg type="Long" const="True">0</arg>
      </equal>
    </not>
  </and>
  <and>
    <on>
      <arg const="True">Cup</arg>
      <arg const="True">Box1</arg>
    </on>
    <equal>
      <arg>Cup/color</arg>
      <arg const="True">red</arg>
    </equal>
    <equal>
      <arg type="Double">Cup/capacity</arg>
      <arg type="Double" const="True">42.0</arg>
    </equal>
  </and>
</or>
</context-list>
```

Programm 8.1: XML-Darstellung für eine zu überprüfende Bedingung

Datenquelle	Überprüfte Atome	Gesamtlaufzeit	Atome pro Sek.
Lokaler Speicher	100000	ca. 3000ms	ca. 33000
Blackboard	100000	ca. 3200ms	ca. 31000
Simulationsumgebung	1000	ca. 2800ms	ca. 360

Tabelle 8.2: Laufzeit bei der Überprüfung atomarer Bedingungen in unterschiedlichen Schichten des Weltmodells

Abbildung 8.6: Verlauf der Evaluierung der Bedingung aus Programm 8.1. Zu erkennen ist, dass vorzeitig bekannte Terme nicht bis zum Ende ausgewertet werden (Schritt 2 und 3). Der gesamte Ausdruck wird letztendlich in Schritt 8 mit WAHR bewertet. Die einzelnen Atome werden im lokalen und globalen Weltmodell überprüft.

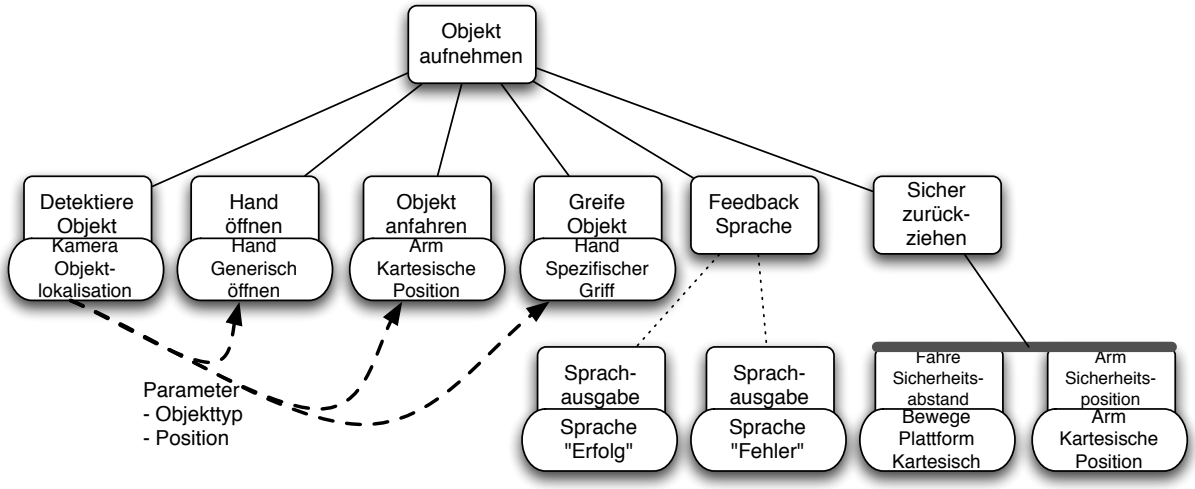


Abbildung 8.7: Flexibles Programm *Greife Objekt*. Gestrichelte Linien stellen Kandidatenbeziehungen dar; die parallele Gruppe ist durch den grauen Balken gekennzeichnet.

dingungen besitzt.

Die Verarbeitungsgeschwindigkeit ist also für reale Anwendungen hinreichend schnell. Allerdings sollte bei der Formulierung von komplexen Bedingungstermen darauf geachtet werden, dass quantifizierte Terme sorgsam verwendet werden, vor allem in Verbindung mit der Überprüfung geometrischer Relationen.

8.2.2 Ablauf der Ausführung

Der eigentliche Ablauf der Verarbeitung und Ausführung Flexibler Programme wird anhand von Beispielen experimentell evaluiert. Dazu soll zunächst das recht übersichtliche Beispiel in Abbildung 8.7 dienen (siehe auch [Knoop 06a]). Darin sind mehrere wichtige der vorgestellten Mechanismen vorhanden. Zum einen enthält es die Übergabe von Parametern von einem Blatt zum nächsten; zum anderen sind für einen Sitz in `Feedback Sprache` mehrere Kandidaten vorhanden, zwischen denen gewählt werden muss.
Die Ausführung des Flexiblen Programms ist in Abbildung 8.8 in mehreren Schritten abgebildet. Abhängig vom Ergebnis der Objektlokalisation (binär: gefunden/nicht gefunden) werden die nächsten Schritte durchgeführt. Insbesondere die Position des Objekts wird im lokalen Datenspeicher durch die Lokalisation abgelegt und von den weiteren Aktionen verwendet.
Abhängig von der Greifaktion (binär: gegriffen/nicht gegriffen) wird der Kandidat für die Sprachausgabe gewählt. Anschließend werden Plattform und Arm gleichzeitig in eine sichere Position gefahren. Dies ist für den Arm eine fest definierte Stellung, während die Plattform sich vom Tisch weg zum nächsten Knoten im topologischen Graphen bewegt. Diese Funktionalität ist in den Agenten vorhanden.

Ein komplexes Beispiel ist in Abbildung 8.9 dargestellt. Das Ziel des FPs ist das gleiche wie das des FPs in Abbildung 8.7, allerdings mit mehr Fehlertoleranz.

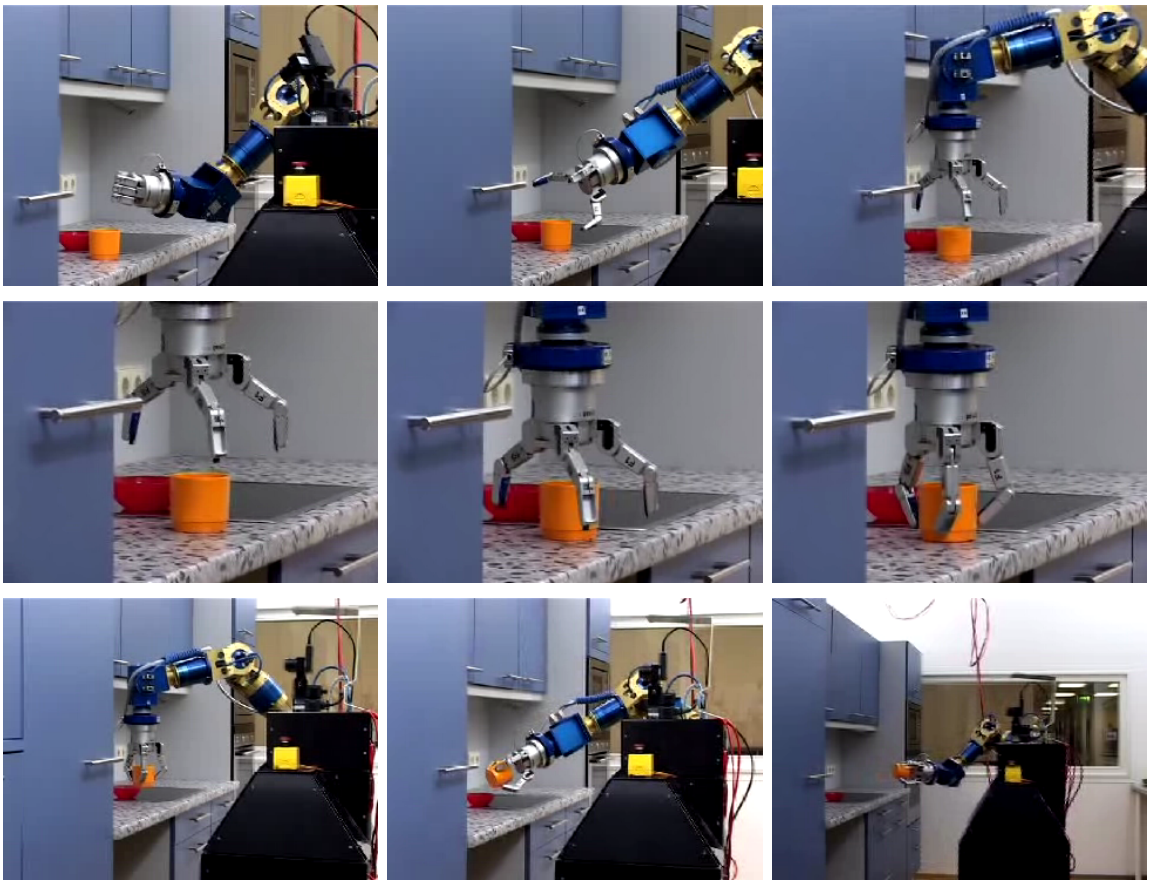


Abbildung 8.8: Die Einzelschritte zum Greifen eines Objekts in der Ausführung mit dem Roboter (von links nach rechts)

Die Wurzel der Handlung hat fünf Sitze. Diese fünf Sitze bestehen aus den Handlungen zum Auffinden des Tisches (1), an den Tisch fahren sowie Hand und Arm vorbereiten (2), greifen (3), abrücken bzw. erneut versuchen (4) und zur Ausgabe einer Sprachmeldung (5). Einige der Konstrukte dieser Handlung sollen nun näher beleuchtet werden.

- Der Knoten *Hand Arm vorbereiten* wird mehrfach verwendet. Dies ist möglich, weil ein Flexibles Programm in sich abgeschlossen und unabhängig ist, also nicht von seiner Position in einer übergeordneten Handlung abhängt. Die Handlung *Hand Arm vorbereiten* ist in Abbildung 8.9 aus Gründen der Übersichtlichkeit nicht doppelt dargestellt.

- Das gesamte Programm *Greife Tasse von Tisch* ruft sich selbst rekursiv auf. Dies geschieht, wenn der Greifvorgang nicht erfolgreich ausgeführt werden konnte und dies durch die Kraft- und Positionswerte des Greifers festgestellt wird. Auf diese Weise wird das Programm so lange wiederholt, bis es erfolgreich ausgeführt wurde. Es ist sinnvoll, hier eine Abbruchbedingung (beispielsweise über einen Zähler der Anzahl der Versuche) einzuführen; dieses ist nicht dargestellt.

- Der zweite und vierte Sitz des Wurzelknotens beinhalten die Auswahl zwischen Kandidaten. Die Auswahl geschieht mittels der in Kapitel 8.2.1 vorgestellten Verfahren. Beispielsweise werden nach dem Greifversuch die Kraftwerte der Finger gespeichert und

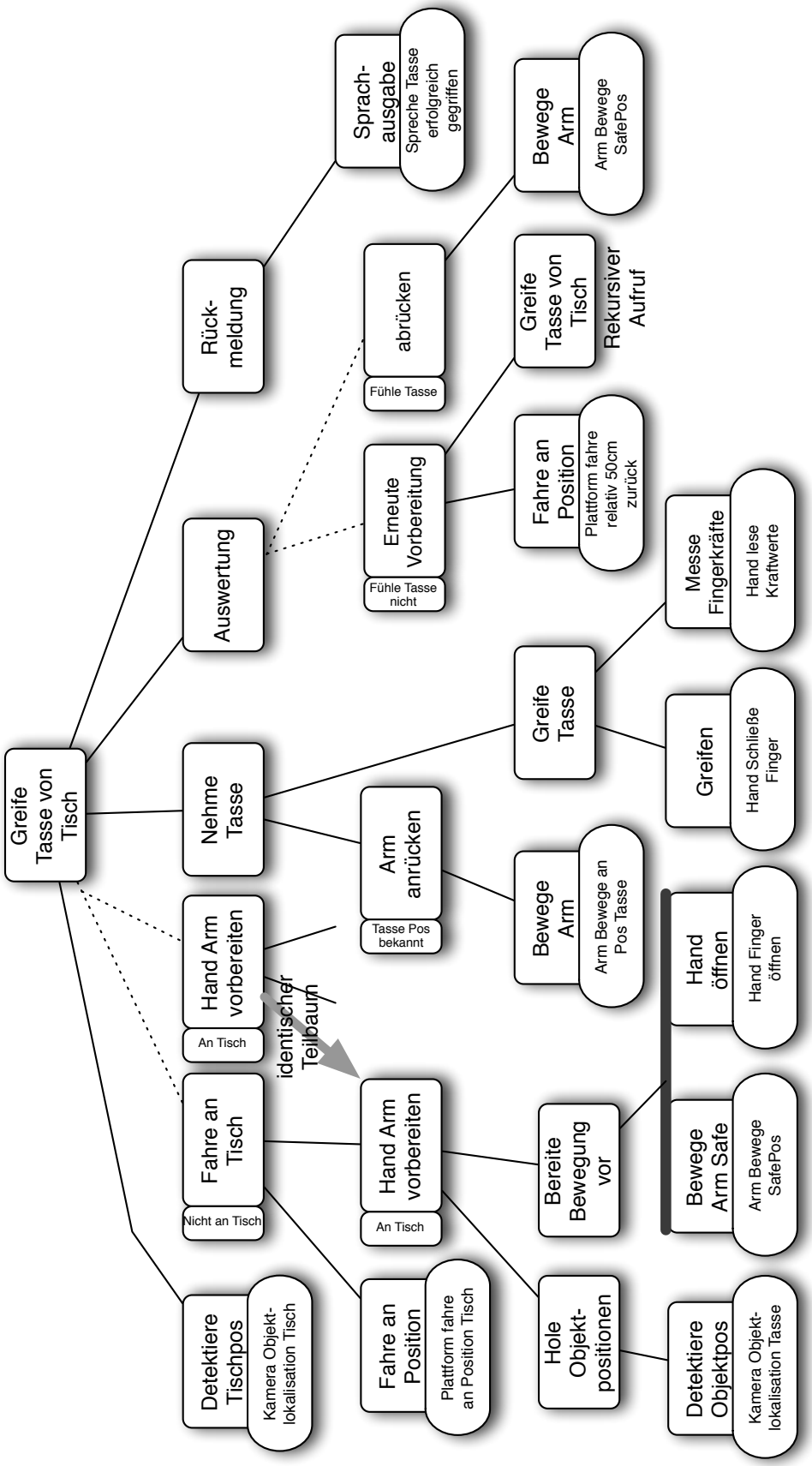


Abbildung 8.9: Schematische Darstellung eines komplexen Flexiblen Programms. Mehrere Mechanismen können erkannt werden: Teilprogramme werden wiederverwendet (siehe *Hand Arm vorbereiten*; die Handlung ist nur einmal vollständig dargestellt), rekursive Aufrufe (siehe Aufruf der Wurzel *Greife Tasse von Tisch*), und die Entscheidung zwischen mehreren Kandidaten abhängig vom Weltzustand. Des Weiteren werden die Knoten *Bewege Arm Safe* und *Hand öffnen* parallel ausgeführt.

später für die Auswahl zwischen den Kandidaten in *Auswertung* verwendet. Zustände, die für die Auswahl verwendet werden, werden in diesem Fall als Variablen zwischen verschiedenen Knoten des Baums ausgetauscht.

- Innerhalb des zweiten Sitzes des Wurzelknotens existiert eine parallele Gruppe mit den Knoten *Bewege Arm Safe* und *Hand öffnen*. Diese beiden Knoten werden also gleichzeitig angestoßen, und die parallele Gruppe wird genau dann als abgearbeitet (beendet) betrachtet, wenn beide Knoten beendet sind.

Der Ablauf bei der Verarbeitung des FPs aus Abbildung 8.9 ist in den Abbildungen 8.10 bis 8.13 dargestellt. Zunächst wird der erste Sitz des Wurzelknotens abgearbeitet, dann zwischen den möglichen Kandidaten für den zweiten Sitz ausgewählt. Der ausgewählte Kandidat wird instantiiert und seine Kindknoten den Vorgaben nach ausgeführt (Abbildung 8.10). Im vorliegenden Beispiel sind dies *Hole Objektpositionen* und *Bereite Bewegung vor*. Sind diese Teilhandlungen ausgeführt, wird das zweite Kind der Wurzel als beendet markiert und das dritte (*Nehme Tasse*) betreten. Dessen Kindknoten werden wiederum der Reihe nach ausgeführt (Abbildung 8.11). Ist dies geschehen, wird wiederum zwischen mehreren Kandidaten gewählt, um die weiteren Aktionen abhängig vom Ergebnis des Greifvorgangs zu gestalten. Im Beispiel ist dieser als erfolgreich bewertet, und das FP wählt den Kandidaten *abrücken* (Abbildung 8.12). Danach wird der letzte Kindknoten der Wurzel betreten und ausgeführt. Da dieser aus einer Sprachausgabe ohne Bedingungen besteht, gibt es hier keine Auswahlmöglichkeiten. Letztendlich wird der instantiierte und ausgeführte Handlungsbaum als vollständig abgearbeitet markiert (Abbildung 8.13).

An dieser Stelle kann der verarbeitete Baum verglichen werden mit dem ursprünglichen Handlungsbaum (Abbildung 8.9). Die Unterschiede zwischen einem Flexiblen Programm als Handlungsanweisung und einem ausgeführten Flexiblen Programm bestehen vor allem darin, dass der instantiierte Handlungsbaum keine Alternativen enthält, und diese auch nicht weiterhin vermerkt sind. Alternativen können sich nur in *nicht instantiierten* Handlungsbäumen befinden [Schmidt-Rohr 05].

8.3 Verfolgung menschlicher Bewegungen

Die Verfolgung menschlicher Bewegungen aus Robotersicht wird allein mit Robotersensorik durchgeführt. Die experimentelle Evaluation betrachtet verschiedene Aspekte: Zum einen wird die Fusion der unterschiedlichen Messverfahren und Sensordatenquellen betrachtet. Die Verfolgung wird mit unterschiedlichen Sensordaten evaluiert, um die Effektivität der Fusion zu bewerten. Andererseits wird die Bewegungsverfolgung mit allen verfügbaren Datenquellen als *Black Box* betrachtet und bewertet, um die Effizienz des des Gesamtsystems bewerten zu können, sowohl für die Verfolgung der Körperpose an sich als auch für die eigentliche Anwendung, die Interpretation der Bewegungen und Klassifikation von Aktivitäten. Alle Evaluationen werden grundsätzlich unter der Vorraussetzung der Echtzeitfähigkeit gemacht, um der Forderung nachzukommen, dass die Verfolgung und Interpretation grundsätzlich online erfolgen muss. Zunächst werden kurz die momentanen Einsatzszenarien diskutiert.

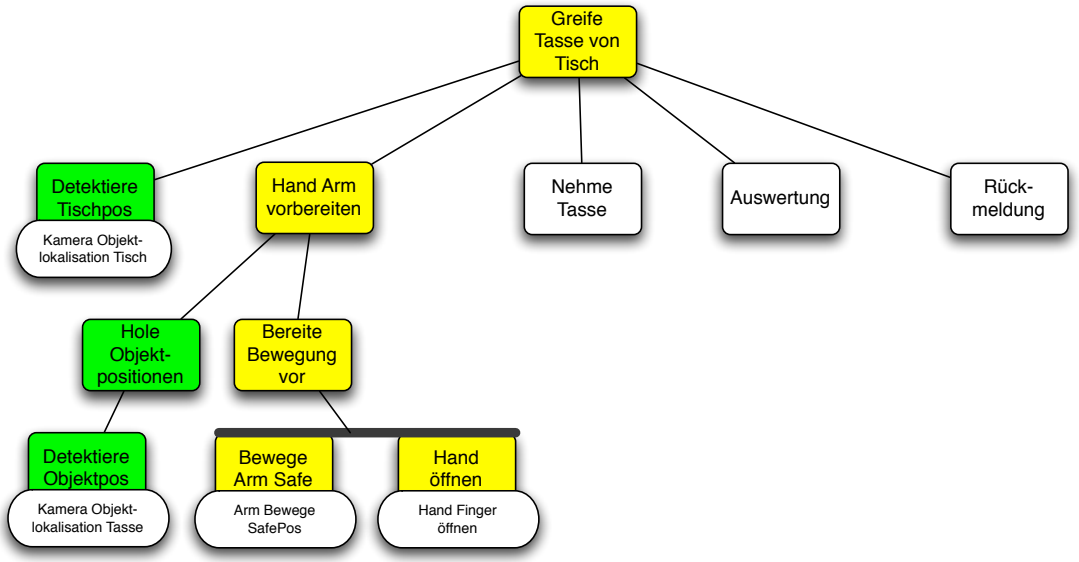


Abbildung 8.10: Ein möglicher Zwischenschritt bei der Abarbeitung des Flexiblen Programms aus Abbildung 8.9. Grüne Boxen bezeichnen abgearbeitete Teile, gelbe sind in Bearbeitung, und weiße Blöcke sind noch nicht besucht.

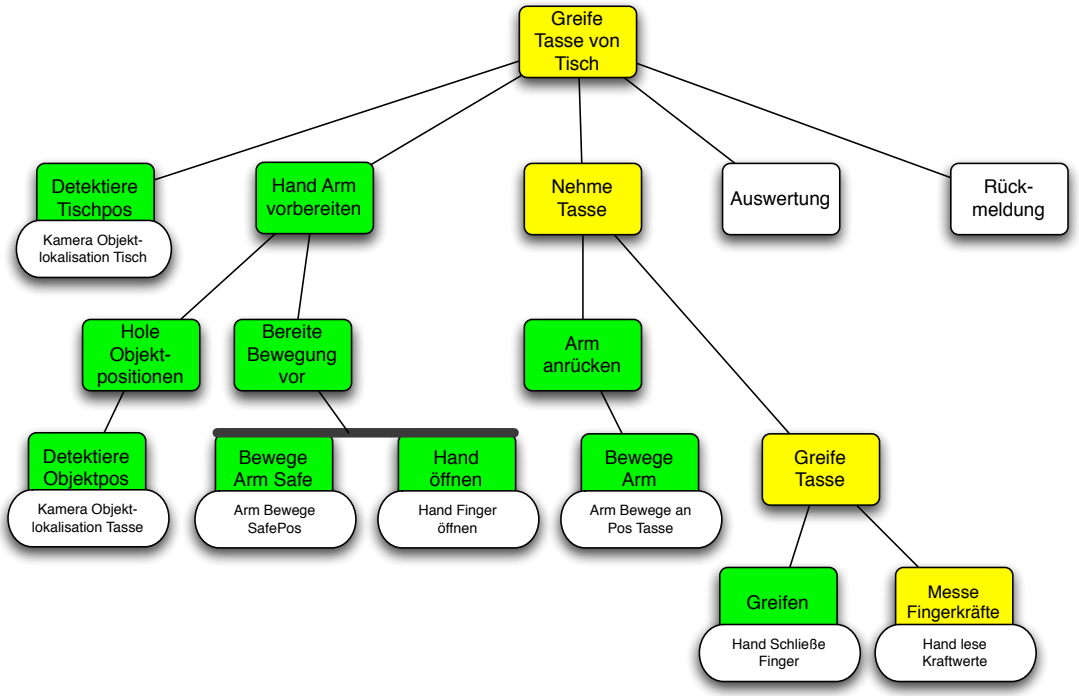


Abbildung 8.11: Weiterer Zwischenschritt bei der Bearbeitung des FPs.

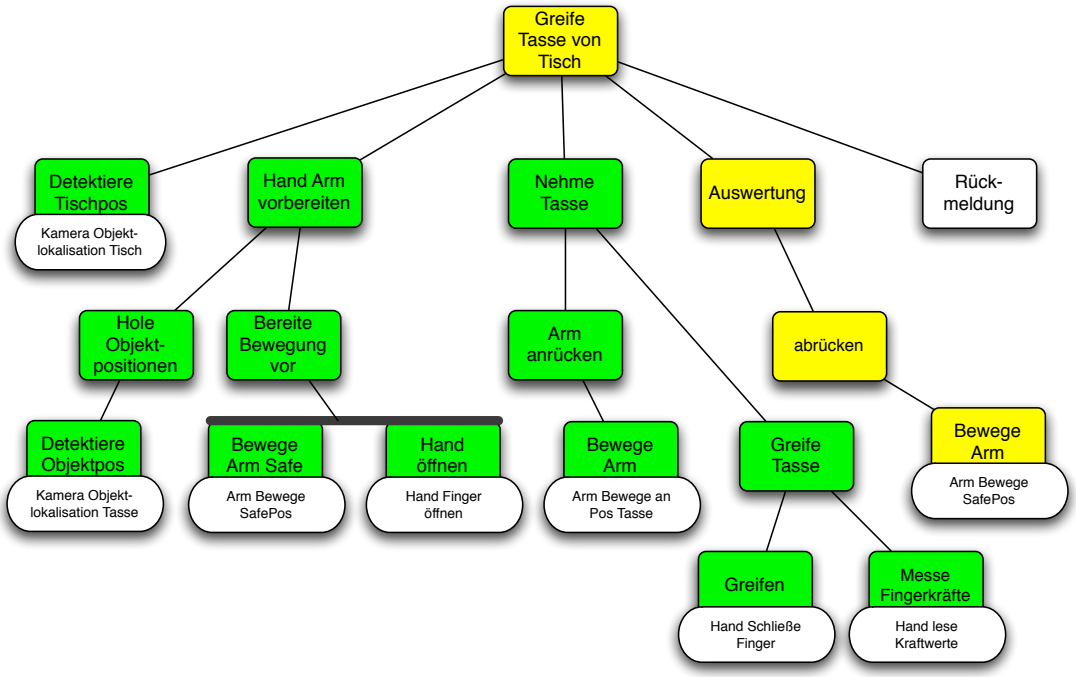


Abbildung 8.12: Dritter Zwischenschritt bei der Bearbeitung des FPs aus Abbildung 8.9.

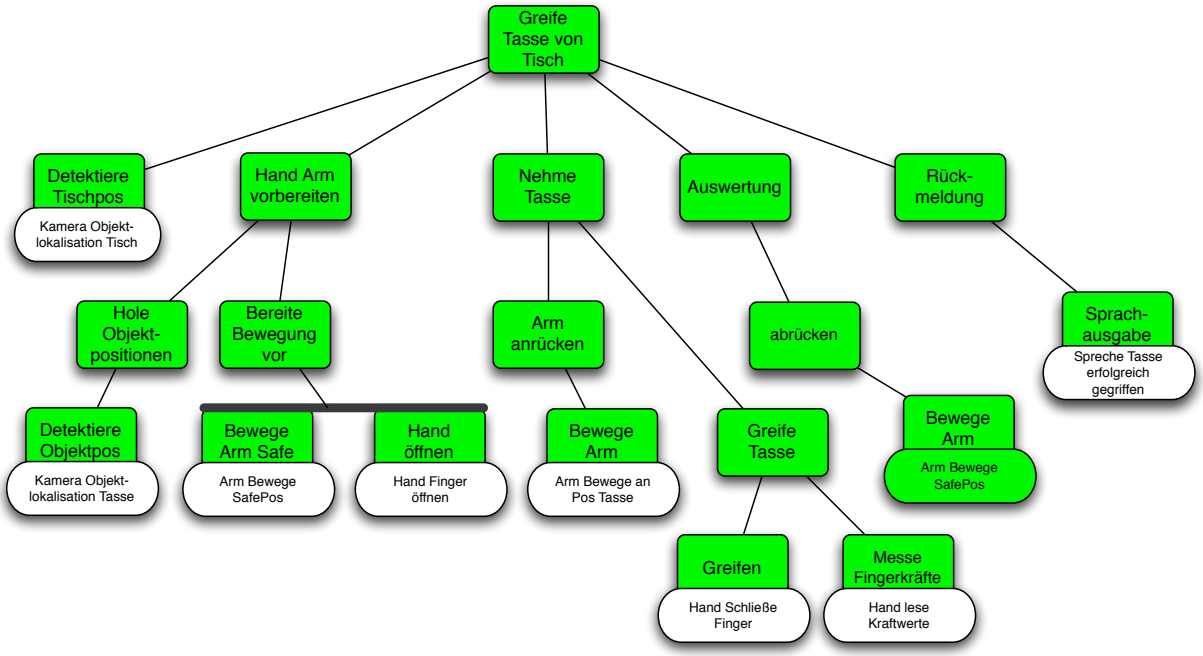


Abbildung 8.13: Das Flexible Programm ist beendet; alle instantiierten Knoten, inklusive der Wurzel, sind als *fertig* markiert.

Abbildung 8.14: Einsatzszenarien für die Bewegungsverfolgung und Aktivitätserkennung. Oben die Roboter ALBERT II (IAIM) und *Jido* (LAAS/CNRS), darunter der Sensoraufbau der Sensorik für Experimente sowie für die Posenverfolgung im Trainingscenter.

8.3.1 Einsatzszenarien

Das vorgestellte Verfahren zur Verfolgung der Körperpose wird in verschiedenen Szenarien eingesetzt. Zusätzlich zum Roboter ALBERT II (siehe auch Abbildung 3.5) wird die vorgestellte Posenverfolgung auch in anderen Systemen genutzt (siehe Abbildung 8.14).

Mit dem Roboter *Jido*, der am LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes), Frankreich entwickelt wurde, konnte die vorgestellte Posenverfolgung für die Interaktion zwischen Mensch und Roboter getestet werden [Fontmarty 07]. Die Posenverfolgung wird zur physischen Interaktion genutzt, um die Roboterbewegungen beispielsweise bei Übergabe ei-

nes Objekts denen des Menschen anzupassen. Auf Basis der Aktivitätserkennung werden Handlungsentscheidungen vom Roboter getroffen. Einzelheiten können den Publikationen des EU-Projektes *Cogniron* [Cogniron 06] entnommen werden.
Zur Unterstützung des Programmierens durch Vormachen kann auch das PdV-Trainingscenter mit einer Tiefenbildkamera ausgestattet werden, so dass die Bewegungen und Aktivitäten des Vorführenden verfolgt und klassifiziert werden können. Der Sensoraufbau muss dabei so positioniert werden, dass die Person sich zumindest zu großen Teilen im Blickfeld der Sensoren befindet.

8.3.2 Bewertung des Fusionsverfahrens

Zur Bewertung des Fusionsverfahrens wird eine Sequenz von ca. 20 Sekunden Länge betrachtet, die verschiedene Arten der Bewegung der zu verfolgenden Person enthält. Dies ist einerseits eine *Verbeugung*. Diese Bewegung ist eine großflächige Bewegung des ganzen Oberkörpers. Die zweite Bewegung in der betrachteten Sequenz ist *Winken* mit einem Arm. Winken ist eine schnelle Bewegung eines Arms.
Abbildung 8.15 zeigt verschiedene Bilder der Originalsequenz in der ersten Zeile. Für die Verfolgung werden folgende Datenquellen verwendet:

- Die Punktwolke der Tiefenbildkamera wird dem Algorithmus zur Verfolgung zur Verfügung gestellt. Die Posenverfolgung allein auf Basis der Tiefendaten ist in der zweiten Zeile von Abbildung 8.15 dargestellt.

- Die Bilder der Farbkamera werden zur Erkennung und Verfolgung von Hautfarbregionen verwendet und dem Algorithmus als Zuordnung zwischen Körpermerkmal (Kopf / Hand) und einem Geradenausschnitt zugeführt (siehe Kapitel 7.2.5). Die Verfolgung allein auf Basis der Hautfarbverfolgung ist in der dritten Zeile dargestellt.

Die Verfolgung der Körperpose auf Basis der Fusion beider Datenquellen ist in der vierten Zeile von Abbildung 8.15 dargestellt. Für die Fusion wurde das Sensormodell für beide Datenquellen entsprechend Tabelle 8.3 gewählt.

Datenquelle	Gewicht in der Fusion
Punkt der Tiefenbildkamera	1.0
Farbbasierte Gesichtserkennung	30.0
Farbbasierte Handerkennung	20.0

Tabelle 8.3: Sensormodell zur Fusion von Tiefenbildern und Hautfarbverfolgung. Relevant ist die Relation der Gewichte; der Tiefenbildsensor wird als Referenz mit $g = 1.0$ genutzt.

Aus der Betrachtung der Sequenz in Abbildung 8.15 lassen sich mehrere Schlüsse hinsichtlich der vorgestellten Fusionsmethode zur Posenverfolgung ziehen:

- Großflächige Bewegungen sind allein mit der Tiefenbildkamera und der resultierenden Punktwolke sehr gut beobachtbar. Dies ist anhand der Verbeugung in den Bildern 50 bis 120 erkennbar. Auf der anderen Seite sind feine, schnelle Bewegungen wie die

Abbildung 8.15: Experimente zur Bewertung der Fusion von verschiedenen Datenquellen. Die Sequenz beinhaltet *verbeugen* und *winken*, zwei sehr unterschiedliche Bewegungstypen. Während *verbeugen* (Bewegung mit dem Torso, dem größten Modellteil) gut verfolgt werden kann allein auf Basis der Tiefenbilder (zweite Zeile), kann *winken* (feine Bewegung der Extremitäten) so nicht verfolgt werden. Die farbbasierte Handverfolgung gleicht dies sehr gut aus für die Fusion (vierte Zeile), während sie aufgrund der Mehrdeutigkeiten allein nicht in der Lage ist, das Modell der Körperbewegung nachzuführen (dritte Zeile).

Winkgeste in den Bildern 120 bis 144 nicht zu verfolgen. Dies liegt vor allem an der geringen Auflösung und schlechten Tiefenauflösung der Tiefenbildkamera.

- Die Verfolgung und Posenschätzung nur auf Basis der Hautfarbmerkmale funktioniert sehr gut und robust für die direkt beobachteten Körperteile (Gesicht und Hände). Allerdings kann aufgrund der hohen Dimensionalität des Körpermodells die Pose nicht korrekt aus diesen wenigen Informationen geschätzt werden. Dies ist besonders auffällig in Bild 81, wo die reine Verfolgung von Gesicht und Händen zu einer falschen Schätzung des Modells führt.

- Die Fusion beider Eingabesensoren führt zu Verbesserungen der genannten Probleme und ist somit in der Lage, die positiven Eigenschaften der einzelnen Ansätze zu kombinieren. Sowohl großflächige Bewegungen sind gut verfolgbar (verbeugen), als auch feine Bewegungen einzelner Extremitäten (winken).

Die vorgestellte Fusion von Merkmalen zur modellbasierten Körperposenschätzung ist also in der Lage, verschiedene Sensordatenquellen gewinnbringend zu kombinieren. Weitere Sensoren können auf die gleiche Weise eingebunden werden, beispielsweise der Laserscanner des Roboters, oder ein Richtmikrophon zur Sprecherverfolgung.

8.3.3 Bewertung der Posenverfolgung

Die Verfolgung der Körperpose wird nach drei verschiedenen Gesichtspunkten evaluiert: Eine qualitative Bewertung, die eine Bewertung durch Benutzer mit einschließt, soll grundsätzlich Aufschluss über die Fehlerraten bei der Verfolgung von Gesten und Aktivitäten geben. Die quantitative Bewertung betrachtet einzelne Parameter des Algorithmus im Detail. Eine Laufzeitbewertung bewertet das Verfahren hinsichtlich des Rechenaufwands, um die Echtzeitfähigkeit der Posenverfolgung zu untersuchen.

Qualitative Bewertung

Eine qualitative Bewertung wurde durchgeführt mit 100 Sequenzen mit jeweils zwischen 15 und 30 Sekunden Länge. Die Sequenzen enthalten 10 verschiedene deutbare Bewegungen, beispielsweise *zeigen*, *laufen*, *verbeugen*, oder *klatschen*, und wurden von verschiedenen Personen durchgeführt.
Diese 100 Sequenzen wurden zusammen mit den Originalaufnahmen der Kamera manuell klassifiziert. Das Ergebnis der Posenverfolgung wurde dabei in einer der drei Gruppen *(0) Bewegung verloren*, *(1) akzeptable Abweichungen*, oder *(2) gute Kongruenz* eingeteilt.
Diese Bewertung ist besonders interessant hinsichtlich der Verwendung der Posenverfolgung zur Klassifikation von Aktivitäten. Offensichtlich können Aktivitäten nur korrekt klassifiziert werden, wenn die Bewegung auch verfolgt werden konnte.
Das Ergebnis der Auswertung ist in Tabelle 8.4 wiedergegeben. Es zeigt sich, dass 95% der Sequenzen akzeptable Abweichungen oder gute Kongruenz mit der ausgeführten Bewegung aufweisen. Werden die drei Klassen mit 0, 1 und 2 bewertet wie angegeben, so errechnet sich der Mittelwert zu $\oslash = 1.58$.
Ein weiteres Kriterium für die Qualität einer Posenschätzung ist die Robustheit gegenüber inhomogenem Hintergrund und gegenüber Verdeckungen. Dies wurde in einer zweiten Evaluation qualitativ betrachtet.

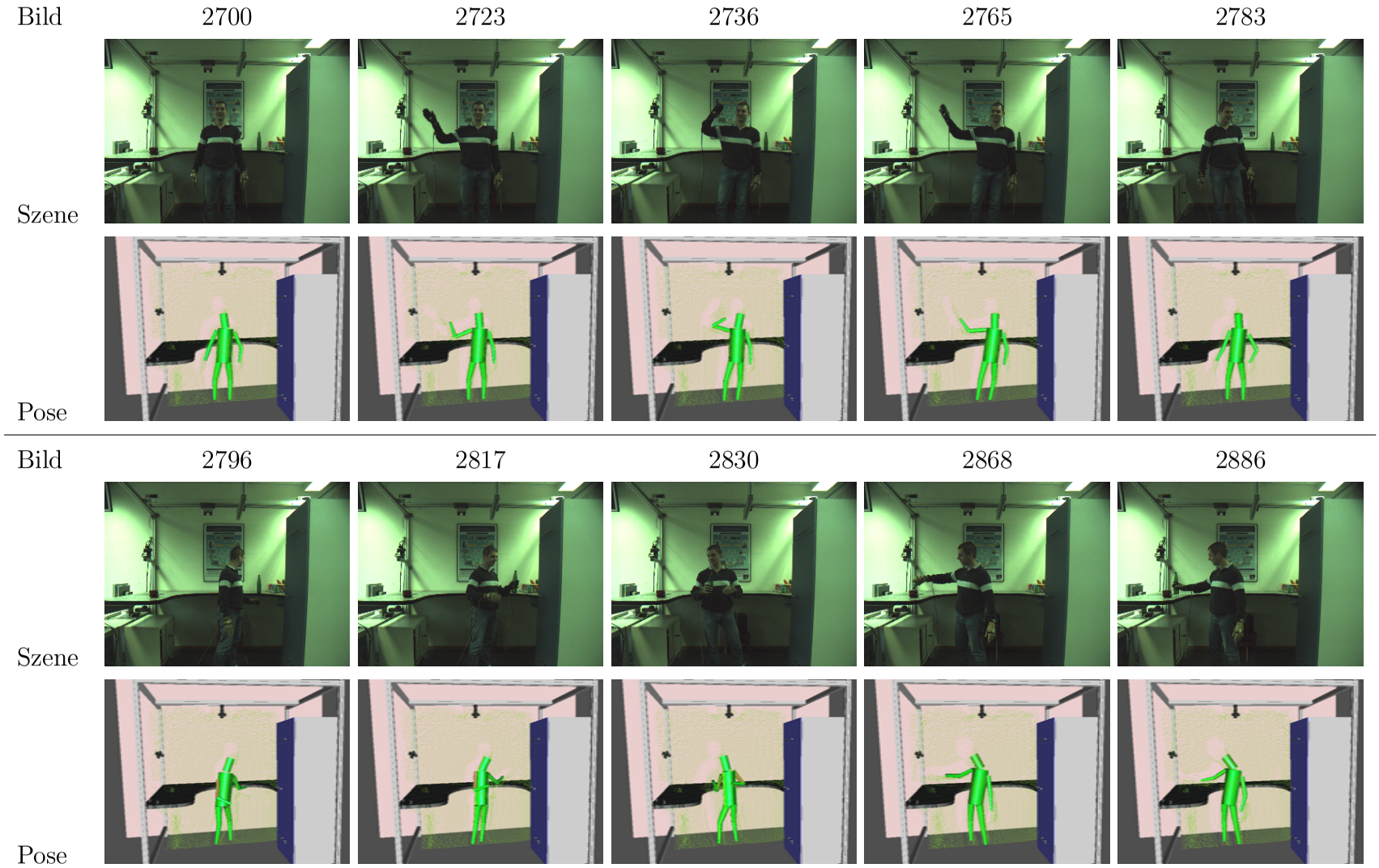


Abbildung 8.16: Sequenz der Posenverfolgung vor inhomogenem Hintergrund (Teil 1). Die Visualisierung der Trainingsumgebung dient der Orientierung und Validierung.

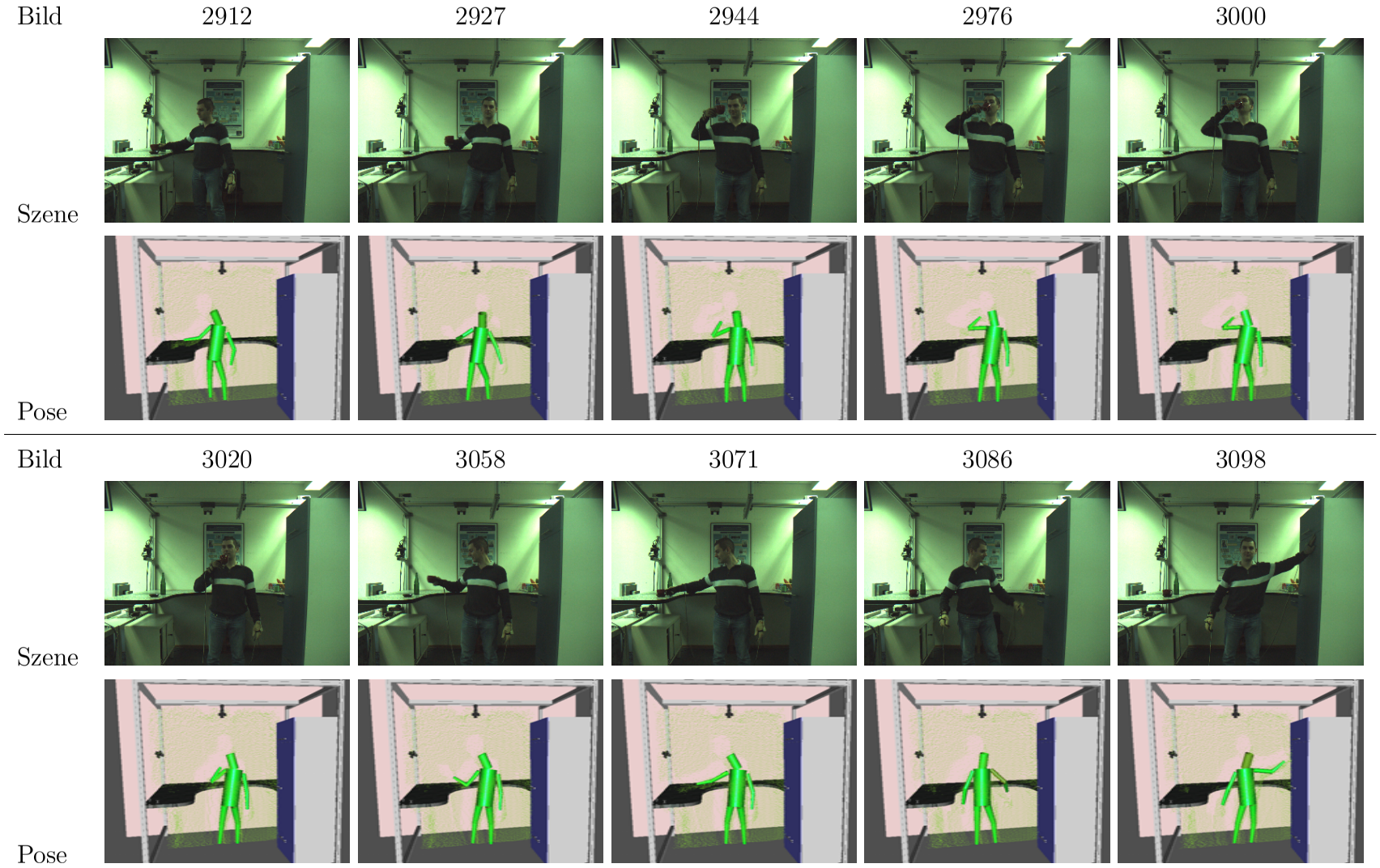


Abbildung 8.17: Sequenz der Posenverfolgung vor inhomogenem Hintergrund (Teil 2)

	(0) Teile verloren	(1) Akzeptable Abweichungen	(2) Gute Kongruenz
Anzahl der Sequenzen	5	32	63

Tabelle 8.4: Ergebnis der qualitativen Bewertung mit 100 Sequenzen. Die Einteilung in die Gruppen erfolgte manuell.

Abbildung 8.16 und 8.17 zeigen Bilder aus einer solchen Sequenz, die in der in Kapitel 3.1 vorgestellten Vorführumgebung aufgezeichnet wurde. Dabei ist zu beachten, dass das Modell nicht frei im Raum steht und damit viele Störquellen vorhanden sind. Die Bewegung kann trotzdem mit akzeptablen Abweichungen verfolgt werden. Allerdings ist zu beachten, dass der Sensor so positioniert wurde, dass er freie Sicht auf den Großteil des Körpers der vorführenden Person hatte.

Quantitative Bewertung

Bei dem vorgestellten Verfahren hängen Laufzeit (und damit die Wiederholrate) und Berechnungsaufwand von der Anzahl der unabhängigen Messungen ab, sind aber unabhängig von den jeweils gewählten Gewichten. Um diese Abhängigkeit allgemein und die Wiederholrate insbesondere zu quantifizieren, wurden verschiedene Untersuchungen durchgeführt. Dabei entspricht das Körpermodell wieder dem in Kapitel 4.4 vorgestellten geometrischen Modell aus 10 verallgemeinerten Zylindern.

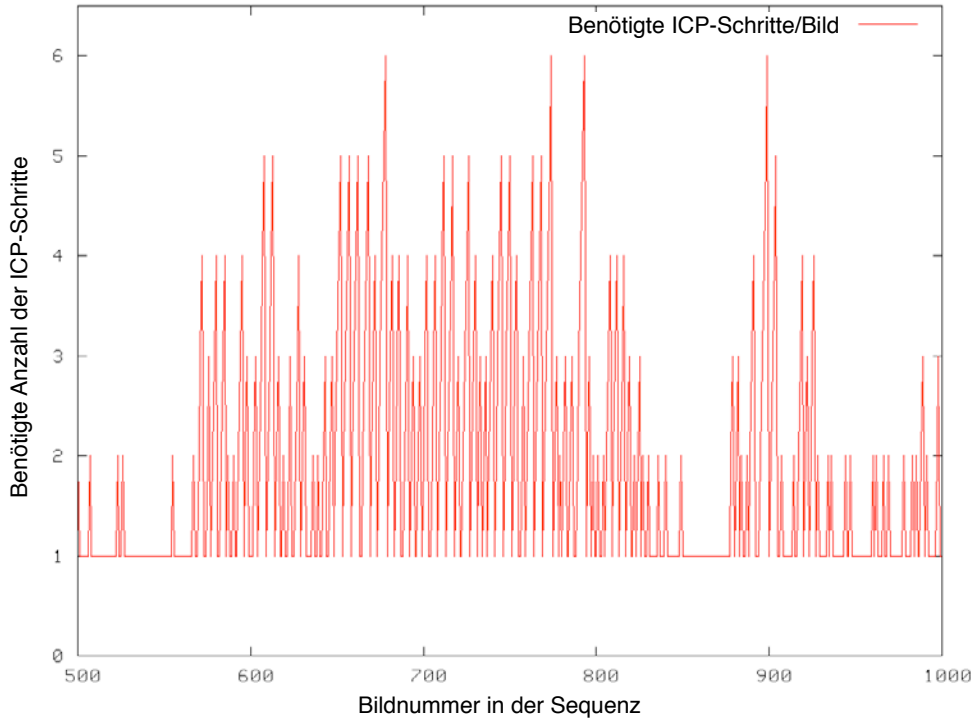


Abbildung 8.18: Anzahl der ICP-Schritte während einer typischen Bewegungssequenz. Dauer der Sequenz: $\approx$ 25 Sekunden.

In allererster Linie hängt der Berechnungsaufwand pro Zyklus von der Anzahl der benötigten ICP-Schritte ab (siehe Abbildung 7.12). Die Anzahl der Iterationen wiederum hängt ab von der Bewegung des Körpers (dem Versatz) zwischen zwei aufeinanderfolgenden Zeitschritten. Abbildung 8.18 zeigt die Anzahl der ICP-Schritte während einer typischen Bewegungssequenz. Es ist zu erkennen, dass in Phasen ohne viel Bewegung eine Iteration pro Zeitschritt ausreichend ist, das Modell anzupassen und nachzuführen (siehe Zeitschritte 500 – 570). Schnelle und ausladende Bewegungen werden dagegen durch mehrere ICP-Schritte pro Zeitschritt kompensiert (650 – 800).

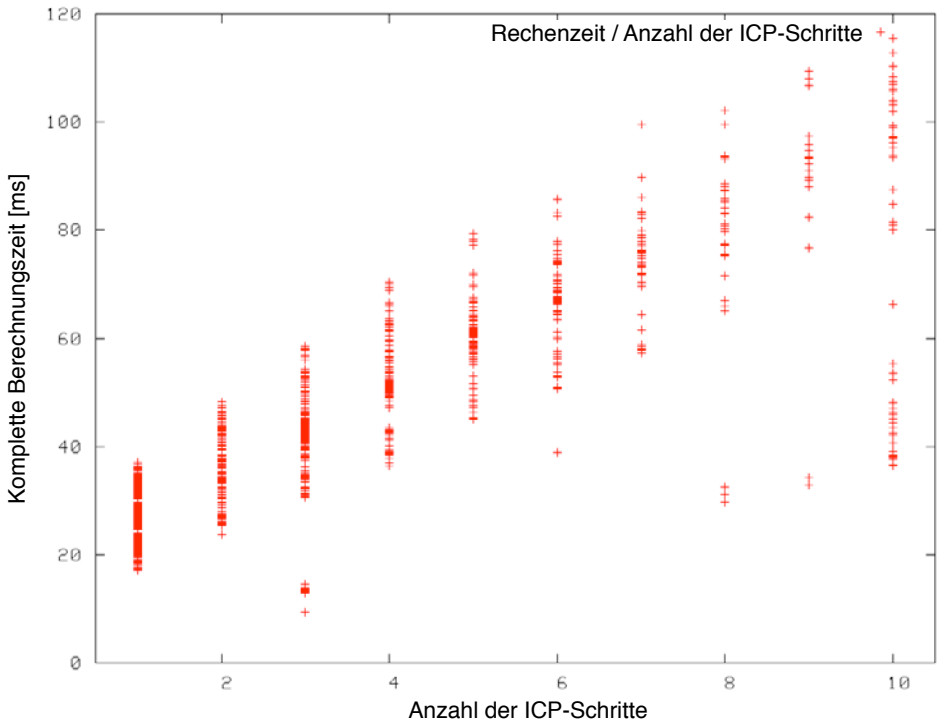


Abbildung 8.19: Benötigte Berechnungszeit für einen Zeitschritt, aufgetragen über der Anzahl der verwendeten ICP-Schritte.

Wie erwähnt, steigt die benötigte Berechnungszeit mit der Anzahl der ICP-Schritte an. Diese Relation ist in Abbildung 8.19 für eine typische Sequenz aufgetragen. Hier zeigt sich ein Nachteil des vorgestellten Verfahrens: Die Anzahl der ICP-Schritte steigt an mit der Geschwindigkeit der Bewegung. Durch die längere Berechnungszeit pro Zeitschritt verringert sich in diesem Fall jedoch noch die Wiederholrate, was zu noch größerem Versatz des Körpers zwischen zwei Zeitschritten des Algorithmus führt. Man erhält also eine negative Abhängigkeit zwischen Bewegungsgeschwindigkeit und Wiederholrate.

Aus diesem Grund wird für übliche Sequenzen heuristisch eine maximale Anzahl von ICP-Schritten gesetzt. Diese liegt üblicherweise bei $n = 6$, so dass nach maximal 6 Schritten abgebrochen und zur nächsten Messung übergegangen wird. Dies hat sich als guter Kompromiss zwischen Genauigkeit und Geschwindigkeit herausgestellt.

Aus diesen Vorraussetzungen ergibt sich laut Abbildung 8.19 eine Periodendauer von 20 – 70ms, was einer Wiederholrate von 14.2 bis 50Hz entspricht. Die maximale Wiederholrate

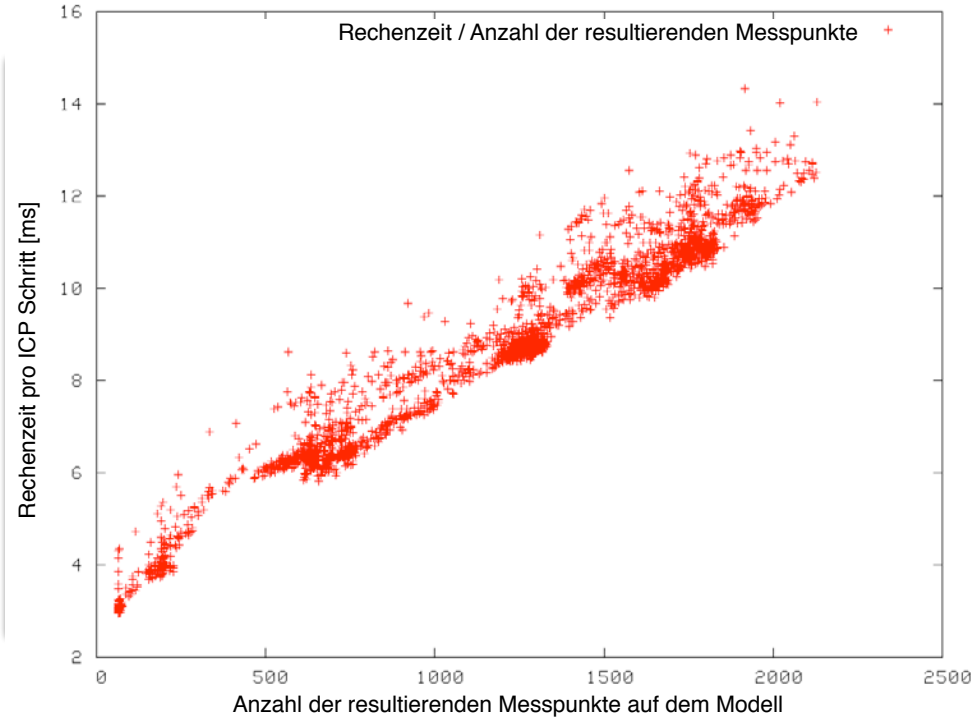


Abbildung 8.20: Berechnungsdauer pro Zeitschritt, aufgetragen über der Anzahl der resultierenden Messungen des zu verfolgenden Körpers

wird also nur durch die Messfrequenz der beteiligten Sensoren begrenzt, die bei 25 – 30Hz liegt.
Die Relation zwischen der resultierenden Anzahl von Messungen des zu verfolgenden Körpers und der Berechnungsdauer eines Iterationsschritts ist in Abbildung 8.20 dargestellt. Für jede Messung des Körpers werden verschiedene (auch geometrische) Berechnungen durchgeführt, was zu der fast vollständig linearen Beziehung in Abbildung 8.20 führt. Diese Abhängigkeit ist ein großer Vorteil des Verfahrens, und nicht offensichtlich, da die resultierende Menge von Messungen nur einen Bruchteil der gesamten betrachteten Menge an Messungen darstellt. Das bedeutet, dass die irrelevanten Messungen die Rechenzeit nicht stark negativ beeinflussen können.

Die Ergebnisse zeigen, dass das vorgestellte Verfahren in der Lage ist, mit mehreren Tausend Messungen pro Zeitschritt umzugehen. Die Rechenzeit bleibt für alle denkbaren Eingaben im Bereich der Echtzeit, das Verfahren kann also direkt auf dem Roboter eingesetzt werden. Ein Nachteil ist die schon angesprochene negative Abhängigkeit zwischen Geschwindigkeit der Bewegung und Berechnungsaufwand: je schneller das Ziel sich bewegt, umso mehr Zeit wird für das iterative Anpassen des Modells benötigt, was wiederum zu größerem Versatz zwischen zwei Zeitschritten führt.
Dies wird durch einen guten Kompromiss zwischen Genauigkeit und Wiederholrate ausgeglichen, indem die maximale Anzahl der Iterationen pro Zeitschritt begrenzt wird. Es ist auch denkbar, an dieser Stelle zwischen verschienden Verhalten und Charakteristika hin und her zu schalten, je nach Anforderung der weiteren Komponenten: Für physische Interakti-

on (Objektübergabe etc.) wird die Genauigkeit erhöht, wodurch die Reaktivität verringert wird. Auf der anderen Seite kann für eine lose Beobachtung eines oder mehrerer Menschen in der Umgebung des Roboters die Beobachtungsgenauigkeit verringert werden, wodurch zum einen der Berechnungsaufwand sinkt und zum anderen die mögliche Dynamik in der Bewegung steigt (siehe auch [Knoop 06c, Knoop 07b]).

8.4 Interpretation und Klassifikation der Bewegungen

Die beobachteten Körperbewegungen dienen in erster Linie zur Klassifikation von Aktivitäten und Gesten des Menschen. Um die in Kapitel 7.3 beschriebenen Verfahren zur Aktivitätsklassifikation zu bewerten, muss zunächst ein Gütemaß für die Bewertung der Klassifikation aufgestellt werden.
Das beschriebene Verfahren beinhaltet keine zeitliche Segmentierung. Für jedes Bild wird eine Hypothese über in dem Moment ausgeführte Aktivitäten berechnet. Diese Hypothese auf Basis der einzelnen Bilder wird zur Bewertung der Klassifikation herangezogen. Das Gütemaß bewertet also die Anzahl der korrekt klassifizierten Bilder innerhalb der Testsequenzen. Diese sind genauso wie die Trainingsdaten manuell annotiert. Die resultierenden Werte r_k der Neuronale Netze, die als Klassifikatoren verwendet werden (siehe Kapitel 7.3.4), werden als zur Klasse gehörend interpretiert, wenn gilt $r_k > 0.7$.
Die Evaluation des vorgestellten Ansatzes erfolgt in zwei getrennten Schritten. Zunächst wird die generelle Aussagekraft der extrahierten Merkmale bewertet, indem der vollständige Merkmalssatz (siehe Tabelle 7.1) zur Klassifikation verwendet wird. Nur wenn diese Menge an Merkmalen die geforderte Information in einer Weise enthält, dass sie durch den Klassifikator extrahiert werden kann, ist eine Reduktion der Merkmalsmenge mit den vorgestellten Verfahren sinnvoll. Eine zweite Auswertung zeigt die Ergebnisse der Auswahl von Merkmalen. Diese werden verglichen mit den Ergebnissen der vollständigen Merkmalsmenge, um den Informationsverlust durch Merkmalsauswahl mit der gewonnenen Geschwindigkeit und Transparenz vergleichen zu können.

8.4.1 Aussagekraft der Merkmale

Tabelle 8.5 zeigt das Ergebnis der Klassifikation einerseits für die Menge der primitiven Merkmale (siehe Kapitel 7.3.2) sowie für die vollständige Merkmalsmenge. Das Klassifikationsergebnis wird dafür folgendermaßen bewertet:

- Ein Bild wird als *korrekt* gewertet, wenn das Bild nur der zugehörigen Klasse zugeordnet wird.
- Ein Bild wird als *mehrdeutig* gewertet, wenn es sowohl zur richtigen Klasse als auch zu mindestens einer weiteren, nicht enthaltenen Klasse zugeordnet wird.
- Als *Fehlklassifikation* werden Bilder gewertet, die nicht der richtigen, aber mindestens einer falschen Klasse zugeordnet werden.
- Ein Bild gilt als *verpasst*, wenn es keiner Klasse zugeordnet wird, obwohl mindestens eine Aktivität enthalten ist.

	Korrekt		Mehrdeutig		Fehlklassifikation		Verpasst	
Balancieren	95.2	*99.5*	2.1	*0.0*	0.3	*0.0*	2.4	*0.5*
Verbeugen	99.0	*99.7*	0.5	*0.0*	0.0	*0.0*	0.5	*0.3*
Heranwinken	97.7	*99.0*	1.1	*0.0*	0.2	*0.0*	1.1	*1.0*
Klatschen	73.9	*99.7*	1.3	*0.0*	2.6	*0.0*	22.2	*0.3*
Flügelschlagen	99.2	*100.0*	0.4	*0.0*	0.0	*0.0*	0.4	*0.0*
Hand schütteln	53.1	*87.6*	0.4	*7.4*	2.2	*0.0*	44.3	*5.0*
Treten	79.1	*94.6*	1.3	*0.0*	2.5	*0.0*	17.1	*5.4*
Manipulation	55.4	*99.3*	0.1	*0.0*	1.9	*0.1*	42.7	*0.7*
Sitzen	99.9	*97.1*	0.1	*2.9*	0.0	*0.0*	0.0	*0.0*
Gehen	69.8	*97.8*	0.3	*0.0*	0.6	*0.0*	29.3	*2.2*
Winken	98.9	*100.0*	0.0	*0.0*	0.0	*0.0*	1.1	*0.0*

Tabelle 8.5: Ergebnis der Aktivitätserkennung für die primitive Merkmalsmenge (links) und die komplette Menge an Merkmalen (rechts, kursiv). Alle Raten sind in % angegeben.

An dieser Stelle zeigen die Ergebnisse bereits, dass die Extraktion weiterer, komplexerer Merkmale aus den primitiven sinnvoll ist. Der Klassifikator kann mit diesen wissensbasiert aufgestellten Merkmalen viele Körperbewegungen wesentlich besser klassifizieren. Dies zeigt sich beispielsweise bei *Hand schütteln*, *klatschen* und *laufen* sehr deutlich, wo der Anteil der korrekt erkannten Bilder bei der vollständigen Merkmalsmenge wesentlich höher ist als bei der Menge der primitiven Merkmale.

8.4.2 Bewertung der Merkmalsauswahl

In einem weiteren Schritt wird die Merkmalsauswahl nach Kapitel 7.3.3 bewertet, indem eine reduzierte Menge an Merkmalen zur Klassifikation verwendet wird. Die Ergebnisse werden dann mit denen der vollständigen Merkmalsmenge verglichen.
Aus der in Tabelle 7.1 Menge von 118 Merkmalen werden zunächst die 10 Merkmale mit der höchsten Relevanz nach Kapitel 7.3.3 gewählt. Die Menge der relevantesten 10 sowie der relevantesten 5 Merkmale wird jeweils für den Trainings- und Klassifikationsschritt des Neuronalen Netzes verwendet.
Abbildung 8.21 zeigt das Klassifikationsergebnis für die mit den vorgestellten Verfahren reduzierten Merkmalsmengen mit 5 bzw. 10 Merkmalen. Es zeigt sich, dass die getroffene Auswahl von Merkmalen sinnvoll und ausreichend ist, da bereits für 5 von 118 Merkmalen die Erkennungsrate bei 8 von 11 Aktivitäten über 95% liegt. Tabelle 8.6 stellt dies detailliert dar.
Tabelle 8.7 fasst die Mittelwerte der mit den einzelnen Merkmalsmengen erreichten Erkennungsraten zusammen. Das Klassifikationsergebnis unter Verwendung der Merkmalsauswahl liegt zwischen dem der vollständigen und der primitiven Merkmalsmenge. Bei einer Reduk-

	Korrekt	Mehrdeutig	Fehlklassifikation	Verpasst
Balancieren	97.3	1.7	0.2	0.8
Verbeugen	97.3	2.0	0.0	0.8
Heranwinken	65.2	4.0	0.0	30.8
Klatschen	92.9	7.1	0.0	0.0
Flügelschlagen	99.4	0.0	0.0	0.6
Hand schütteln	72.5	0.0	1.9	25.6
Treten	94.2	0.9	0.0	4.9
Manipulation	99.8	0.0	0.0	0.2
Sitzen	96.2	3.7	0.0	0.1
Gehen	97.0	0.0	0.0	3.0
Winken	100.0	0.0	0.0	0.0

Tabelle 8.6: Klassifikationsergebnis mit den fünf relevantesten Merkmalen. Alle Raten sind in % angegeben.

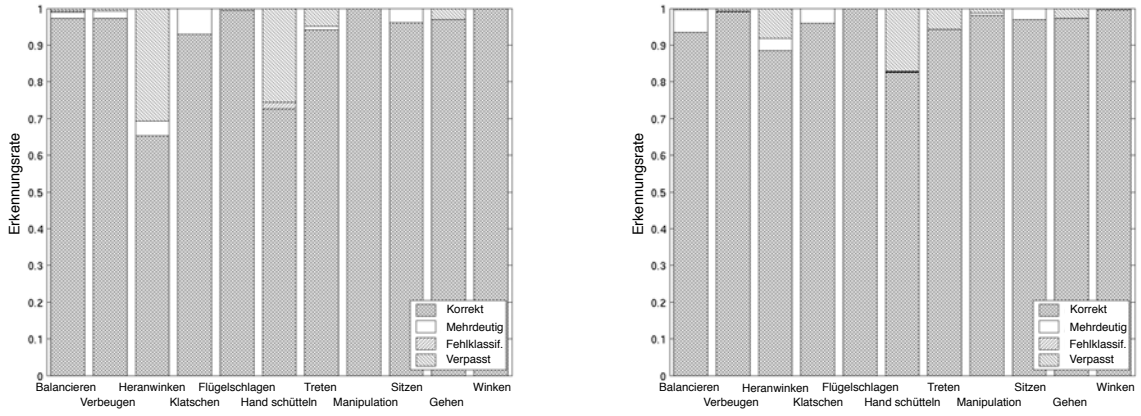


Abbildung 8.21: Visualisierung des Klassifikationsergebnisses für die 5 relevantesten Merkmale (links), sowie die Menge der 10 relevantesten Merkmale (rechts)

Merkmalsmenge	Korrekt	Mehrdeutig	Fehlklassifikation	Verpasst
Primitive Merkmale	84.0 %	0.8 %	0.9 %	14.4 %
Vollständige Menge	98.3 %	0.6 %	0.0 %	1.1 %
5 relevanteste Merkmale	92.3 %	2.1 %	0.1 %	5.5 %
10 relevanteste Merkmale	95.7 %	1.6 %	0.0 %	2.7 %

Tabelle 8.7: Zusammenfassung der Erkennungsraten für die Menge der primitiven Merkmale (45 Merkmale), die vollständige Menge (118 Merkmale), sowie die 5 und 10 als am relevantesten bewerteten Merkmale

tion von 118 auf 10 bzw. 5 Merkmale sinkt die mittlere Erkenungsrate von $98,3\%$ auf $95,7\%$ bzw. $92,3\%$, womit sich zeigt, dass die vorgestellte Merkmalsauswahl in der Lage ist, die Relevanz von Merkmalen zu bewerten.

8.5 Adaption von Handlungswissen

Schlussendlich soll qualitativ ein Experiment vorgestellt werden, das alle beschriebenen Komponenten im Gesamtkontext verwendet und verbindet. Dazu wird ein Handlungsbeispiel ähnlich zu den in Kapitel 7.4 vorgestellten schematischen Beispielen verwendet. Das zu Grunde liegende Flexible Programm ist in Abbildung 8.22 mit zwei Alternativen dargestellt; die Visualisierung ist dem Editor für Flexible Programme entnommen. Auf Basis der Beobachtung und Interpretation des Menschen werden die verschiedenen Handlungsalternativen gewählt. Der gelb hervorgehobene Knoten beinhaltet die hierfür relevante Vorbedingung. Der orange hervorgehobene Teilbaum besteht aus der aktiven Beobachtung einer Person, um auf eine Aufforderungsgeste zu warten.
Die Gesamthandlung aus Abbildung 8.22 besteht also aus den folgenden Schritten:

- Zunächst werden die benötigten Ressourcen *Hand* und *Arm* belegt (Teilbaum *RequestArmHand*).
- Der Roboter winkt einmal mit Hand und Arm, zieht sich zurück und wartet auf eine Reaktion des Menschen (Teilbaum *WaveOnceandWaitForWaving*).
- Winkt der Mensch zurück (Beobachtung orange hervorgehoben), so wird die Handlung fortgesetzt. Für die Beobachtung wird die Ressource *VooDoo* (Beobachtung) belegt und danach sofort wieder freigegeben (Teilbaum *waitForWave*).
- Der vor dem Roboter liegende Apfel wird gegriffen (Teilbaum *PickUpApple*). Die Bewegung besteht aus Anrücken, Greifen und Abrücken.
- Abhängig von der momentanen Situation, in diesem Fall von der momentanen Geste des Menschen, wird eine von zwei Handlungsalternativen gewählt. Macht der Mensch eine Aufforderungsgeste, so wird der Apfel auf einem bereitgestellten Teller platziert (Teilbaum *ReactOnComeOn*, Abbildung 8.22 oben). Ist dies nicht der Fall, behält der Roboter das Objekt in der Hand und zieht den Arm zurück (Abbildung 8.22 unten).

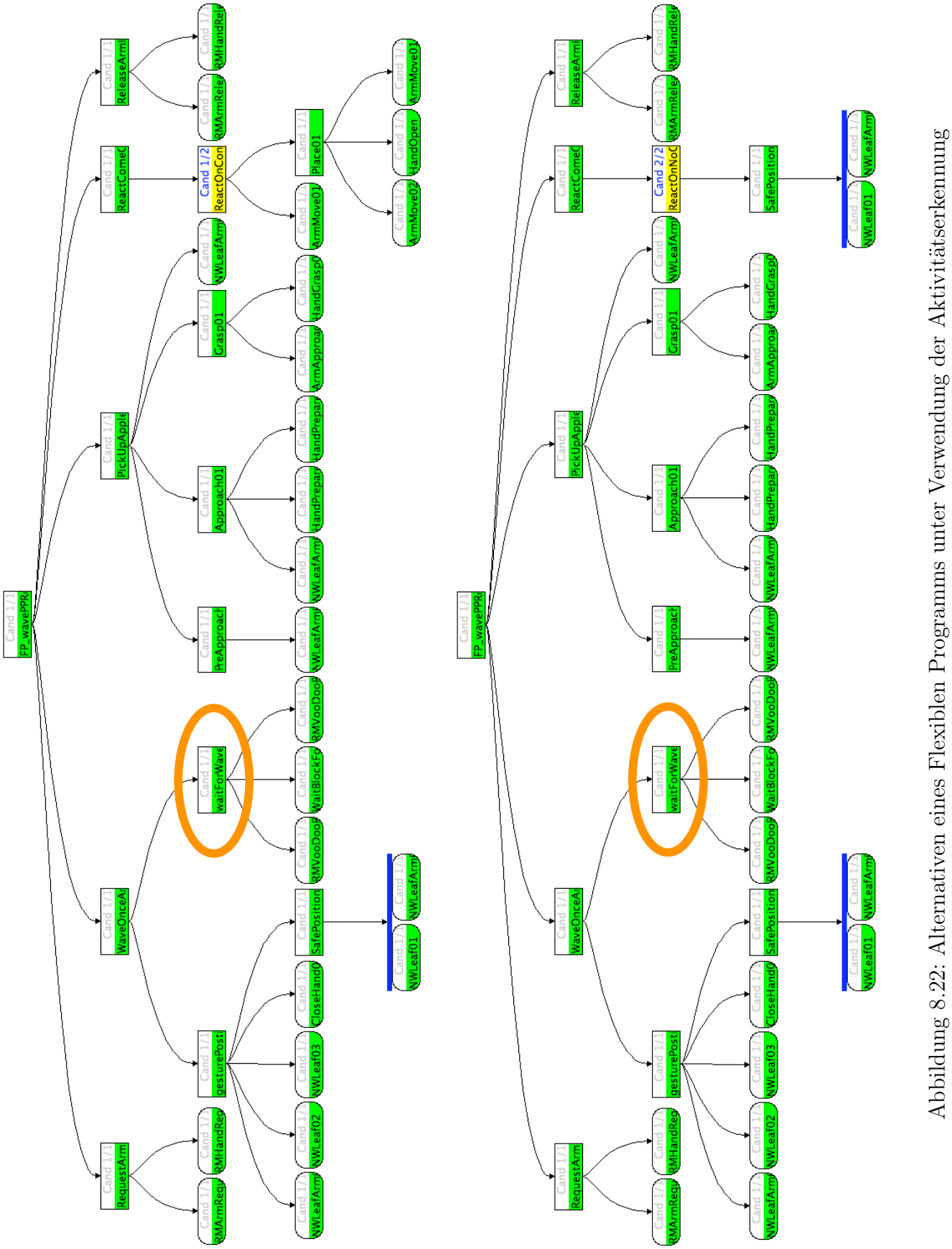


Abbildung 8.22: Alternativen eines Flexiblen Programms unter Verwendung der Aktivitätserkennung

- Schließlich werden alle verwendeten Ressourcen freigegeben.

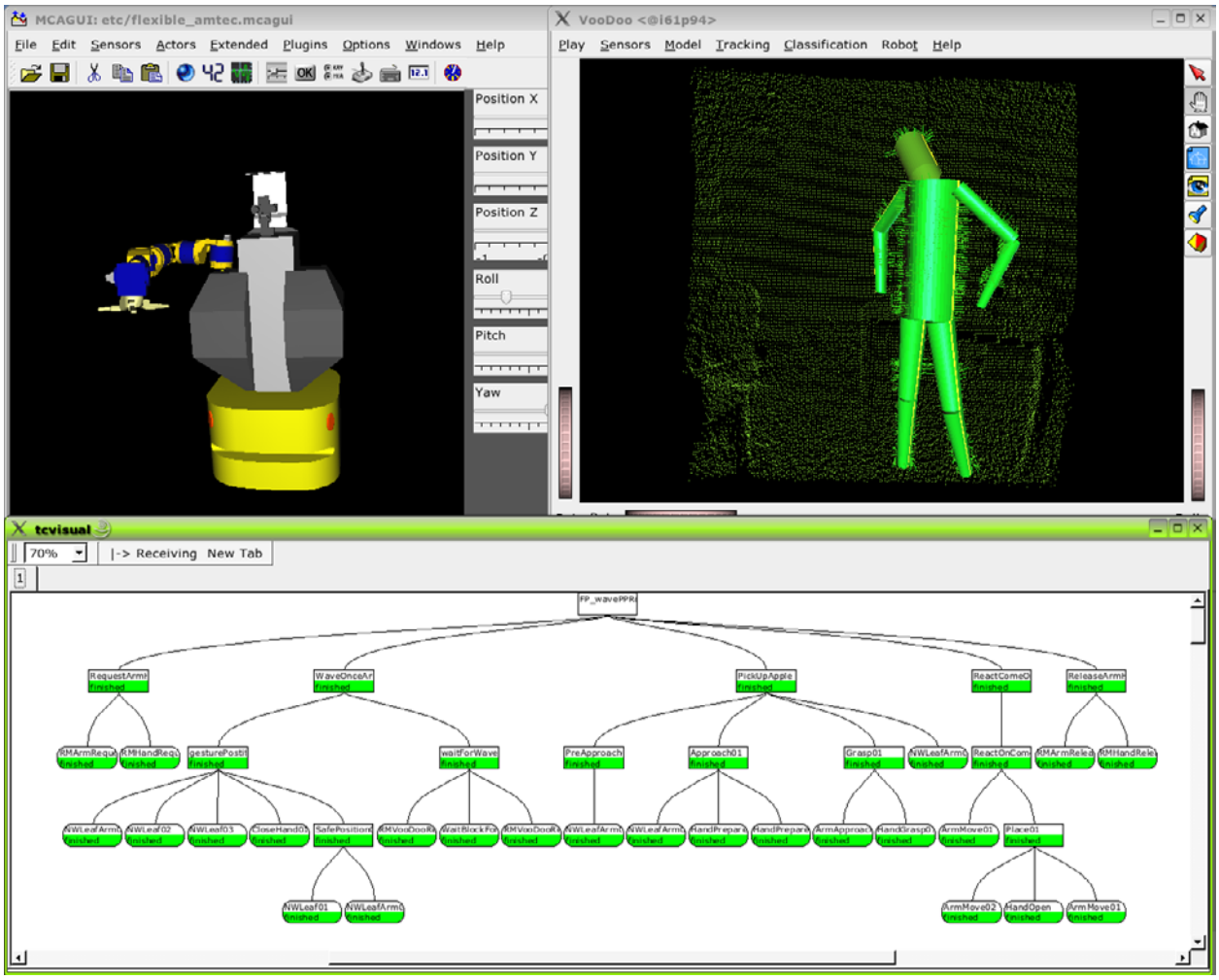


Abbildung 8.23: Experimentierumgebung mit Steuerung und Simulation des Roboters, Personenverfolgung und Aktivitätserkennung sowie Laufzeitdarstellung Flexibler Programme

Abbildung 8.23 zeigt die Benutzeroberfläche der wichtigsten Komponenten zur Robotersteuerung. Die Simulation und Steuerung des Roboters wurde von der Gruppe IDS (Interaktive Diagnosesysteme) am FZI Karlsruhe in MCA2[1] realisiert; des Weiteren sind die Benutzeroberfläche der Personenverfolgung und Bewegungsinterpretation und die graphische Visualisierung des aktuellen Handlungsbaums zu sehen. Alle diese Komponenten sind während der Ausführung mit den Systemen auf dem Roboter verbunden und zeigen in Echtzeit den Systemzustand an.
Mit diesen Steuerelementen ist es nun möglich, das Flexible Programm aus Abbildung 8.22 zu verwenden und die Abläufe zu verfolgen.
Abbildungen 8.24 und 8.25 zeigen eine Sequenz während der Ausführung des FPs aus Abbildung 8.22, wobei jeweils die reale Szene und die Ansicht der Experimentierumgebung zum gleichen Zeitpunkt dargestellt sind. Interessante Punkte der Ausführung sollen im Folgenden diskutiert werden.

[1] siehe `http://mca2.org`

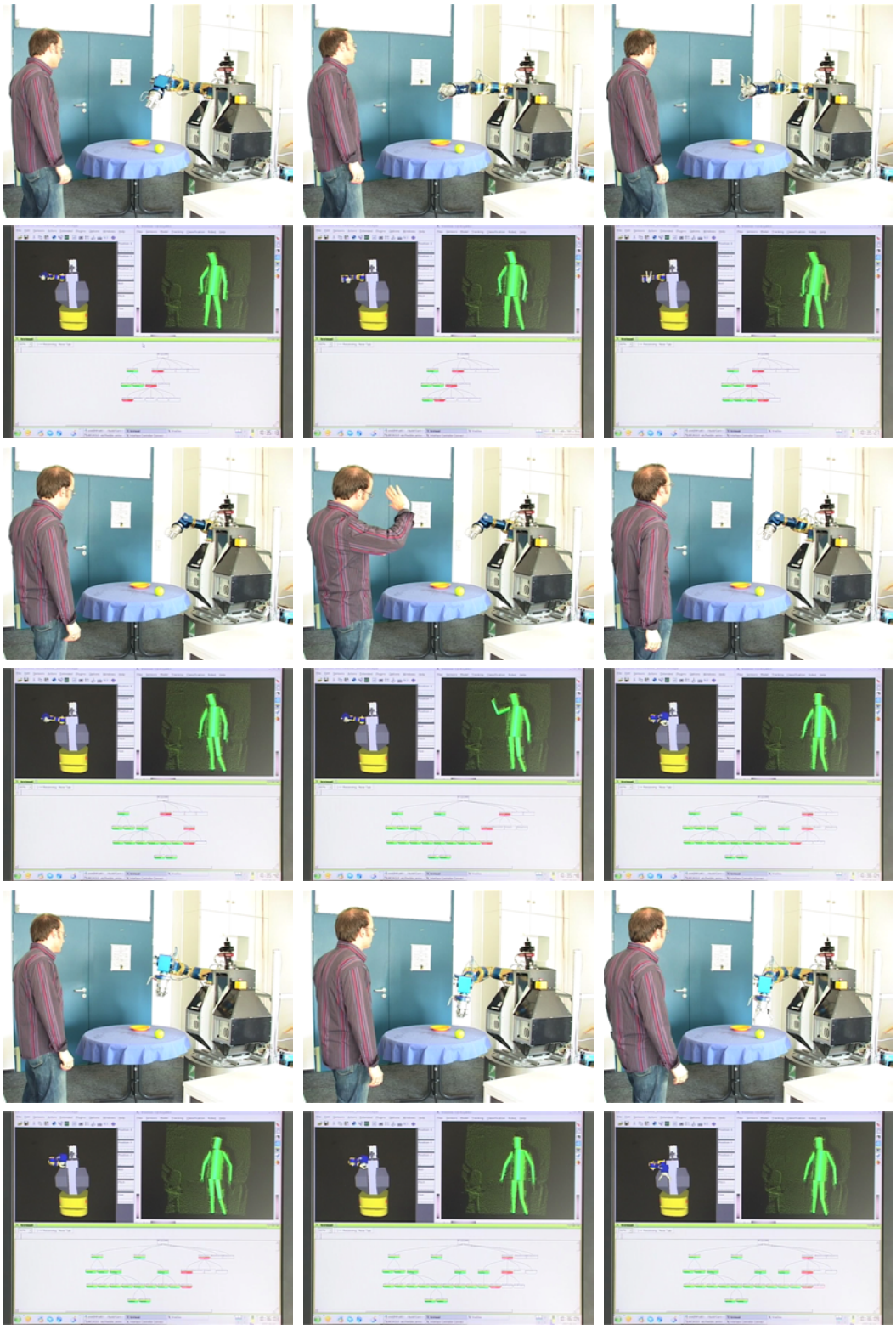
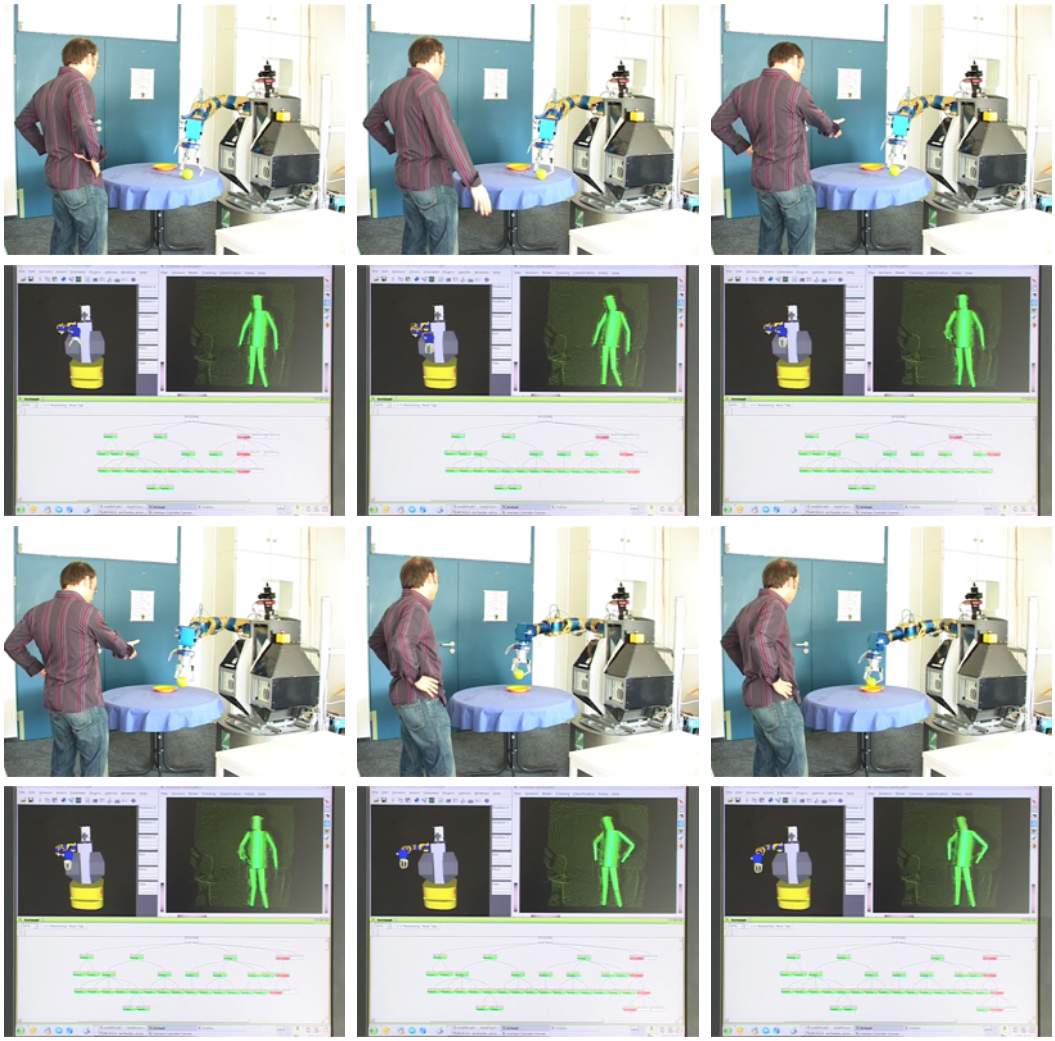


Abbildung 8.24: Ausführung des Flexiblen Programms aus Abbildung 8.22 mit Gestenerkennung zur Adaption des Programms zur Laufzeit (Teil 1). Die obere Sequenz zeigt die reale Szene, die untere die Kontrollmodule mit der Ablaufdarstellung des Flexiblen Programms.



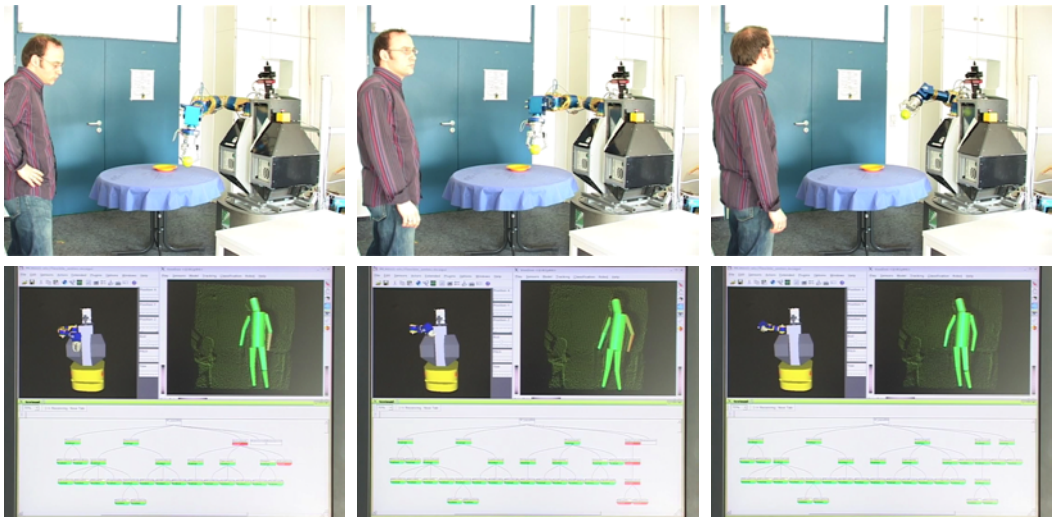


Abbildung 8.25: Ausführung des Flexiblen Programms aus Abbildung 8.22 mit Gestenerkennung zur Adaption des Programms zur Laufzeit (Teil 2). Die untere Reihe zeigt den Ablauf mit alternativer Handlungskette (Untere Alternative aus Abbildung 8.22).

- Am Anfang der Ausführung sind alle benötigten Ressourcen frei. Die Belegung erfolgt also instantan ohne Verzögerung.

- Die Ausführung der Geste (Abbildung 8.24, obere Reihe rechts) besteht aus einer Armbewegung und zwei Handbewegungen. Danach wird die Hand geschlossen und der Arm in eine sichere Position bewegt.

- An dieser Stelle wartet der Roboter auf das Winken der beobachteten Person. Die Beobachtungskomponente wird dazu zunächst als Ressource belegt. Sie wird so parametriert, dass durch eine Wink-Geste eine entsprechende Rückgabe erzeugt wird. Dann wird die Ressource wieder freigegeben. Diese Schritte sind in Abbildung 8.22 orange markiert.

- Die Abläufe zum Greifen des Objekts sind ähnlich zum Beispiel aus Kapitel 8.2.2 aufgebaut.

- Der in Abbildung 8.22 gelb markierte Knoten enthält alternative Handlungsstränge für verschiedene Situationen. An dieser Stelle wird durch Überprüfung von Bedingungen im bekannten Umweltmodell zwischen diesen Alternativen ausgewählt. Dabei wird ausgenutzt, dass die Beobachtungskomponente erkannte Gesten ständig im Umweltmodell aktualisiert. Ist eine Aufforderungsgeste zum Entscheidungszeitpunkt erkannt worden, so wird der Ablauf in Abbildung 8.25 der oberen zwei Bildreihen gewählt: Das Objekt wird auf dem Teller platziert. Ist diese Geste nicht vorhanden, wird der Ablauf entsprechend der unteren Reihe ausgeführt: Der Roboter behält das Objekt in der Hand.

- In der Visualisierung des laufenden Flexiblen Programms erkennt man deutlich den inkrementellen Aufbau der Handlung zur Laufzeit. Die Handlungsalternativen werden erst dann gewählt, wenn der Elternknoten betreten wird. Am Beispiel der gezeigten Alternativen ist dies in Abbildung 8.25, oben rechts der Fall.

Die Beobachtungskomponente unterscheidet drei Fälle. Entweder ist kein Mensch erkannt, oder ein Mensch ist erkannt, führt aber keine bekannte Geste aus, oder ein Mensch ist erkannt und führt eine der trainierten Geste aus. Im letzteren Fall wird zusätzlich der erkannte Gestentyp gemeldet. Zur Parametrierung und Adaption von Flexiblen Programmen kann also auch die Anwesenheit von Menschen genutzt werden. Dies ist sinnvoll, wenn eine Person für die Ausführung zwingend notwendig ist, beispielsweise bei einer Objektübergabe.
Grundsätzlich zeigt dieses Experiment mögliche Wege, die Beobachtung und Erkennung von Gesten in Roboterprogrammen zu nutzen. Die Beobachtungskomponente wird genauso wie alle anderen Teilkomponenten behandelt, und kann dadurch sowohl für aktive Beobachtung parametriert werden als auch zur passiven (nicht gezielten) Aktualisierung des Umweltmodells beitragen. Genauso können Parameter des Körpermodells für die Interaktion genutzt werden. Werden beispielsweise die beobachteten Positionen der Hände der Person im Umweltmodell gespeichert und aktualisiert, ist eine adaptive Steuerung des Arms möglich. Allerdings sollte der Weg über das Umweltmodell nicht für sensible Regelungen genutzt werden; hierfür sind weitere, virtuelle Komponenten notwendig, die direkt mit den Teilkomponenten interagieren.

8.6 Zusammenfassung

In mehreren Experimenten wurde die Leistungsfähigkeit der vorgestellten Verfahren demonstriert. Eine Versuchsreihe zur Abbildung von Handlungswissen auf einen konkreten Roboter sowie die Ausführung solcher Programme in Simulation und auf dem realen Roboter haben die Anwendbarkeit und Eignung der vorgestellten Modelle und Verfahren demonstriert. Die zur Interaktion vorgestellte Posenverfolgung und Klassifikation von Gesten und Aktivitäten konnte an mehreren Beispielen und in verschiedenen Umgebungen demonstriert werden. An den Experimenten zeigt sich die Echtzeitfähigkeit und Robustheit der Verfahren. Insbesondere ist hier die Verfolgung der Körperpose zu erwähnen, da dies große Mengen an Messungen und die Anpassung einer hohen Anzahl an Freiheitsgraden bedeutet. Dabei wurde nur auf roboterinterne Sensorik zurückgegriffen.
Das Experiment zur Adaption von Handlungswissen umfasst letztendlich alle vorgestellten Verfahren und Komponenten und stellt auch die Verwendbarkeit im Gesamtkontext unter Beweis.

Kapitel 9

Zusammenfassung

Gegenstand dieser Arbeit waren Methoden zur Darstellung und Erzeugung von Handlungswissen für Serviceroboter. Im folgenden Kapitel sollen nun der Beitrag und die Erkenntnisse zusammenfassend diskutiert werden, um abschließend einen Ausblick auf mögliche Erweiterungen, Ergänzungen und Konsequenzen geben zu können.

9.1 Ergebnisse und Beitrag

Wachsende Rechnerkapazitäten, neue Sensorkonzepte und zunehmend kompaktere Aktorik ermöglichen es in den letzten Jahren immer mehr, flexible, integrierte Roboter mit hochentwickelten, perzeptiven und aktorischen Fähigkeiten zu konstruieren. Diese Entwicklung verlangt nach einer flexiblen Repräsentation für Handlungswissen, die auch auf intuitivem Wege programmiert werden kann. Die vorliegende Arbeit stellt Lösungsansätze hierfür vor. Ausgangspunkt waren die Forderungen nach einer flexiblen Handlungsbeschreibung für Serviceroboter, der Möglichkeit der Erzeugung dieses Handlungswissens aus abstraktem Handlungswissen, sowie der Erkennung und Interpretation von menschlichen Bewegungen zur Parametrierung und Kommandierung von Handlungen. Damit gliedert sich die Arbeit ein in den Prozess des Programmierens durch Vormachen. Der Fokus liegt insbesondere auf der Abbildung akquirierten Handlungswissens auf ausführende Systeme sowie auf der Interaktion mit diesen.
Frühere Arbeiten diskutieren diese Fragestellung zum einen aus Sicht der Vorführung und des Lernens: Durch Beobachtung einer Vorführung einer Aufgabe wird Handlungswissen abgeleitet, das generalisiert und abgelegt werden kann. Zum anderen wird die Fragestellung anhand existierender Robotersysteme untersucht, die schon heutzutage komplexe Handlungen durchführen können. Die Abbildung gelernten Handlungswissens auf solche Systeme ist allerdings bisher kaum betrachtet worden und ein wesentlicher Beitrag dieser Arbeit.
Folgende Aspekte wurden dabei im Einzelnen untersucht:

- Eine Handlungsrepräsentation in Form von Hierarchischen Aufgabennetzwerken wurde entwickelt, die zur Laufzeit erzeugt und verändert werden kann. Diese Flexiblen Programme bestehen aus hierarchisch verknüpften Teilhandlungen, die aus atomaren Handlungen aufgebaut sind. Vorteil einer hierarchischen Handlungsrepräsentation ist dabei vor allem die Lesbarkeit und Wiederverwendbarkeit von Handlungsteilen.

- Die Kontrolle und Ausführung dieser Handlungen wurde eingebettet in eine hybride Architektur, die eine flexible Zusammenstellung von Ausführungskomponenten ermöglicht. Die benötigten Konzepte und Komponenten zur Darstellung und Kommunikation wurden entwickelt.
- Eine Handlungsrepräsentation gewinnt erst dann Flexibilität, wenn zur Laufzeit Zustände und Bedingungen in der Umwelt beobachtet und ausgewertet werden können, um Entscheidungen zwischen Handlungsalternativen zu treffen. Eine solche Repräsentation sowie die notwendigen Komponenten und Schnittstellen wurden vorgestellt.
- Ein allgemeingültiger Ansatz zur Erzeugung ausführbaren Handlungswissens für Serviceroboter aus abstrakten Handlungen wurde eingeführt. Am Beispiel der Erzeugung von Flexiblen Programmen aus Makrooperatoren wurden diese Abbildung sowie das allgemein notwendige Hintergrundwissen über das ausführende System untersucht.
- Zur Handlungsauswahl wurde ein Kontextmodell sowie ein Verfahren vorgestellt, kontextbasiert Handlungen zu aktivieren und zu starten. Der Kontext wird dabei repräsentiert durch eine Menge an Zuständen der Umwelt und des Roboters selbst. Zur Auswahl von Handlungen wurde ein Handlungsrahmen definiert, mit dessen Hilfe Flexible Programme mit Situationen oder Teilsituationen assoziiert werden können.
- Um einerseits die Beobachtung während des Programmierens durch Vormachen zu unterstützen, und andererseits die Interaktion zwischen Mensch und Roboter intuitiv zu gestalten, wurde ein Ansatz vorgestellt, mit dessen Hilfe menschliche Gesten und Aktivitäten beobachtet und interpretiert werden können. Damit ist es möglich, den Roboter intuitiv zu kommandieren. Es eröffnet die Möglichkeit, Programme während ihrer Ausführung zu parametrieren oder zu modifizieren, indem Gesten und Aktivitäten durch den Roboter interpretiert werden.

Zur Evaluierung der vorgestellten Verfahren wurden umfangreiche Experimente mit dem Roboter, einer Simulationsumgebung sowie einzelnen Komponenten durchgeführt und dokumentiert. Dabei lag der Fokus auf Handhabungen, die im Haushalt alltäglich sind. Dies sind Handlungen, die auch durch das PdV-System modelliert sind. Das vorgestellte Verfahren zur Posenverfolgung des Menschen wurde in verschiedene Systeme eingebettet und evaluiert. Dazu gehören sowohl verschiedene Roboter als auch die Trainingsumgebung des Programmierens durch Vormachen.

9.2 Diskussion

Durch die in dieser Arbeit vorgestellten Verfahren werden Vorführung, Ausführung und Adaption von Handlungswissen für Serviceroboter in ein Gesamtkonzept zusammengeführt. Sowohl aus den Ansätzen der entwickelten Einzelkomponenten als auch aus der Integration dieser auf dem Serviceroboter Albert II und deren Evaluierung lassen sich wesentliche Punkte ableiten. Dies sind zu allererst die Vorraussetzungen, die für die Anwendung der vorgestellten Verfahren und Modellierung erfüllt sein müssen.

- Das Modell für Handlungswissen auf Seite der Vorführung, der Makrooperator, wird modelliert mit einer Menge an Basisoperationen. Die demonstrierte Handlung muss

so gestaltet sein, dass sie mit diesen Basisoperationen repräsentierbar ist. Dies stellt eine Forderung an den Vorführenden dar, die während der Vorführung beachtet werden muss. Zusätzlich müssen die demonstrierten Operationen für das System sichtbar und beobachtbar sein. Dies gilt genauso auf PdV-Seite für Objekt- und Grifferkennung wie bei der Erkennung von Gesten und Aktivitäten auf dem Roboter.

- Die zur Ausführung auf den Roboter abzubildenden Makrooperatoren müssen Aufgaben enthalten, die durch das Robotersystem ausführbar sind. Grundsätzlich ist es möglich, beispielsweise zweihändige Handlungen zu demonstrieren; diese können von einem Roboter mit einem Manipulator jedoch nicht durchgeführt werden, wenn sie koordinierte Aktionen der Hände enthalten.

- Die Abbildung von abstraktem Handlungswissen auf Roboterprogramme muss als Abbildungsfunktion für ein dediziertes Robotersystem bekannt sein. Diese besteht aus den in Kapitel 5.9 diskutierten Komponenten. Die Abbildung muss auch für jeden weiteren Robotertyp neu definiert werden, da sie abhängig von der Konfiguration, Beschreibungssprache etc. des Zielroboters ist.

- Gesten und Aktivitäten werden auf Basis des vorgestellten Körpermodells des Menschen erkannt. Entsprechend können nur solche Bewegungen klassifiziert werden, die mit dieser Modellierung darstellbar sind. Das Modell kann zwar problemlos auf Hände, Füße oder andere Strukturen erweitert werden, allerdings sind die Bewegungen dieser Gliedmaßen durch die Robotersensorik nicht auflösbar. Gesten mit Fingern, Anzeigen von Zahlen mit den Fingern, etc. sind also nicht erkennbar. Die Erkennung, und damit die Kommunikation auf Basis von Gesten, beschränkt sich auf ausladende Körperbewegungen.

Im Gegensatz zu früheren Ansätzen werden Vorführung und Ausführung in dieser Arbeit größtenteils entkoppelt. Dies ist grundsätzlich wünschenswert, um dem Vorführenden die Demonstration in seinem eigenen Aktionsraum zu ermöglichen. Allerdings muss die Demonstration immer noch unter verschiedenen Randbedingungen geschehen; Vorführung und Ausführung können noch stärker in ihren Aktionsräumen getrennt werden. Bei der Abbildung von Makrooperatoren auf Roboterprogramme zeigt sich diese Einschränkung in der mangelnden Flexibilität bei der Abbildung der Trajektorien. Die Abbildung solcher subsymbolischer Parameter kann ein weiterer Schritt hin zur Unabhängigkeit zwischen Vorführung und Ausführung sein.
Gleichzeitig werden Vorführung und Ausführung in dieser Arbeit zeitlich näher zusammengebracht, mit der Vision, das Handlungslernen direkt am Robotersystem interaktiv und schrittweise durchzuführen. Die Bewegungsbeobachtung sowie die flexible Handlungsbeschreibung, die zur Laufzeit erzeugt und verändert werden kann, sind entscheidende Schritte hin zu diesem Ziel.
Die flexible Handlungsbeschreibung ist hierarchisch in Form von Aufgabennetzwerken aufgebaut. Es zeigt sich, dass diese Repräsentation intuitiv verständlich ist, und durch die Wiederverwendbarkeit von Teilhandlungen auch das mauelle Aufbauen und Testen von Handlungen stark unterstützt. Die graphische Visualisierung, eingebettet in die vorgestellte Funktionalität des Editors für Handlungsbäume, erleichtert sowohl den Prozess der Erstellung als auch der Überwachung von Handlungen. Die im Handlungsbaum enthaltene Abstraktion von Teilhandlungen kann weiter genutzt werden, um auch intuitiver fassbare Informationen über den

internen Roboterzustand auszugeben. Mechanismen hierfür wurden in Form von verschiedenen graphischen Visualisierungen vorgestellt, deren Darstellungen sich allgemeinverständlich an Comic-ähnlichen Ausdrücken und Symbolen orientiert.

Die vorgestellte Kontextmodellierung sowie das Konzept zur kontextbasierten Handlungsauswahl bereiten den Weg zu flexiblen Servicerobotersystemen. Die zugehörigen Regeln werden manuell aufgestellt; dies ist der erste Schritt hin zu noch flexibleren, autonomen Servicerobotern.

Die Erkennung menschlicher Aktivitäten auf Basis der gesamten Körperbewegung stellt einen universellen Ansatz zur Interpretation menschlichen Verhaltens dar. Einfache Aktivitäten sind auf diese Weise gut und robust erkennbar, was durch die vorgestellten Ergebnisse und Evaluationen belegt wird. Sowohl die Erfahrung als auch die Literatur zeigen allerdings auch, dass zur Erkennung komplexer menschlicher Aktivitäten die Betrachtung des Kontexts unabdingbar ist. Dies geschieht im Rahmen der vorliegenden Arbeit nicht; die Menge der Aktivitäten beinhaltet nur einfache Gesten und Aktivitäten, die ohne weiteres Wissen direkt aus der Bewegung ableitbar sind. Das ist ausreichend für direkte Kommandos und Gesten, was in den vorgestellten Experimenten demonstriert wird. soll dieses Konzept jedoch zukünftig auch zur Erkennung komplexer Aktivitäten eingesetzt werden, ist eine tiefere Modellierung und Verwendung des Kontexts, also der Umwelt des Roboters sowie des Menschen, eminent wichtig. Letzteres war nicht Gegenstand der vorliegenden Arbeit; die Beobachtung der Körperbewegung legt nichtsdestotrotz zu einer solchen Situationsinterpretation eine wichtige Grundlage.

9.3 Ausblick

Aus den genannten Punkten lassen sich verschiedene Aufgabenstellungen ableiten, welche die in der vorliegenden Arbeit vorgestellten Verfahren sinnvoll erweitern und ergänzen können.

Zur Abbildung von Parametern aus Makrooperatoren auf Roboterprogramme kann eine Betrachtung der Arbeitsräume sowie eine Definition von Abbildungen zwischen diesen eine Ausführung auch dann ermöglichen, wenn die Arbeitsräume von Demonstrator und ausführendem System äußerst unterschiedlich ausfallen. Ansätze dazu sind in der Literatur zu finden (siehe [Alissandrakis 02c]).

Die automatisch erzeugten Roboterprogramme werden im Konzept der vorliegenden Arbeit durch die Ausführung evaluiert. Dies ist ein riskantes Vorgehen; grundsätzlich sollten Roboterprogramme vor der Ausführung formal und inhaltlich bewertet werden können, um die Korrektheit soweit wie möglich gewährleisten zu können. Ansätze zu einer solchen Bewertung sind bisher nicht entwickelt worden, würden das Konzept aber wesentlich unterstützen und erweitern. Dies ist ein offenes Problem; auch in der Literatur findet sich dazu keine allgemeine Lösung.

Die Modellierung des Kontexts sowie die darauf basierende Handlungsauswahl kann feiner modelliert und beispielsweise mit probabilistischen Modellen dargestellt werden. Dadurch ergibt sich ein in der Entscheidung flexibleres System, dessen Entscheidungsmechanismen auch mit Verfahren des Maschinellen Lernens verknüpft werden können.

Um auch komplexe Aktivitäten erkennen und klassifizieren zu können, ist die Betrachtung und Modellierung des Kontexts unerlässlich. Ein solcher Schritt ließe sich mit der kontext-

basierten Handlungsauswahl verbinden, sofern beide eine einheitliche Kontextmodellierung verwenden.
Damit entstünde ein Robotersystem, das einerseits in der Lage ist, neue Handlungen zu lernen und in die eigene Repräsentation zu übertragen, andererseits auch die Entscheidungsmechanismen besitzt, diese Handlungen kontextbasiert auszuwählen. Dies ist ein großer Schritt in Richtung des flexiblen Serviceroboters, der ständig lernt und vorhandenes Wissen optimiert und überprüft.

Anhang A

Technische Daten der Sensorik

Der Serviceroboter ALBERT II besitzt mehrere Sensoren zur Erfassung der Umwelt. Dazu zählt der Laserscanner der Firma SICK [SICK 06], der zur Navigation verwendet wird, dessen Messwerte aber auch für die Personenverfolgung eingesetzt werden können. Des Weiteren besitzt der Roboter eine Stereo-Farbkamera und eine Tiefenbildkamera. Diese sind auf einer Schwenk-Neige-Einheit von Typ PTU-46-17.5 der Firma *Directed Perception* [Perception 07] zusammen montiert.

A.1 Die Stereo-Farbkamera

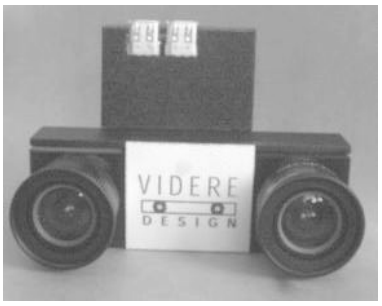


Abbildung A.1: Mega-D Kamera von Videre Design

Die Stereo-Farbkamera der Firma *Videre Design* [Vid 06] ist vom Typ *Mega-D* und besitzt zwei CMOS-Sensoren mit einer Auflösung von jeweils 1288×1032 Pixeln. Die Kamera wird über *FireWire* angeschlossen und ausgelesen. Daraus ergeben sich die maximalen Bildraten. Die Auflösung kann reduziert werden, wodurch sich die Bildrate von 7.5 Bilder pro Sekunde (*frames per second, fps*) bei voller Auflösung auf 26 fps bei 640×480 und 110 fps bei 320×240 (siehe [Videre Design 01]) erhöht. Die genauen Daten des Sensors (siehe Abbildung A.1) finden sich in Tabelle A.1.

A.2 Die 3d Tiefenbildkamera

Die Tiefenbildkamera von Typ *Swissranger SR2* (siehe Abbildung A.2) wurde von CSEM, Schweiz hergestellt. Der Sensor basiert auf der Messung der Laufzeit und verfügt über eine

Maximale Auflösung	1288×1032
Brennweite	7.5mm
Horizontaler Öffnungswinkel	65.24°
Vertikaler Öffnungswinkel	53.74°
Pixelgröße	$7.5\mu m$
Abstand der Bildmittelpunkte	90mm

Tabelle A.1: Daten der Stereo-Farbkamera

eingebaute aktive Beleuchtungseinheit. Das Messprinzip macht diesen Sensor unabhängig von der äußeren Beleuchtung. Es wird bei jeder Messung ein vollständiges und dichtes Tiefenbild zur Verfügung gestellt. Die Daten des Sensors finden sich in Tabelle A.2, Erfahrungen zu Eigenschaften und Kalibrierung in [Kahlmann 05a, Kahlmann 05b].

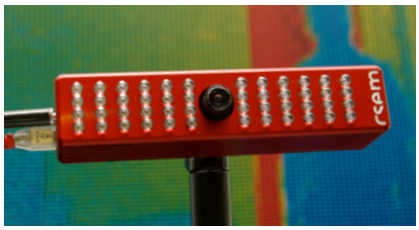


Abbildung A.2: Tiefenbildkamera Swissranger SR-2

Auflösung	160×124
Tiefenauflösung	> 5mm
Wellenlänge der Beleuchtung	870nm (infrarot)
Maximale Entfernung	7.5,
Bildrate	< 30Hz
Horizontaler Öffnungswinkel	46°
Vertikaler Öffnungswinkel	42°
Stromverbrauch	max. 18W
Brennweite	8mm
Gewicht	200g

Tabelle A.2: Daten der Tiefenbildkamera Swissranger SR-2; aus [CSEM 03].

Anhang B

Graphen und Bäume

Im Folgenden werden einige Definitionen und Eigenschaften zu Graphen und Bäumen gegeben. Diese sind swohl für die Modellierung von Makrooperatoren als auch von Flexiblen Programmen relevant.

Definition B.1 (Gerichteter nichtattributierter Graph) Seien V, W zwei nichtleere Mengen (Alphabete). Ein gerichteter nichtattributierter Graph (engl.: *u-graph*) ist ein 3-Tupel $g = (N, E, \lambda)$, wobei gilt:

- N ist eine endliche Menge von Knoten.
- $E = (E_w)_{w \in W}$ ist ein Tupel von Relationen $E_w \subseteq N \times N$ für jedes $w \in W$.
- $\lambda : N \to V$ ist eine Abbildung, die jeden Knoten mit einem Namen aus dem Alphabet V kennzeichnet.

Ein Paar $(n_1, n_2) \in E_w$ ist als eine gerichtete Kante (engl.: *edge*) von Knoten n_1 zu Knoten n_2 mit dem Label w zu interpretieren.

Definition B.2 (Gerichteter attributierter Graph) Seien V, W wie oben sowie A, B zwei nichtleere Mengen von Knoten- bzw. Kantenattributen. Ein Attribut ist hierbei als eine Funktion zu verstehen, die einen Knoten bzw. eine Kante auf einen Wert aus einer spezifischen Wertemenge abbildet. Ein gerichteter attributierter Graph (engl.: *a-graph*) ist ein 5-Tupel $(N, E, \lambda, \alpha, \beta)$, wobei gilt:

- N, E, λ wie oben.
- $\alpha : N \to 2^A$ weist jedem Knoten eine Menge von Attributen zu.
- $\beta = (\beta_w)_{w \in W}$ ist ein Tupel von Funktionen $\beta_w : E_w \longrightarrow 2^B$. Jedes β_w weist einer Kante mit dem Label w eine Menge von Attributen zu.

Wenn es nicht auf die Eigenschaft "'nichtattributiert"' oder "'attributiert"' ankommt, wird in dieser Arbeit zur Abkürzung allgemein von einem Graphen gesprochen. Graphen, bei denen die Kantenrelationen stets symmetrisch sind, werden als ungerichtet bezeichnet, denn in diesem Fall können für zwei Knoten n_1, n_2 vorhandene Kanten (n_1, n_2) und (n_2, n_1) stets zu einer einzigen ungerichteten Kante zusammengefasst werden.

Definition B.3 (Teilgraph) Seien g_1, g_2 Graphen gemäß Definition B.1 oder B.2. g_1 ist ein Teilgraph von g_2 ($g_1 \subseteq g_2$), wenn alle Knoten und Kanten von g_1 auch zu g_2 gehören und zusätzlich dieselben Labels und ggf. Attribute mit zugehörigen Werten haben.

Definition B.4 (Pfad, Zyklus) Sei g ein Graph, $n \geq 2$ und $p = (v_1, \ldots, v_n)$ eine Folge von Knoten aus N, sodass es für jedes $i \in \{1, \ldots, n-1\}$ ein $w \in W$ mit $(v_i, v_{i+1}) \in E_w$ gibt. Dann gilt:

- Sind alle Knoten aus p paarweise verschieden, so heißt p Pfad (von v_1 nach v_n).
- Sind die Knoten $v_1, \ldots, v_{n-1}$ paarweise verschieden und gilt $v_1 = v_n$, so heißt p Zyklus (mit Anfangs- und Endpunkt v_n).

Definition B.5 (Zusammenhang) Ein Graph g heißt zusammenhängend vom Knoten $v \in g$ aus, falls es von v aus zu jedem Knoten $w \in g$, $w \neq v$, einen Pfad gibt.

Definition B.6 (Baum) Ein Graph g heißt Baum, falls folgende Eigenschaften zutreffen:

- g hat einen ausgezeichneten Knoten, die Wurzel. Dieser ist nur über Ausgangskanten mit anderen Knoten verbunden.
- Sei v ein beliebiger, von der Wurzel verschiedener Knoten von g. Dann exisiert genau ein Pfad von der Wurzel nach v.

Aus dieser Definition folgt:

- Ein Baum hat keine Zyklen.
- Jeder Baum ist von seiner Wurzel aus zusammenhängend.
- Jeder von der Wurzel verschiedene Knoten v ist mit genau einem Knoten w über eine Eingangskante verbunden. w wird als Vaterknoten (oder kurz: Vater) von v bezeichnet.
- Jeder Knoten v ist mit $m \geq 0$ Knoten über eine Ausgangskante verbunden. Diese Knoten $c_1, \ldots, c_m$ heißen Kindknoten (oder kurz: Kinder) von v. Ist $m = 0$, so heißt v Blatt des Baumes. Ein Knoten, der weder Wurzel noch Blatt ist, heißt innerer Knoten des Baumes.

Anhang C

Der Iterative-Closest-Point Algorithmus

Der *Iterative-Closest-Point*-Algorithmus (Abkürzung: *ICP*) wird verwendet, um ein geometrisches Modell mit den Messungen des Objekts in Einklang zu bringen. Dafür werden zwei geordnete Punktmengen aufgebaut, die in den zwei verschiedenen Koordinatensystemen von Messung und Modell liegen. Berechnet werden die Translation $\vec{t}$ und die Rotation $\mathbf{R}$, mit der die erste Punktmenge auf die zweite transformiert werden kann. Zur Körperverfolgung besteht üblicherweise die eine Punktmenge aus Messungen des Körpers, und die zweite Menge aus Punkten auf der Oberfläche des Modells.
Nach [Besl 92] wird die erste Menge mit $P = \{\vec{p_i}\}$, und die zweite mit $X = \{\vec{x_i}\}$ bezeichnet. Beide haben die gleiche Größe $N_x = N_p = N$, wobei jeder Punkt $\vec{p_i}$ dem Punkt $\vec{x_i}$ zugeordnet ist.
Zur Bestimmung der Transformation sind mindestens 3 Punktzuordnungen notwendig. Da üblicherweise einerseits mehr als 3 Messungen zur Verfügung stehen und andererseits die Messungen verrauscht sind, wird das Problem als ein Schätzproblem betrachtet. Ziel ist, die Summe über die Quadrate der Abstände zwischen den zugeordneten Punkten zu minimieren:

$$f(\mathbf{R}, \vec{t}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{R}(\vec{x_i}) + \vec{t} - \vec{p_i}||^2 \tag{C.1}$$

Mit $\vec{\mu_p}$ and $\vec{\mu_x}$ als Mittelwert von P und N, sowie mit $\vec{p_i}' = \vec{p_i} - \vec{\mu_p}, \vec{x_i}' = \vec{x_i} - \vec{\mu_x}$ and $\vec{t'} = \vec{t} + \mathbf{R}(\vec{\mu_x}) - \vec{\mu_p}$, kann man C.1 folgendermaßen ausdrücken:

$$f(\mathbf{R}, \vec{t}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{R}(\vec{x_i}') - \vec{p_i}' + \vec{t'}||^2 \tag{C.2}$$

Dies wird zu:

$$f(\mathbf{R}, \vec{t}) = \frac{1}{N} \Big(\sum_{i=1}^{N} ||\mathbf{R}(\vec{x_i}') - \vec{p_i}'||^2 - 2\vec{t} \cdot \sum_{i=1}^{N} (\mathbf{R}(\vec{x_i}') - \vec{p_i}') + N||\vec{t'}||^2 \Big) \tag{C.3}$$

Der erste Teil ist unabhängig von $\vec{t'}$, der zweite ist Null. Deshalb ist $f(\mathbf{R}, \vec{t})$ dann minimal, wenn $\vec{t'} = 0$ gilt. Dies ergibt:

$$\vec{t} = \vec{\mu_p} - \mathbf{R}(\vec{\mu_x}). \tag{C.4}$$

$\vec{t}$ stellt die gesuchte Translation dar. Damit kann man Gleichung C.2 folgendermaßen schreiben:

$$f(\mathbf{R}, \vec{t}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{R}(\vec{x_i}') - \vec{p_i}'||^2 \tag{C.5}$$

Mit $||\mathbf{R}(\vec{x_i}')|| = ||\vec{x_i}'||$, kann dies folgendermaßen formuliert werden:

$$f(\mathbf{R}, \vec{t}) = \frac{1}{N} \Big(\sum_{i=1}^{N} ||\vec{x_i}'||^2 - 2 \cdot \sum_{i=1}^{N} \mathbf{R}(\vec{x_i}') \cdot \vec{p_i}' + \sum_{i=1}^{N} ||\vec{p_i}'||^2 \Big) \tag{C.6}$$

Die Maximierung von

$$\sum_{i=1}^{N} \mathbf{R}(\vec{x_i}') \cdot \vec{p_i}' \tag{C.7}$$

ergibt dann die optimale Rotation $\mathbf{R}$ (siehe [Horn 87]).
Zur Berechnung der Punktzuordnung zwischen P und N müssen allerdings zunächst die Messungen dem Modell zugeordnet werden. Ausgangsdaten sind ungeordnete Messungen sowie eine geometrische Beschreibung der Oberfläche des Modells. Die Berechnung erfolgt über geometrische Betrachtungen, wobei für jeden Punkt der Messung der nächste Punkt (engl.: *Closest Point*) auf der Modelloberfläche bestimmt wird. Dadurch erhält man die Zuordnung zwischen jedem $\vec{p_i}$ und einem $\vec{x_i}$. Diese Berechnung wird in jedem Schritt wiederholt, da sie nur eine Schätzung der Korrespondenz aufgrund geometrischer Annahmen darstellt.
Der vollständige Algorithmus läuft also folgendermaßen ab:

1. Berechnung der Punktbeziehungen auf Basis der aktuellen Messungen und des aktuellen Modells, um die Menge der zugeordneten Modellpunkte X_0 zu erhalten.
2. Berechnung der Summe über die Quadrate der Abstände $d_0(P, X_0)$.
3. Schätzung der Transformation, und Anwendung der Transformation auf das Modell.
4. Neuberechnung der Punktbeziehungen auf Basis der neuen Modellposition zu X_i.
5. Berechnung der Summe über die Quadrate der Abstände zu $d_i(P, X_i)$.
6. Bei erfülltem Abbruchkriterium $d_{i-1}(P, X_{i-1}) - d_i(P, X_i) < \epsilon$ wird abgebrochen, ansonsten weiter bei Schritt 3.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[Alami 00] R. Alami, R. Chatila, S. Fleury, M. Herrb, F. F. Ingrand, M. Khatib, B. Morisset, P. Moutarlier, T. Siméon. Around the lab in 40 days ... Tagungsband: *Proc. 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[Alami 06] R. Alami, R. Chatila, A. Clodic, S. Fleury, M. Herrb, V. Montreuil, E. A. Sisbot. Towards human-aware cognitive robots. Tagungsband: *Proceedings of the The Fifth International Cognitive Robotics Workshop (The AAAI-06 Workshop on Cognitive Robotics, COGROB 2006)*, Boston, Massachusetts, USA, 2006.

[Alami 98] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[Albiez 02] J. Albiez, T. Luksch, K. Berns, R. Dillmann. An activation based behaviour control architecture for walking machines. Tagungsband: *ICSAB: Proceedings of the seventh international conference on simulation of adaptive behavior on From animals to animats*, Seiten 118–126, Cambridge, MA, USA, 2002. MIT Press.

[Albus 88] J. Albus, R. Lumia, H. McCain. Hierarchical control of intelligent machines applied to space station telerobots. Tagungsband: *Transactions on Aerospace and Electronic Systems*, Band 24, Seiten 535–541, September 1988.

[Albus 89] J. Albus, H. McCain, R. Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). Technischer Bericht 1235, Nat'l Inst. Standards and Technology, Gaithersburg, 1989.

[Alissandrakis 02a] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn. Do as I do: correspondences across different robotic embodiments. In D. Polani, J. Kim, T. Martinetz, Hrsg., Tagungsband: *Proc. Fifth German Workshop on Artificial Life*, Seiten 143–152, 2002.

[Alissandrakis 02b] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn. Imitation with ALICE: Learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 32(4):482–496, 2002.

[Alissandrakis 02c] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn. Do as I Do: Correspondences across Different Robotic Embodiments. In D. Polania, J. Kim, T. Martinetz, Hrsg., Tagungsband: *Proc. Fifth German Workshop on Artificial Life (GWAL5)*, Seiten 143–152, März 2002.

[Alissandrakis 03] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn. Solving the correspondence problem between dissimilarly embodied robotic arms using the ALICE imitation mechanism. In T. S. for the Study of Artificial Intelligence, S. of Behaviour, Hrsg., Tagungsband: *Proceedings of the Second International Symposium on Imitation in Animals and Artifacts*, Seiten 79–92, 2003.

[Alissandrakis 06] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn. Action, state and effect metrics for robot imitation. Tagungsband: *Proceedings of the 15th IEEE International*

Symposium on Robot and Human Interactive Communication, University of Hertfordshire, Hatfield, United Kingdom, September 2006.

[Amtec 00] Amtec. *Products: Rotary*, 2000. http://www.amtec-robotics.com.

[Andersen 05] H.-C. Andersen. Generierung von ausführbaren Roboterprogrammen aus Makrooperatoren. Diplomarbeit, Universität Karlsruhe (TH), Oktober 2005.

[Anderson 04] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, Oktober 2004.

[Anderson 98] J. R. Anderson, C. Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum Associates, Mahwah, NJ, Juni 1998.

[Asimov 42] I. Asimov. Runaround. In *Astounding Science Fiction magazine*. John W. Campbell, März 1942.

[Au 06] C. Au. Eine Komponente zur Handlungsauswahl für den Serviceroboter Albert. Studienarbeit, Universität Karlsruhe (TH), IAIM, Institut für Technische Informatik, Juli 2006.

[Azad 04] P. Azad, A. Ude, R. Dillmann, G. Cheng. A full body human motion capture system using particle filtering and on-the-fly edge detection. Tagungsband: *Proceedings of IEEE-RAS/RSJ International Conference on Humanoid Robots*, Santa Monica, USA, 2004.

[Bar 06] Barrett Technologies. *Homepage von Barrett Technologies*, 2006. http://www.barrett.com/robot/index.htm.

[Battiti 94] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, Juli 1994.

[Becher 06] R. Becher, P. Steinhaus, R. Zöllner, R. Dillmann. Design and implementation of an interactive object modelling system. Tagungsband: *Robotik/ISR*, München, Germany, Mai 2006.

[Bertram 05] D. Bertram. Modellierung und Überprüfung von Bedingungen zur Aufgabenmodellierung für einen Serviceroboter. Studienarbeit, Universität Karlsruhe, 2005.

[Besl 92] P. J. Besl, N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, February 1992.

[Billard 04] A. Billard, R. Siegwart. Robot learning from demonstration. *Robotics and Autonomous Systems*, 47:65–67, 2004.

[Blum 03] S. A. Blum. From a corba-based software framework to a component-based system architecture for controlling a mobile robot. Tagungsband: *International Conference of Computer Vision Systems (ICVS 2003)*, Seiten 333–344, Graz, Austria, April 2003.

[Bonasso 97] R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, M. G. Slack. Experiences with an architecture for intelligent, reactive agents. Tagungsband: *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, Band 9, Seiten 237–256, 1997.

[Brännström 06] S. Brännström. Extraction, evaluation and selection of motion features for human activity recognition purposes. Diplomarbeit, Kungl. Techniska Högskolan (KTH), 2006.

[Breazeal 00] C. L. Breazeal. *Sociable machines: expressive social exchange between humans and robots*. Dissertation, Department of Electrical Engineering and Computer Science, MIT, 2000. Supervisor R. A. Brooks.

[Breazeal 02] C. L. Breazeal, B. Scassellati. Challenges in building robots that imitate people. *Imitation in animals and artifacts*, Seiten 363–390, 2002.

[Breazeal 03] C. L. Breazeal, B. Scassellati. Robots that imitate humans. Technischer Bericht, The Media Lab, Massachusetts Institute of Technology, 77 Massachusetts Ave NE18-5FL, Cambridge MA 02139, USA, 2003.

[Brockhaus 90] *Brockhaus Enzyklopädie*. F. A. Brockhaus AG. 19. Auflage, Band 12, Seite 141, Mannheim, 1990.

[Brooks 86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, März 1986.

[Brooks 90] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, Juni 1990.

[Brooks 99] R. A. Brooks, C. L. Breazeal, M. Marjanovic, B. Scassellati, M. Williamson. The Cog Project: Building a Humanoid Robot. In C. Nehaniv, Hrsg., *Lecture Notes in Artificial Intelligence 1562*, Seiten 52–87. Springer, 1999.

[Burghart 05] C. Burghart, R. Mikut, R. Stiefelhagen, T. Asfour, H. Holzapfel, P. Steinhaus, R. Dillmann. A cognitive architecture for a humanoid robot: A first approach. Tagungsband: *Proceedings of 2005 5th IEEE-RAS International Conference on Humanoid Robots*, Tsukuba, Japan, 2005. IEEE Institute of Electrical and Electronics Engineers.

[Calinon 05] S. Calinon, F. Guenter, A. Billard. Goal-Directed Imitation in a Humanoid Robot. Tagungsband: *Proceedings of the International Conference on Robotics and Automation (ICRA)*, April 18-22 2005.

[Chella 05] A. Chella, H. Dindo, I. Infantino. A Cognitive Framework for Learning by Imitation. Tagungsband: *Proc of the IEEE ICRA Workshop on The Social Mechanisms of Robot Programming by Demonstration*, Barcelona, Spain, 2005.

[Chil 06] CHIL: Computers in the Human Interaction Loop. Integrated Project, founded by the EU. Webseite. http://chil.server.de/, Dezember 2006.

[Cogniron 06] Cogniron. "The Cognitive Robot Companion", EU-Projekt, gefördert unter FP6-IST-002020. Webseite http://www.cogniron.org, 2006.

[Connell 89] J. Connell. A colony architecture for an artificial creature. Technischer Bericht, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, 1989.

[Coradeschi 00] S. Coradeschi, A. Saffiotti. Anchoring symbols to sensor data: preliminary report. Tagungsband: *Proc. of the 17th AAAI Conference*, Seiten 129–135, Menlo Park, California, 2000.

[Coradeschi 01] S. Coradeschi, A. Saffiotti. Perceptual anchoring of symbols for action. Tagungsband: *Proceedings of the International Conference on Artificial Intelligence*, Seiten 407–412, Seattle, Washington, USA, 2001.

[Coste-Maniere 00] E. Coste-Maniere, R. Simmons. Architecture, the backbone of robotic systems. Tagungsband: *Proc. 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[Coutaz 05] J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan. Context is key. *Communincations ACM*, 48(3):49–53, 2005.

[Crowley 04] J. L. Crowley, J. Coutaz. Context Aware Observation of Human Activity. Tagungsband: *PSIPS 2004, Oulu, Finland*, Juni 2004.

[CSEM 03] CSEM. *Swissranger SR-2 Data Sheet*. CSEM, Technologies for Innovation, Badenerstrasse 569, CH-8048 Zurich, Switzerland, 2003.

[CSEM 06] CSEM. *Swissranger Webseite*, 2006. http://www.swissranger.ch/.

[Cutkosky 89] M. R. Cutkosky. On Grasp Choice, Grasp Models, and the Design of Hands for Manufacturing Tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269–279, 1989.

[Demirdjian 02] D. Demirdjian, T. Darrell. 3-D articulated pose tracking to untethered deictic references. Tagungsband: *Multimodel Interfaces*, Seiten 267–272, 2002.

[Demirdjian 03] D. Demirdjian. Enforcing constraints for human body tracking. Tagungsband: *2003 Conference on Computer Vision and Pattern Recognition Workshop Vol. 9*, Seiten 102–109, Madison, Wisconsin, USA, 2003.

[Dey 01] A. K. Dey. Understanding and using context. Tagungsband: *Proceedings of Personal and Ubiquitous Computing*, 2001.

[Dillmann 99] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zöllner, M. Bordegoni. Learning Robot Behaviour and Skills based on Human Demonstration and Advice: the Machine Learning Paradigm. Tagungsband: *9th International Symposium of Robotics Research (ISRR 1999)*, Seiten 229–238, Snowbird, Utah, USA, 9.-12. Oktober 1999.

[Ehrenmann 01a] M. Ehrenmann, O. Rogalla, R. Zöllner, R. Dillmann. Teaching service robots complex tasks: Programming by demonstration for workshop and household environments. Tagungsband: *Proceedings of the 2001 International Conference on Field and Service Robots (FSR)*, Band 1, Seiten 397–402, Helsinki, Finnland, 11.–13. Juni 2001.

[Ehrenmann 01b] M. Ehrenmann, R. Zöllner, S. Knoop, R. Dillmann. Sensor Fusion Approaches for Observation of User Actions in Programming by Demonstration. Tagungsband: *Proceedings of the 2001 International Conference on Multi Sensor Fusion and Integration for Intelligent Systems (MFI)*, Band 1, Seiten 227–232, Baden-Baden, 19.–22. August 2001.

[Ehrenmann 02] M. Ehrenmann. *Handlungsbeobachtung zur Instruierung von Robotersystemen.* Dissertation, Universität Karlsruhe (TH), 2002.

[Ehrenmann 03] M. Ehrenmann, R. Zöllner, O. Rogalla, S. Vacek, R. Dillmann. Observation in programming by demonstration: Training and exection environment. Tagungsband: *Humanoids 2003, Karlsruhe/Munich, Germany, Oktober 2003*, 2003.

[Ekvall 06a] S. Ekvall, D. Kragic. Learning task models from multiple human demonstrations. Tagungsband: *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, University of Hertfordshire, UK, 2006.

[Ekvall 06b] S. Ekvall, D. Kragic. Task learning using graphical programming and human demonstrations. Tagungsband: *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, University of Hertfordshire, UK, 2006.

[Elinas 02] P. Elinas, J. Hoey, D. Lahey, J. D. Montgomery, D. Murray, S. Se, J. J. Little. Waiting with José, a vision-based mobile robot. Tagungsband: *Proc. 2002 IEEE International Conference on Robotics and Automation*, Washington, D.C., May 2002.

[Erol 94a] K. Erol, J. Hendler, D. Nau. Semantics for hierarchical task-network planning. Technischer Bericht CS-TR-3239, Computer Science Dept., University of Maryland, 1994.

[Erol 94b] K. Erol, J. Hendler, D. S. Nau. HTN planning: Complexity and expressivity. Tagungsband: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Band 2, Seiten 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[Erol 94c] K. Erol, J. A. Hendler, D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. Tagungsband: *Artificial Intelligence Planning Systems*, Seiten 249–254, 1994.

[Erol 96] K. Erol, J. A. Hendler, D. S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Fikes 71] R. Fikes, N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Tagungsband: *IJCAI*, Seiten 608–620, 1971.

[Finkemeyer 05] T. Finkemeyer, B.; Kröger, F. Wahl. Aktionsprimitive: Ein universelles Roboter-Programmierparadigma. *at - Automatisierungstechnik*, 04:189–196, 2005.

[Firby 01] R. J. Firby. The RAP Language Manual, Version 2. Technischer Bericht, I/NET Inc., 2001.

[Fleury 97] S. Fleury, M. Herrb, R. Chatila. GenoM: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. Tagungsband: *International Conference on Intelligent Robots and Systems*, Band 2, Seiten 842–848, Grenoble (France), September 1997. IEEE.

[Fontmarty 07] M. Fontmarty, T. Germa, B. Burger, L. F. Marin, S. Knoop. Implementation of human perception algorithms on a mobile robot. Tagungsband: *Submitted to the Conference on Intelligent Autonomous Vehicles*, Toulouse, France, 2007.

[Friedrich 98] H. Friedrich. *Interaktive Programmierung von Manipulationssequenzen.* Dissertation, Universität Karlsruhe, 1998.

[Friedrich 99] H. Friedrich, V. Grossmann, M. Ehrenmann, O. Rogalla, R. Zöllner, R. Dillmann. Towards cognitive elementary operators: grasp classification using neural network classifiers. Tagungsband: *Proceedings of the IASTED International Conference on Intelligent Systems and Control (ISC)*, Band 1, Santa Barbara, Kalifornien, USA, 28.-30. Oktober 1999.

[Fritsch 03] J. Fritsch, M. Kleinehagenbrock, S. Lang, T. Plötz, G. A. Fink, G. Sagerer. Multi-Modal Anchoring for Human-Robot-Interaction. *Robotics and Autonomous Systems, Special issue on Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems*, 43(2–3):133–147, 2003.

[Fügen 06] C. Fügen, P. Gieselmann, H. Holzapfel, F. Kraft. Natural human robot communication. Tagungsband: *Proceedings of the Workshop on Human Centered Robotic Systems (HCRS)*, München, Germany, 2006.

[Gamma 04] E. Gamma, R. Helm, R. E. Johnson. *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software.* Addison Wesley in Pearson Education Deutschland, ISBN 3-8273-2199-9, 2004.

[Gat 97a] E. Gat. ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents. Tagungsband: *Proceedings of the Aerospace Conference, 1997. IEEE*, Snowmass at Aspen, CO, USA, 1997.

[Gat 97b] E. Gat. On three-layer architectures. *Artificial Intelligence and Mobile Robots. MIT/AAAI Press*, 1997.

[Ghallab 98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

[Giesler 05] B. Giesler. *Erweiterte Realität in der Zusammenarbeit mit Servicerobotern.* Dissertation, Universität Karlsruhe (TH), 2005.

[Guyon 03] I. Guyon, A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3(1):1157–1182, 2003.

[Hanheide 06] M. Hanheide. *A Cognitive Ego-Vision System for Interactive Assistance.* Doktorarbeit, Technische Fakultat der Universitat Bielefeld, Bielefeld, Oktober 2006.

[Hart 04] S. Hart, R. Grupen, D. Jensen. A Relational Representation for Generalized Knowledge in Robotic Tasks. Technischer Bericht 04-101, Computer Science Department, University of Massachusetts Amherst, 2004.

[Hart 05] S. Hart, R. Grupen, D. Jensen. A Relational Representation for Procedural Task Knowledge. Tagungsband: *Proceedings of the 2005 American Association for Artificial Intelligence (AAAI) Conference*, Pittsburgh, Pennsylvania, 2005.

[Hartley 91] R. Hartley, F. Pipitone. Experiments with the Subsumption Architecture. Tagungsband: *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 1991.

[Hoffmann 01] J. Hoffmann. FF: The Fast-Forward Planning System. *The AI Magazine*, 2001. Forthcoming.

[Horn 87] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Optical Society of America Journal A*, 4:629–642, April 1987.

[ICA 06] ICAPS Conference. *International Planning Competition*, 2006.

[Ingrand 96] F. F. Ingrand, R. Chatila, R. Alami, F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. Tagungsband: *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, USA, 1996.

[Jensen 02a] B. Jensen, G. Froidevaux, X. Greppin, A. Lorotte, L. Mayor, M. Meisser, G. Ramel, R. Siegwart. The interactive autonomous mobile system RoboX. Tagungsband: *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Seiten 1221–1227. IEEE, 2002. EPFL, Lausanne, Switzerland.

[Jensen 02b] B. Jensen, G. Froidevaux, X. Greppin, A. Lorotte, L. Mayor, M. Meisser, G. Ramel, R. Siegwart. Visitor Flow Management using Human-Robot Interaction at Expo.02. Tagungsband: *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2002)*, 2002.

[Kaelbling 98] L. P. Kaelbling, M. L. Littman, A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Kahlmann 05a] T. Kahlmann, H. Ingensand. Range Imaging Sensor Properties and Calibration. Tagungsband: *Proceedings of the 1st Range Imaging Day*, ETH Zürich, Switzerland, September 2005.

[Kahlmann 05b] T. Kahlmann, T. Oggier, F. Lustenberger, N. Blanc, H. Ingensand. 3D-TOF sensors in the automobile. Tagungsband: *Proceedings of SPIE*, Band 5663, Seiten 216–224, 2005.

[Kaiser 96] M. Kaiser. *Interaktive Akquisition elementarer Roboterfähigkeiten.* Dissertation, Universität Karlsruhe (TH), 1996.

[Kang 97] S. Kang, K. Ikeuchi. Toward Automatic Robot Instruction from Perception: Mapping Human Grasps to Manipulator Grasps. *Robotics and Automation*, 13(1):81–95, Februar 1997.

[Kaptelinin 95] V. Kaptelinin, K. Kuutti, L. J. Bannon. Activity theory: Basic concepts and applications. Tagungsband: *EWCHI '95: Selected papers from the 5th International Conference on Human-Computer Interaction*, Seiten 189–201, London, UK, 1995. Springer-Verlag.

[Kellokumpu 05] V. Kellokumpu, M. Pietikäinen, J. Heikkilä. Human activity recognition using sequences of postures. Tagungsband: *Proc. IAPR Conference on Machine Vision Applications (MVA 2005)*, Seiten 570–573, Tsukuba Science City, Japan, 2005.

[Kleinehagenbrock 02] M. Kleinehagenbrock, S. Lang, J. Fritsch, F. Lömker, G. A. Fink, G. Sagerer. Person tracking with a mobile robot based on multi-modal anchoring. Tagungsband: *In Proc. IEEE Int. Workshop on Robot and Human Interactive Communication (ROMAN)*, Berlin, Germany, September 2002. IEEE.

[Knoop 04] S. Knoop, S. Vacek, R. Zöllner, C. Au, R. Dillmann. A CORBA-Based Distributed Software Architecture for Control of Service Robots. Tagungsband: *Proceedings of the IEEE International Conference on Intelligent Robots and System*, Seiten 3656–3661, Sendai, Japan, 2004. IEEE Institute of Electrical and Electronics Engineers.

[Knoop 05a] S. Knoop, S. Vacek, R. Dillmann. Modeling Joint Constraints for an Articulated 3D Human Body Model with Artificial Correspondences in ICP. Tagungsband: *Proceedings of the International Conference on Humanoid Robots (Humanoids 2005)*, Tsukuba, Japan, 2005. IEEE-RAS.

[Knoop 05b] S. Knoop, M. Pardowitz, R. Dillmann. *Programming by Demonstration - Interaction, Learning and Execution*, Kapitel Humanoid Robots, Seiten 19–35. Buch zur Summerschool "Guidance and Control of Autonomous Systems" Duisburg/Essen, 2005.

[Knoop 06a] S. Knoop, S. R. Schmidt-Rohr, R. Dillmann. A Flexible Task Knowledge Representation for Service Robots. Tagungsband: *The 9th International Conference on Intelligent Autonomous Systems (IAS-9)*, Kashiwa New Campus, The University of Tokyo, Tokyo, Japan, März 2006.

[Knoop 06b] S. Knoop, S. Vacek, R. Dillmann. Sensor fusion for 3d human body tracking with an articulated 3d body model. Tagungsband: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, Florida, 2006.

[Knoop 06c] S. Knoop, S. Vacek, K. Steinbach, R. Dillmann. Sensor fusion for model based 3d tracking. Tagungsband: *Proceedings of the IEEE International Conference on Multi Sensor Fusion and Integration (MFI)*, 2006.

[Knoop 07a] S. Knoop, M. Pardowitz, R. Dillmann. From abstract task knowledge to executable robot programs. *to appear in: Journal of Intelligent and Robotic Systems: Theory and Applications. Robotics, Virtual Reality, and Agents and their Body: A Special Issue in Memory of Marco Somalvico*, 2007.

[Knoop 07b] S. Knoop, S. Vacek, R. Dillmann. *Human Body Model for Articulated 3D Pose Tracking*, Kapitel 28. International Journal of Advanced Robotic Systems, Special Issue on Humanoid Robots, 2007.

[Kohavi 97] R. Kohavi, G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[Konolige 97] K. Konolige, K. L. Myers, E. H. Ruspini, A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235, 1997.

[Kramer 06] T. Kramer. Pfadplanung für einen redundanten manipulator mittels entkoppelter potentialfelder. Diplomarbeit, Universität Karlsruhe (TH), Mai 2006.

[Krum 05] U. Krum, H. Holzapfel, A. Waibel. Clarification questions to improve dialogue flow and speech recognition in spoken dialogue systems. Tagungsband: *Proceedings of Interspeech'05*, Lisbon, 2005.

[Kwak 99] N. Kwak, C.-H. Choi. Improved Mutual Information Feature Selector for Neural Networks in Supervised Learning. Tagungsband: *Proceedings of the International Joint Conference on Neural Networks, 1999*, Band 2, Seiten 1313–1318, 1999.

[Laird 87] J. E. Laird, A. Newell, P. S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Langley 05] P. Langley, J. E. Laird, S. Rogers. Cognitive Architectures: Research Issues and Challenges. Technischer Bericht, Computational Learning Laboratory Center for the Study of Language and Information Stanford University, Stanford, CA 94305, 2005.

[Langley 06] P. Langley. Cognitive Architectures and General Intelligent Systems. Technischer Bericht, Computational Learning Laboratory, CSLI, Stanford University, CA, 2006.

[Lewis 99a] R. L. Lewis. Cognitive Theory, SOAR, Oktober 1999.

[Lewis 99b] R. Lewis. Cognitive modeling, symbolic. *The MIT Encyclopedia of the Cognitive Sciences*, 1999.

[Long 05] F. Long, C. Ding. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 27(8):1226–1238, 2005. Member-Hanchuan Peng.

[Lösch 05] M. Lösch. Erkennung statischer Zeigegesten in Farbbildern. Studienarbeit, Universität Karlsruhe (TH), Mai 2005.

[Mallet 02] A. Mallet, S. Fleury, H. Bruyninckx. A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. Tagungsband: *International Conference on Intelligent Robotics and Systems*, Lausanne (Switzerland), Oktober 2002. IEEE.

[Matarić 02] M. J. Matarić. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. *Imitation in animals and artifacts*, Seiten 391–422, 2002.

[Meltzoff 02] A. N. Meltzoff. Elements of a developmental theory of imitation. *The imitative mind: Development, evolution, and brain bases. Cambridge: Cambridge University Press*, Seiten 19–41, 2002.

[Meusel 01] P. Meusel. *Nachgiebiger Kraft-Momenten-Sensor*. DLR Institut für Robotik und Mechatronik, 2001.

[Miller 03] A. T. Miller, S. Knoop, H. I. Christensen, P. K. Allen. Automatic grasp planning using shape primitives. Tagungsband: *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2003.

[Minar 04] J. Minar. Visualisierung von Zuständen im Serviceroboter Albert. Diplomarbeit, Universität Karlsruhe (TH), Institut für Rechnerentwurf und Fehlertoleranz, Fakultät für Informatik, 2004.

[Nau 03] D. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, F. Yaman. SHOP2: An HTN Planning System. *Journal on Artificial Intelligence Research*, 20, 2003.

[Nehaniv 02] C. L. Nehaniv, K. Dautenhahn. The Correspondence Problem in Social Learning: What Does it Mean for Behaviors to Match Anyway? *Perspectives on Imitation: From Cognitive Neuroscience to Social Science*, 2002.

[Nickel 04a] K. Nickel, E. Seemann, R. Stiefelhagen. 3D-Tracking of Head and Hands for Pointing Gesture Recognition in a Human-Robot Interaction Scenario. Tagungsband: *Proceedings of the Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FGR)*, Seiten 565–570, 2004.

[Nickel 04b] K. Nickel, R. Stiefelhagen. Real-Time Person Tracking and Pointing Gesture Recognition for Human-Robot Interaction. Tagungsband: *ECCV Workshop on HCI*, Seiten 28–38, 2004.

[Nicolescu 02] M. N. Nicolescu, M. J. Matarić. A Hierarchical Architecture for Behavior-Based Robots, 2002.

[Nicolescu 03] M. N. Nicolescu, M. J. Matarić. Natural methods for robot task learning: Instructive demonstrations, generalization and practice, 2003.

[Nilsson 80] N. J. Nilsson. *Principles of artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[Oliver 02] N. Oliver, E. Horvitz, A. Garg. Layered representations for human activity recognition. Tagungsband: *ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces*, Seite 3, Washington, DC, USA, 2002. IEEE Computer Society.

[Otero 06a] N. Otero, S. Knoop, C. L. Nehaniv, D. Syrdal, K. Dautenhahn, R. Dillmann. Distribution and recognition of gestures in human-robot interaction. Tagungsband: *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, Piscataway, NJ, USA, September 2006. IEEE Institute of Electrical and Electronics Engineers.

[Otero 06b] N. Otero, C. L. Nehaniv, D. Syrdal, K. Dautenhahn. Naturally occurring gestures in a human-robot teaching scenario: an exploratory study comparing the use of gestures only or gestures and speech. Technical Report 450, School of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire, 2006.

[Pardowitz 05] M. Pardowitz, R. Zöllner, R. Dillmann. Learning sequential constraints of tasks from user demonstrations. Tagungsband: *Proc. IEEE-RAS Intl. Conf. on Humanoid Robots (HUMANOIDS), Tsukuba, Japan*, Seiten 424 – 429, 2005.

[Pardowitz 06a] M. Pardowitz, R. Zöllner, R. Dillmann. Incremental learning of task sequences with information-theoretic metrics. Tagungsband: *Proceedings of the European Robotics Symposium (EUROS06)*, Seiten 254–266, März 2006.

[Pardowitz 06b] M. Pardowitz, R. Zöllner, S. Knoop, R. Dillmann. Using Physical Demonstrations, Background Knowledge and Vocal Comments for Task Learning. Tagungsband: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.

[Pembeci 02] I. Pembeci, G. D. Hager. A comparative review of robot programming languages. Technischer Bericht, CIRL Lab Technical Report, 2002.

[Perception 07] D. Perception. Homepage http://www.dperception.com/. Webseite, Februar 2007.

[Peters II 03] R. A. Peters II, C. L. Campbell, W. J. Bluethmann, E. Huber. Robonaut Task Learning through Teleoperation. Tagungsband: *Robonaut Task Learning through Teleoperation*, 2003.

[Rabiner 89] L. Rabiner. A tutorial on Hidden Markov Models and selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–285, Februar 1989.

[Rao 04] R. Rao, A. Shon, A. Meltzoff. A Bayesian model of imitation in infants and robots, 2004.

[Rao 99] A. Rao. Report of Project II: Hidden Markov Model (HMM). Technischer Bericht CSE655 Pattern Recognition, State University of New York at Buffalo, Mai 1999.

[Rogalla 02] O. Rogalla. *Abbildung von Benutzerdemonstrationen auf variable Roboterkonfigurationen*. Dissertation, Institut für Technische Informatik, Universität Karlsruhe (TH), 2002.

[Salb 03a] T. Salb. *Risikoreduktion in der Mund-Kiefer-Gesichts-Chirurgie mittels rechnerbasierter Modellierung und Erweiterter Realität*. Dissertation, Universität Karlsruhe (TH), 2003.

[Salb 03b] T. Salb, J. Brief, B. Giesler, T. Welzel, T. Gockel, S.Hassfeld, J.Mühling, R.Dillmann. INPRES (intraoperative presentation of surgical planning and simulation results) - augmented reality for craniofacial surgery. Tagungsband: *Proceedings of Spie - Medical Imaging: Visualization, Image-Guided Procedures, and Display*, Band 5744, Bellingham, 2003. SPIE Press.

[Schaal 03] S. Schaal, A. Ijspeert, A. Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions: Biological Sciences*, 358(1431):537–547, 2003.

[Schaude 96] H. Schaude. Dokumentation zu KaVis. Technischer Bericht, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe (TH), 1996.

[Schmidt-Rohr 05] S. R. Schmidt-Rohr. Hierarchische Repräsentation und dynamische Ausführung von Handlungswissen für einen autonomen Agenten. Studienarbeit, Universität Karlsruhe, 2005.

[Schmidt-Rohr 06] S. R. Schmidt-Rohr. High level planning considering uncertainty on service robots with human-robot interaction. Diplomarbeit, Universität Karlsruhe (TH), 2006.

[Schrempf 05] O. C. Schrempf, U. D. Hanebeck. A generic model for estimating user intentions in human-robot cooperation. Tagungsband: *Proceedings of the 2nd Intl. Conference on Informatics in Control, Automation and Robotics (ICINCO 2005)*, 3, Seiten 251–256, Barcelona, Spanien, September 2005.

[Schwarz 98] E. Schwarz. Makrogenerierung aus Benutzervorführungen. Diplomarbeit, Institut für Prozessrechentechnik und Robotik (IPR), Universität Karlsruhe (TH), 1998.

[Sen 04] A. Sen. Kismet und Emotionen. Ausarbeitung im Rahmen des Seminars Humanoide Roboter, Universität Bielefeld, 2004.

[Shon 04] A. P. Shon, D. B. Grimes, C. L. Baker, R. P. Rao. A probabilistic framework for model-based imitation learning. Tagungsband: *CogSci*, 2004.

[SICK 06] SICK. *SICK Homepage*, 2006. http://www.sick.com.

[Sidenbladh 00] H. Sidenbladh, M. J. Black, D. J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. Tagungsband: *Proceedings of the 6th European Conference on Computer Vision-Part II*, Seiten 702–718, 2000.

[Sidenbladh 01] H. Sidenbladh. *Probabilistic Tracking and Reconstruction of 3D Human Motion in Monocular Video Sequences*. Dissertation, KTH, Stockholm, Sweden, 2001.

[Siegwart 03] R. Siegwart, K. O. Arras, S. Bouabdhalla, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, N. Tomatis. Robox at expo.02: A Large Scale Installation of Personal Robots. *Special issue on Socially Interactive Robots, Robotics and Autonomous Systems*, 42:203–222, 2003.

[Simmons 02] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. T. Stentz, R. M. Zlot. A layered architecture for coordination of mobile robots. Tagungsband: *Multi-Robot Systems: From Swarms to Intelligent Automata, Proceedings from the 2002 NRL Workshop on Multi-Robot Systems*. Kluwer Academic Publishers, Mai 2002.

[Simmons 03] R. Simmons, D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A. Schultz, M. Abramson, W. Adams, A. Atrash, M. Bugajska, M. Coblenz, M. MacMahon, D. Perzanowski, I. Horswill, R. Zubek, D. Kortenkamp, B. Wolfe, T. Milam, B. Maxwell. GRACE: An autonomous robot for the AAAI Robot Challenge. *AI Magazine*, 24(2):51–72, 2003.

[Simmons 94] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), Februar 1994.

[Simmons 98] R. Simmons, D. Apfelbaum. A Task Description Language for Robot Control. Tagungsband: *Proceedings of the Conference on Intelligent Robotics and Systems*, 1998.

[Spexard 06] T. Spexard, A. Haasch, J. Fritsch, G. Sagerer. Human-like person tracking with an anthropomorphic robot. Tagungsband: *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Seiten 1286–1292, Orlando, Florida, Mai 2006. IEEE.

[Stein 06] T. Stein, A. Fischer, K. Bös, V. Wank, I. Boesnach, J. Moldenauer. Guidelines for motion control of humanoid robots: Analysis and modeling of human movements. *International Journal of Computer Science in Sports*, 5(1), August 2006.

[Steinbach 05] K. Steinbach. Erweiterung des ICP-Algorithmusses zur 3D-Posenschätzung von Personen. Studienarbeit, Universität Karlsruhe (TH), 2005.

[Stiefelhagen 04] R. Stiefelhagen, C. Fuegen, P. Gieselmann, H. Holzapfel, K. Nickel, A. Waibel. Natural human-robot interaction using speech, gaze and gestures. Tagungsband: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004.

[The SOAR group 06] The SOAR group. The soar homepage. Website, http://sitemaker.umich.edu/soar/home, Dezember 2006.

[Thrun 99a] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, D. Schulz. Minerva: A second generation mobile tour-guide robot. Tagungsband: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[Thrun 99b] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. R. Rosenberg, N. Roy, J. Schulte, D. Schulz. MINERVA: A tour-guide robot that learns. Tagungsband: *KI - Kunstliche Intelligenz*, Seiten 14–26, 1999.

[Ude 96] A. Ude. *Rekonstruktion von Trajektorien aus Stereobildfolgen für die Programmierung von Roboterbahnen*. Dissertation, Universität Karlsruhe, 1996. Erschienen in: VDI Verlag, Fortschr. Ber. VDI Reihe 10 Nr. 448. Düsseldorf.

[Vacek 05] S. Vacek, S. Knoop, R. Dillmann. Classifying Human Activities in Household Environments. Tagungsband: *Modelling Others from Observation (MOO): Workshop at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[Veloso 95] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[Vicon 06] Vicon. *Vicon Motion Capture Systems*, 2006. http://www.vicon.com.

[Vid 06] Videre Design. *Videre Design Homepage*, 2006. http://www.videredesign.com/.

[Videre Design 01] Videre Design. *STH-MD1/-C Stereo Head User's Manual*. http://www.videredesign.com/manuals.htm, 2001.

[Viola 01] P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. Tagungsband: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

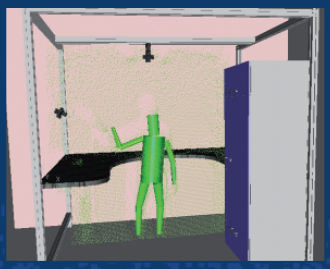
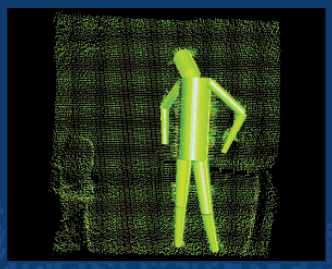
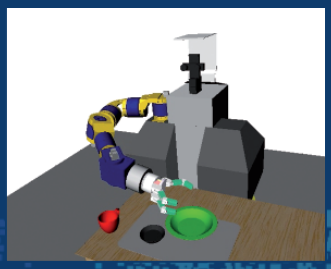
[Weber 00] S. Weber, M. J. Matarić, O. C. Jenkins. Experiments in imitation using perceptuo-motor primitives. In C. Sierra, M. Gini, J. S. Rosenschein, Hrsg., Tagungsband: *Proceedings of the Fourth International Conference on Autonomous Agents*, Seiten 136–137, Barcelona, Catalonia, Spain, 2000. ACM Press.

[Wurm 03] K. Wurm. ACT-R und SOAR - ein Vergleich. Technischer Bericht, Albert-Ludwigs-Universität Freiburg, 2003.

[Yu 04] L. Yu, H. Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224, 2004.

[Zöllner 05a] R. Zöllner, M. Pardowitz, S. Knoop, R. Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. *International Conference on Robotics and Automation (ICRA) 2005, Barcelona, Spain*, Seiten 1535 – 1540, 2005.

[Zöllner 05b] R. Zöllner. *Erlernen zweihändiger feinmotorischer Handhabungen*. Dissertation, Universität Karlsruhe (TH), GCA-Verlag, November 2005.

Die Anforderungen an Serviceroboter unterscheiden sich grundlegend von denen klassischer Industrieroboter. Dies umfasst sowohl die notwendige Flexibilität in menschenzentrierten, dynamischen Umgebungen, als auch die Fähigkeit, intuitiv mit Menschen interagieren zu können. Dazu sind völlig neue Mensch-Maschine-Schnittstellen notwendig, die auch Methoden zur Gesten- und Aktivitätserkennung beinhalten.
Vor diesem Hintergrund behandelt dieses Buch Möglichkeiten der Repräsentation, Akquisition und flexiblen Ausführung von Handlungswissen für Serviceroboter sowie Interaktionsmechanismen, die es einem Benutzer ermöglichen, intuitiv mit dem Roboter zu kommunizieren.

Zur abstrakten Darstellung von Roboterprogrammen werden Flexible Programme vorgestellt. Diese können aus Vorführungen eines Benutzers erzeugt werden, die mit Hilfe eines *Programmieren durch Vormachen*-Systems analysiert wurden. Um den Benutzer beobachten zu können, wurde ein neues Fusionsverfahren für verschiedene Sensormodalitäten entwickelt. Dadurch können Bewegungen, Gesten und Aktivitäten eines Menschen in Echtzeit verfolgt werden. Die vorgestellten Verfahren wurden auf realen Robotersystemen integriert und evaluiert.

ISBN: 978-3-86644-189-7

www.uvka.de