*Zur Entwicklung eines lern-
fähigen, behaviorbasierten Systems*

Jan-Torsten Milde

Zur Entwicklung eines lernfähigen, behaviorbasierten Systems

Dissertation zur Erlangung des Grades
Doktor der Philosophie (Dr. phil.)

Fakultät für Linguistik und Literaturwissenschaft
der Universität Bielefeld

vorgelegt von

Jan-Torsten Milde

Bielefeld - März 1997

Hiermit erkläre ich, daß von mir eingereichte Arbeit mit dem Titel

Zur Entwicklung eines lernfähigen, behaviorbasierten Systems

weder in der gegenwärtigen noch in einer anderen Fassung einer anderen Hochschule als Qualifikationsarbeit zur Erlangung des Doktorgrades oder eines anderen akademischen Grades vorliegt, oder Gegensatnd eines schon abgeschlossenen akademischen Qualifikationsverfahrens ist.
Ich versichere, daß diese Dissertation selbständig verfaßt wurde. Es wurden ausschließlich die angegebenen Quellen und Hilfsmittel verwendet. Wörtlich oder inhaltlich übernommene Stellen sind als solche gekennzeichnet.

Diese Dissertation ist auf alterungsbeständigem Papier nach DIN ISO 9706 gedruckt.

Bielefeld, den 20.3.1997 Jan-Torsten Milde

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Ziel der Arbeit

Ziel der Arbeit ist die Entwicklung eines lernfähigen, reaktiven Manipulators, der in einer dynamischen Umwelt agiert. Ausgehend von einer behaviorbasierten Steuerungskomponente, die die grundsätzlichen Fähigkeiten des Systems festlegt, soll das System durch Interaktion mit der Umwelt seine Handlungsmöglichkeiten erweitern. Im Rahmen der Arbeit müssen drei Teilziele erreicht werden:

1. der Agent muß realisiert werden (in Form eines Simulationssystems),
2. ein zugrundeliegendes reaktives Steuerungssystem muß geplant und implementiert werden und
3. das darauf aufsetzende Lernverfahren muß in der Lage sein, Handlungssequenzen zu identifizieren und unter Beibehaltung der Reaktivität in das Steuerungssystem zu integrieren.

Der Manipulator ist Teil einer Simulation, welche derart gestaltet werden soll, daß die experimentell gewonnenen Ergebnisse mit vertretbarem Aufwand auf reale System übertragbar sind. Der Manipulator soll seine Umwelt sensorisch erfassen können und auf die Perzepte adäquat reagieren.
Das Steuerungssystem soll nach den Prinzipien behaviorbasierter Systeme entwickelt werden. Das System soll sowohl von einer deliberativen Komponente aus steuerbar sein, als auch zur Laufzeit erweiterbar sein, um so die Eingliederung neuer Handlungsketten zu ermöglichen.
Das Lernverfahren soll ohne explizites Weltmodell auskommen und direkt in das reaktive Steuerungssystem eingebaut sein. Grundlage des Lernverfahrens sind Wahrnehmungen und Handlungen des Systems.

1.2 Motivation

Intelligentes Verhalten bei Menschen und Tieren basiert auf Handlungs-, Wahrnehmungs- und Lernfähigkeit. Sowohl Menschen als auch Tiere sind in der Lage, auf unvorhersehbare Ereignisse adäquat zu reagieren. Hierbei werden die bereits bekannten intelligenten "Verhaltensmuster" abgerufen und auf die entsprechende Situation abgewandelt übertragen. Situationen, die nicht mit den bekannten reaktiven Mustern verarbeitet werden können, führen zu einer erhöhten Belastung (s.a. [24]). Der Mensch ist in der Lage, die relevanten Aspekte einer Situation zu erkennen, sie zu klassifizieren und bei nochmaligem Auftreten einer "ähnlichen" Situation angepaßt zu reagieren. Entscheidend für die korrekte Verarbeitung und Speicherung der so gewonnenen Daten ist die Fähigkeit des Menschen, zu erkennen, daß ein Lernfortschritt stattgefunden hat. Der Mensch weiß, was er weiß bzw. nicht weiß. Diese Rekursivität des Lernens ist der Schlüssel zur kreativen Intelligenz. Sie versetzt den Menschen in die Lage, durch gedankliche Simulation von realen (und fiktiven) Prozessen Ergebnisse vorherzusagen.

Zu Beginn der 30er Jahren dieses Jahrhundert entstand die Ethologie als neuer Teilbereich der Verhaltensforschung. Prägend für diese neue Richtung der Verhaltensbiologie waren die Arbeiten der Biologen Konrad Lorenz [47], [46] und Nikolas Tinbergen [79]. Die Ethologie sieht ein Tier nicht als passiven, reizgesteuerten Organismus, sondern als aktiven, lebendigen Organismus, der in seiner ökologischen Nische überlebt. Dabei stützt sich die Ethologie stark auf die Beobachtung von Tieren innerhalb ihrer Umgebung. So untersucht etwa Tinbergen das Verhalten von Stichlingen, Konrad Lorenz das von Graugänsen und tropische Fischen. Der Ethologie liegen vier Basiskonzepte zugrunde.

1. Jede Art hat ein angeborenes, artspezifisches Verhalten.
2. Es ist bei den Individuen einer Art gleich.
3. Das angeborene Verhalten wird von einer Generation auf die nächste übertragen und dient der Anpassung an veränderte Lebensumstände.
4. Es kann sich durch evolutionäre Mechanismen ändern (Mutation, Crossingover).

Jedes Individuum einer Art unterliegt einer Entwicklung. Die phylogenetische Entwicklung ist bei jedem Individuum gleich und leitet die unterschiedlichen Entwicklungsphasen ein. Die ontogenetische Entwicklung eines Individuums wird durch das artspezifische Verhalten und durch äußeren Einfluß

gesteuert. Die Individualentwicklung folgt dabei einem bestimmten Muster. Deshalb ist die Evolutionstheorie wichtig für die Ethologie. Jedes Individuum ist (begrenzt) lernfähig. Dies zeigt sich in der artspezifischen Lerndisposition, welche in Phasen auftritt.
Innerhalb einer lernsensiblen Phase ist das Individuum bereit, neues Verhalten zu erwerben. Es zeigt dabei starke Reaktionen auf spezifische Reize. In einer lernsensiblen Phase sind diese Reize leicht in der Lage, das Verhalten zu modifizieren. So zeigt etwa Lorenz [47], daß die Prägung der Graugänse irreversibel ist. Dabei ist die Beeinflussung abhängig vom Entwicklungsstand des Individuums. Umstritten ist die Anzahl der Prägungsphasen; Lorenz geht von einer nicht weiter bestimmten "hohen" Anzahl aus ([47]).
Die Grundlage der ethologische Methodologie sind Beobachtungen in der natürlichen Umgebung und in Laborexperimenten. Bei Beobachtungen in der natürlichen Umgebung wird ein sogenanntes "Ethogramm" erstellt. Dieser Katalog der beobachteten Verhaltensweisen und der Spezifika der beobachteten Umgebung stellt die Grundlage für die Klassifikation von Verhaltensweisen entsprechend ihrer Funktion dar. Das Ethogramm wird als "objektive" Beschreibung verstanden, die einen ausreichend großen Ausschnitt der Lebenswelt des beobachteten Tieres erfaßt. Aufgrund dieser Annahme ermöglichen Ethogramme artübergreifende Vergleiche.
Laborexperimente werden eingesetzt, um gezielt Bedingungen zu modifizieren, um die aufgrund von Ethogrammen stattfindende Modellbildung zu verifizieren. So werden Deprivationsexperimente durchgeführt, welche Schlüsselreize in lernsensiblen Phasen verhindern. Damit kann entsprechend der Annahme, daß prägungsrelevante Informationen fehlen, überprüft werden, ob die sonst zu erwartenden Verhaltensweisen auftreten oder nicht.

Motiviert durch die Ergebnisse der Ethologie, überträgt Brooks [11] deren Prinzipien auf Steuerungssysteme für autonome Roboter (Subsumptionsarchitektur). Grundlage des Ansatzes ist die Hypothese, daß, im Gegensatz zur Grundnahme der klassischen KI ([83], [82],[70]), intelligentes Verhalten ausschließlich emergent ist, also ohne Weltrepräsentation und Bedeutungsrepräsentation auskommt. Brooks geht davon aus, daß die Hierarchie der Verhaltensweisen (Behaviors) nur ausreichend komplex sein muß, damit eine Kreatur als intelligent bezeichnet werden kann. Roboter, die nach dieser Architekturvorstellung realisiert worden sind, weisen die Fähigkeit auf, sich ähnlich wie Insekten durch eine ihnen unbekannte Welt zu bewegen. Die reaktiven Fähigkeiten des Systems liegen jedoch von Anfang an fest. Sie erweitern sich nicht im Laufe des "Lebens" der künstlichen Kreatur. Die Behaviors beschreiben also die Potenzmenge aller möglichen Reaktionen des Systems.

Die so definierte Architektur hat den Vorteil, daß das System zu jedem Zeitpunkt sinnvoll reagieren kann. Sich widersprechende und daher sinnlose Aktionen werden nicht ausgeführt. Ein Nachteil besteht in der Unfähigkeit des Steuerungssystems sich zur Laufzeit (Lebenszeit) an neue Umweltbedingungen zu adaptieren.
Der behaviorbasierte Ansatz stellt eine vernünftige Basis dar, um intelligente Systeme zu entwickeln. Als weitestgehend an situative Daten gebundene Systeme sind eine Reihe beobachtbarer Eigenschaften existierende intelligenter System mit dem behaviorbasierten Ansatz allein nicht zu erreichen. So sind behaviorbasierte Systeme[1] nicht in der Lage zur:

- Selbstmodifikation / Erweiterung (modifizierbares Standardbehavior)
- Gedächtnisleistung (Speicherung und Zugriff)
- Introspektion
- automatischen Klassifikation unbekannter Sensordaten

Ausgangspunkt dieser Arbeit ist ein behaviorbasiertes Steuerungssystem welches einen simulierten Manipulator vom Typ PUMA 260 kontrolliert. Das Steuerungssystem wurde so erweitert, daß basierend auf den vorhandenen Grundverhaltensweisen, neue Handlungsketten erlernt werden können und in das bestehende Kontrollsystem eingebaut werden.
Im Laufe der Arbeit wurden drei Experimentalsysteme entwickelt:

1. das Simulationssystem Cyrano
2. der mobile Roboter Samsa
3. die Simulation eines Manipulators PUMA 260

Der Einsatz von Simulationen erlaubt die Durchführung von reproduzierbaren Testläufen. Durch die weitgehend frei konfigurierbaren Simulationssysteme können in kurzer Zeit eine Vielzahl unterschiedlicher Situationen überprüft werden. Dabei müssen bei der Konstruktion der Simulationssysteme eine Reihe von Bedingungen eingehalten werden. Nur so sind die Ergebnisse, die innerhalb der Simulation erarbeitet werden, übertragbar auf reale Systeme.

[1] genauer gesagt: es existiert derzeit überhaupt kein künstliches System, welches alle genannten Eigenschaften erfüllt

Die Arbeit besteht aus 3 Teilen. Teil 1 gibt einen Überblick des aktuellen Forschungsstandes in den Bereichen Simulation, maschinelles Lernen und behaviororientierte Systeme. In Teil 2 wird die Entwicklung des lernfähigen, behaviorbasierten Systems dargestellt. In Teil 3 wird der Ansatz zusammenfassend bewertet.

Kapitel 2

Forschungsüberblick

2.1 Simulation, Steuerung und Lernverfahren

In dem folgenden Kapitel wird ein Überblick über Techniken, Methoden und Ansätze gegeben, die in die Entwicklung des lernfähigen, teilautonomen Agenten eingeflossen sind. Ausgehend von der Überlegung, daß der zu realisierende Agent von einem behaviorbasierten Steuerungssystem kontrolliert werden soll, auf dem anschließend ein Lernverfahren aufsetzt, welches neue Verhaltensweisen erwirbt, werden die Themen *Simulation*, *Steuerung* und *Lernverfahren* in drei Abschnitten beschrieben.
Im ersten Teil werden Simulationssysteme und ihr Einsatz bei der Entwicklung von lernfähigen oder teilautonomen Robotersystemen dargestellt. Simulationen werden in der Robotik häufig verwendet, da die realen Systeme sowohl in der Anschaffung teuer als auch fehleranfällig und damit wartungsintensiv sind. Dies führt zu einer erheblich verlängerten Vorbereitungszeit für die eigentlichen Experimente. Der zweite Teil geht auf die drei wesentlichen Ansätze zur reaktiven Steuerung autonomer Systeme ein. Der dritte Teil stellt Verfahren des maschinellen Lernens dar, welche sowohl in realen Systemen wie auch innerhalb von Simulationen zum Einsatz kommen.
Für die vorliegende Arbeit gilt, daß für jeden der drei Teilbereiche eine Reihe von Bedingungen beachtet werden müssen:

Simulation: Der Agent muß in seiner physikalischen Ausprägung realisiert werden. Die Definition der Morphologie des Systems, die Auslegung der Sensorik und Aktuatorik und die Festlegung des Szenarios bestimmen, zu welchen Fähigkeiten der Agent prinzipiell in der Lage ist. Ist der Agent zudem Teil einer Computersimulation, gilt es, das Verhältnis von dieser zum realen Pendant zu beachten.

Steuerung: Das zu entwickelnde reaktive Steuerungssystem muß so ausgelegt sein, daß eine Anzahl von zugrundeliegenden Basisverhaltensweisen zur Verfügung steht. Dies sollte unter Verwendung der durch die Simulation gelieferten Sensordaten geschehen. Aufbauend auf den Basisbehaviors soll das Lernverfahren abstrakte Behaviors konstruieren und diese in die Steuerungshierarchie einbetten. Entsprechend gilt, daß das Steuerungssystem zur Laufzeit dynamisch modifizierbar sein muß, um diese Integration zuzulassen.

Lernverfahren: Für das Lernverfahren gilt, daß die Integration in das Steuerungssystem so erfolgen muß, daß keine explizite Trennung zwischen Handlungs- und Lernphase stattfindet. Lernen soll aufgrund von Handlung und Wahrnehmung stattfinden. Eine externe Bewertungsfunktion, welche implizit Wissen über Zielzustände kodiert, darf dabei nicht Teil des Systems sein. Ebenso soll eine Repräsentation des Zustandsraumes vermieden werden.

Neben der Darstellung externer Forschungsarbeit wird in diesem Kapitel auf zwei Eigenentwicklungen (das Simulationssystem Cyrano und das mobile Robotersystem SAMSA) eingegangen.

2.2 Simulationen teilautonomer Agenten

Zur Untersuchung von lernfähigen, teilautonomen Agenten kommen in vielen Fällen Simulationen zum Einsatz. Man kann unterscheiden zwischen

1. einfachen diskreten (2 dimensionale "Schachbrett" Systeme, z.B. Cyrano ([60])),

2. kontinuierliche, aber reduziert in der Komplexität der simulierten Sensorik (z.B. DiSCUS, 3d7 ([57],[81]) und

3. kontinuierlichen 3D (z.B. ROTEX, INSIGHT, SimRobot ([13], [73], [69]))

Ansätzen. Auf den folgenden Seiten sollen die Prinzipien dieser Simulationssysteme anhand einiger ausgewählter Beispiele dargestellt werden. Von besonderer Relevanz sind dabei die Arbeiten von Wengerek ([88]) und Strippgen ([73]). Wengerek beschreibt ein Lernverfahren zum Erwerb von auf Basisreflexen basierenden Handlungsfolgen. Dieses wird im wesentlichen in einer Simulation evaluiert. Die anschließende Fallstudie mit einem realen Roboter zeigt die bedingte Übertragbarkeit der Ergebnisse. Die Arbeit von Strippgen befaßt sich mit der Entwicklung eines speziell auf reaktive Steuerungsansätze ausgelegten Simulationssystems für behaviorbasierte Agenten.

2.2.1 Vorüberlegungen zur Simulation

Bei der Entwicklung einer Simulation muß von vorneherein klar sein, welche Annahmen über die modellierte Domäne vorausgesetzt werden. Nur so sind die Möglichkeiten und Beschränkungen der Simulation klar abschätzbar. Aussagen über Generalisierbarkeit, Skalierbarkeit und Aussagekraft der Simulationsergebnisse können nur in Abhängigkeit der Relation zwischen Simulation und der "echten Welt" getroffen werden. Wichtige Charakteristika der Simulation sind *Modularität* und *Erweiterbarkeit*. Eine "ideale" Simulation erlaubt die weitgehend freie Konstruktion eines Roboter durch Verwendung von Sensoren und Effektoren, welche als Softwareobjekte frei konfigurierbar sind (siehe dazu [30]). Fehlermodelle sollten ebenfalls frei konfigurierbar sein. Aus der Sicht des Softwareengineers ist es sinnvoll, die Einzelkomponenten in einer deklarativen-prozeduralen Programmiersprache zu entwickeln. Hierdurch können unterschiedliche Implementationen auf unterschiedlichen Plattformen unterstützt werden. Die Simulation sollte Modelle allgemein verwendeter Sensoren (Sonar, Pyroelektrische Sensoren, Infrarotsensoren,

Mikrophone, Photozellen, etc.) zur Verfügung stellen. Sowohl Sensormodelle als auch Effektormodelle sollten einfach zu erweitern bzw. zu verändern sein. Das Simulationsszenario muß konfigurierbar sein bzgl. der Anordnung der Objekte und ihrer Eigenschaften. Diese müssen von den Modellen der Sensoren und Effektoren einbezogen werden.
Neben der Konfiguration des Roboters und des Simulationsszenarios muß die Simulation als solche möglichst variabel sein. Präzision, Geschwindigkeit, Granularität von Raum und Zeit, kontinuierliche Simulation und diskrete Simulation müssen möglich sein. Wiederholbare pseudozufällige Simulationsläufe erlauben das Austesten der deterministischen Steuerung.
Weiterhin sollte die Simulation als Simulationstoolkit mit einer benutzerfreundlichen Oberfläche (möglichst GUI) ausgestattet sein. Wünschenswert ist ein verteiltes System. Die dadurch mögliche Lastverteilung auf unterschiedlichste Hardware erlaubt auch bei Nichtvorhandensein extrem leistungsfähiger Hardware realistische Simulationsläufe. Ein verteiltes System würde weiterhin die Modularisierung begünstigen und so beispielsweise Steuerung und Simulation entkoppeln.

2.3 Simulationen in der Robotik

Zweidimensionale Simulationen stellen den größten Anteil an derzeit eingesetzten Experimentalsystemen im Bereich der Untersuchung verhaltensbasierter Systeme dar und zwar für: Lernexperimente mit einzelnen Agenten ([5],[6],[74], [35]), Evaluierung von Multiagentensystemen [61] und Simulation von kompletten Ökosystemen ([84]).
Die Welt wird dabei durch ein zweidimensionales Feld repräsentiert. Jede Feldzelle entspricht einer möglichen Position innerhalb der Welt. Je nach Komplexität der Simulation können Positionen unterschiedlich besetzt werden, z.B. durch Agenten, Futter, feindliche Agenten, Hindernisse. Die verhaltensbasierten Agenten sind in der Lage, ihre Position zu ändern, in der Regel durch Bewegung in Einzelschritten auf einem der acht umliegenden Felder der aktuellen Position. Die Bewegung hängt dabei von Umweltparametern, Motivation und internen Zuständen (z.B. Energiebedarf) ab. Die in Abschnitt (2.5.1) beschriebene 2D-Simulation Cyrano entspricht in ihrer Funktionalität den gerade beschriebenen Parametern.
Die Agenten sind in Abhängigkeit vom umgebenden Szenario mit einer Reihe spezifischer Sensoren ausgestattet. Wahrnehmung heißt bei dieser Form der Simulation das Auslesen von speziellen Zelleninhalten der Weltrepräsentation. Der Wahrnehmungshorizont ist begrenzt (auch hier in der Regel auf die umliegenden Felder) und die Interpretation des Datums ist ungestört und ein-

deutig.
Neben der diskreten Repräsentation des Raumes ist bei Simulationen dieses Typs auch die Zeit diskretisiert. In vorgegebenen Taktzyklen können Agenten jeweils eine ausgezeichnete Anzahl von Aktionen durchführen. Dies führt insbesondere bei Multiagentensystemen zu Abhängigkeiten vom verwendeten Schedulingmechanismus.

2.3.1 Robotersimulationen

Kuehnapfel beschreibt in [37] detailliert Ansätze zu Simulation von Robotersystemen. Diese sind jedoch stark auf Nutzung in Kombination mit einer konventionellen Robotersteuerung ausgelegt. Insbesondere im Bereich der Sensorik bieten sie nur wenig Unterstützung.

ROTEX ([13]) : Rotex simuliert einen Roboterarm, welcher mit mehreren Sensoren ausgestattet ist. Der Arm wird im Rahmen der D2-Mission im Weltraum erprobt. Es wird eine Teilautonomie des Roboters angenommen, der aufgrund seiner Sensoren dazu fähig ist, mit Unsicherheiten in seiner Umgebung umzugehen, so daß der Mensch auf einer aufgabenbezogenen Ebene mit dem Roboter kommunizieren kann (*Telesensorprogrammierung*). Innerhalb des Systems werden verschiedene Sensoren (Kamera, Entfernungs- und Kraft-Drehmoment-Sensoren) simuliert. Das reale System und die Simulation sind miteinander koppelbar. So kann zunächst offline programmiert werden und dann die entsprechenden Programme auf den realen Roboter übertragen werden. Alternativ kann der aktuelle Zustand des realen Roboters zur Korrektur der Simulation verwendet werden.

DiSCUS ([57]) : DiSCUS ist eine verteilte Simulation eines Puma 260. Der Arm kann offline programmiert werden (in VAL II), was die Übertragung eines Programms, welches zunächst in der Simulation erprobt wird, auf den realen Roboter ermöglicht. Die Simulation stellt ausschließlich arminterne Sensordaten zur Verfügung (Gelenkstellung, aktuelle Winkelgeschwindigkeit, Greiferstellung etc.).

SimRobot ([69]) : Zielsetzung des Projektes SimRobot ist die Entwicklung eines universellen Baukastens zur Simulation unterschiedlicher Robotersysteme (mobile (radbasierte) Roboter, Manipulatoren und Laufmaschinen). Die Spezifikation erfolgt über eine hierfür konzipierte Beschreibungssprache. Neben einfachen Sensoren wie Tasthaaren und Lichtstärkemessern können virtuelle Kameras eingesetzt werden. Diese sind durch Angabe von Auflösung und Wahrnehmungsbereich (als

Grundfläche der Sichtpyramide) beschreibbar. Das System ist objektorientiert realisiert, wodurch die Modularisierung der Sensoren mittels einfacher Vererbung möglich wird. Die Visualisierung erfolgt mittels eines einfachen Raytracing-Modells, welche unter anderem die Simulation von Lichteffekten zuläßt.

3d7 ([81]) : Es wird ein mobiler Roboter mit einem Laserradar und einem Ultraschallsensor simuliert, wobei die Umgebung dreidimensional modelliert wird.

2.3.2 Die Simulation von Wengerek

Wengerek untersucht in [88], inwieweit ein simulierter Roboterarm mit einem Zweizangengreifer die geeignete Greifposition für eine bestimmte Klasse von Objekten finden kann (s.a [90], [89]). Die zu greifenden Objekte sind aus einem oder mehreren Quadern zusammengesetzt. Ausgehend von einer beliebigen Startposition soll der Roboter lernen, wie eine günstige Greifposition eingenommen wird. Anschließend soll der Roboter in der Lage sein, das Objekt anzuheben (siehe Abb. 2.1).
Der simulierte Roboter verfügt über eine eingeschränkte simulierte Sensorik. Am Endeffektor stehen sechs einfache Entfernungssensoren zur Verfügung. Diese sind strahlbasiert modelliert, liefern also die Koordinaten des möglichen Schnittpunktes eines imaginären Strahls mit der Außenfläche (bzw. mit deren Projektion in die Ebene) der quaderbasierten Objekte. Hierbei gilt als weitere Einschränkung, daß die Strahlen jeweils senkrecht zur Arbeitsfläche stehen und zwar unabhängig von der Ausrichtung der Hand. Dies hat zur Folge, daß bei der Rezeption von Objekten vier "Regionentypen" unterschieden werden (*Kantenregion* und *Mittelregion*, sowie ihre Komplemente). Insgesamt ergibt sich also ein (relativ kleiner) sensorischer Raum mit $4^6 = 4096$ möglichen Zuständen.
Wengerek räumt ein, daß die simulierte Sensorik nicht mit realen, in der Robotik verwendeten Sensoren übereinstimmt. Er ist jedoch der Meinung, daß eine Vielzahl von Eigenschaften existierender Systeme nachgebildet werden können, so z.B. Störungen (siehe hierzu [88] Seite 92ff).
Mögliche Aktionen des Roboterarms beziehen sich auf das "Tool-Koordinatensystem" innerhalb des Endeffektorgreifers. Die Basisbewegungsmuster umfassen Translationen in der XY-Ebene und Rotationen um die Z-Achse des Tool-Koordinatensystems, wobei auch hier von einer zur Oberfläche des zu greifenden Objektes parallelen Lage der Bewegungsebene ausgegangen wird. Damit ergibt sich ein Zustandsraum mit einer maximalen Anzahl von Zustand-Aktionspaaren von $6 * 4^6 = 24576$, innerhalb dessen Greifstrategien

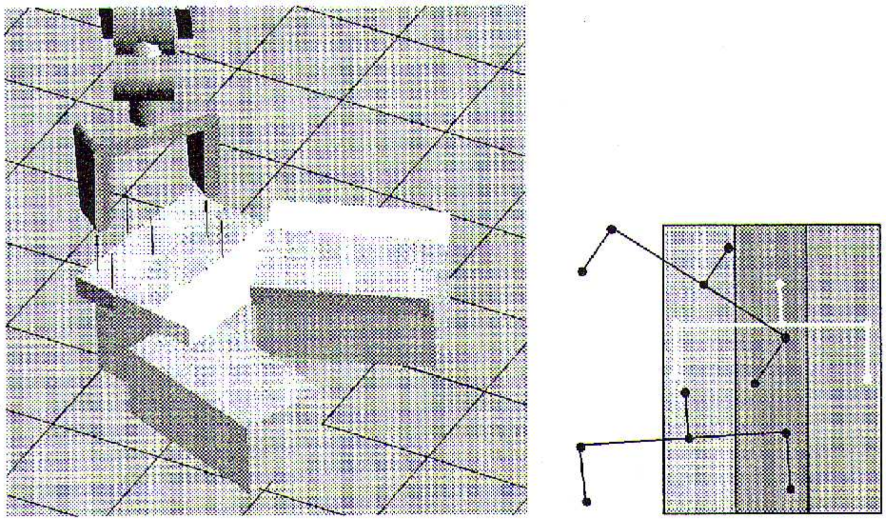

Abbildung 2.1: Momentaufnahme der von Wengerek eingesetzten Simulation. Die linke Abbildung zeigt den simulierten Manipulator bei der sensorgestützten Annäherung an ein Objekt. Die vertikalen Striche deuten die Position und Ausrichtung der virtuellen Sensoren an. Die rechte Abbildung zeigt eine Aufsicht auf eine entsprechende Szene. Hier sind mögliche Sensorkonfigurationen markiert. Die schwarzen Konfigurationen ergeben, trotz unterschiedlicher Ausrichtung des Endeffektors, das gleiche Sensordatum. Die weiß markierte Konfiguration zeigt eine Sensorkonfiguration, welche eine gültige Greifposition definiert.

erlernt werden sollen.

2.3.3 Der experimentelle Ablauf

Ausgehend von einer zufälligen Position soll sich der Roboterarm den Objekten nähern und eine das Zugreifen ermöglichende relative Lage zu den Objekten einnehmen.
Da der Roboter lediglich über Kontaktsensoren verfügt, wird die Grobansteuerung über ein Modul realisiert, welche das Tool-Koordinatensystem in Richtung des Masseschwerpunktes des zu greifenden Objektes bewegt. Sobald einer der Kontaktsensoren aktiv wird, übernehmen die Basisreflexe und der Lernalgorithmus die Kontrolle.
Bei jedem Versuch wird die Startposition zufällig ausgewählt. Nach einer festen Anzahl von Schritten ist das Experiment beendet, es sei denn, es wird im Laufe des Experiments eine Greifposition eingenommen (erfolgreiche Beendigung), oder der Roboter verliert den sensorischen Kontakt zum Objekt (erfolglose Beendigung). Weiterhin ist eine Abbruchbedingung eingeführt worden, die sicherstellt, daß die Greifposition aufgrund der durch das Lernverfahren gewählten Strategie eingenommen wurde und nicht das Ergebnis eines "Zufallstreffers" ist. Erst wenn, ausgehend von der gleichen Startposition, eine definierte Anzahl von unmittelbar nacheinander durchgeführten Versuchen zum Erfolg führt, gilt die Strategie als erfolgreich.

2.4 Virtuelle Umgebungen für situierte Agenten

Virtuelle Umgebungen für situierte Agenten werden verstärkt in der sich entwickelnden Forschungsrichtung *Artificial Life* eingesetzt. Im Gegensatz zu einfachen zweidimensionalen Simulationen zeichnen sich virtuelle Umgebungen durch ihre größere Realitätsnähe aus. Der dazu notwendige hohe Entwicklungsaufwand sorgt dafür, daß qualitativ hochwertige Simulationen eher selten sind.

AnimNL: Ziel des AnimNL Projektes ([3]) ist es, mit Hilfe natürlichsprachlicher Instruktionen einen computergrafisch simulierten Agenten zu steuern. Aus einer Folge von Instruktionen und den ihnen entsprechenden Aktionen wird dann eine fortlaufende Animation des Agenten generiert. Neben der grafischen Animation werden zusätzlich natürlichsprachliche Kommentare generiert, welche Teilaktionen des Agenten erklären.

Das System besteht aus vier Hauptkomponenten, der NLP Einheit, einem inkrementellen Planer, der Simulation und der Visualisierungskomponente. Die Komponenten sind als Pipeline hintereinandergeschaltet.

Amazon: Amazon [14] ist eine Simulation einer zweidimensionalen Spielwelt, in welcher der Agent Sonja agiert. Simuliert wird ein Labyrinth aus Räumen, welche über Korridore und Treppen miteinander verbunden sind. Innerhalb der Räume treten weitere Spielfiguren auf. Weiterhin sind Werkzeuge über den gesamten simulierten Raum verteilt. Diese kann der Agent aufnehmen und in der entsprechenden Situation verwenden. Die Simulation erlaubt die Erzeugung "unvorhersehbarer" Situationen, auf welche der Agent dynamisch reagieren muß.

Der Agent Sonja verfügt über domänenspezifische Sensorik und Aktuatorik und ist natürlichsprachlich steuerbar. Damit ist er in der Lage, die Instruktion situiert zu verarbeiten, und zu handeln.

2.4.1 INSIGHT - Eine Simulationsumgebung für mobile Roboter

Strippgen beschreibt in [73] die Entwicklung einer interaktiven, grafischen Entwicklungsumgebung zur Konstruktion und Evaluation behaviorbasierter mobiler Agenten. Ausgehend von der Beobachtung, daß Experimente mit realen Systemen fehleranfällig und nicht reproduzierbar sind, stand bei INSIGHT die Forderung nach einer möglichst realistischen Simulation einerseits und die Bereitstellung eines einfach zu bedienenden Werkzeugs für die Entwicklung behaviorbasierter Steuerungsprogramme andererseits im Vordergrund.
Die Simulationskomponente realisiert ein Golfplatz-ähnliches Szenario. Um die Natürlichkeit des Geländes zu erhöhen, wird eine Hügellandschaft generiert, in welcher verschiedene Untergrundformen (Gras, Wasser, Sand) und Objekte (Ladestationen, "Golfspieler", Bäume, Fahnen, Bälle) auftreten. Die Untergrundeigenschaften haben Einfluß auf Energieverbrauch, Geschwindigkeit und Fahrverhalten der simulierten Roboter.
Die symbolische Weltrepräsentation der Simulation basiert auf einem Gitter fester Größe, wobei die Schnittpunkte dreidimensionale Koordinaten tragen, welche jeweils lokale Extrema der Landschaft beschreiben. Durch einen interpolierenden Algorithmus wird zu jeder Koordinate, welche nicht auf einem Gitterschnittpunkt liegt, der Höhenwert und die lokale Steigung berechnet. Damit wird es möglich, trotz einer diskreten Repräsentation eine kontinuierlich verlaufende Oberfläche zu erzeugen. Um die Landschaft realistischer

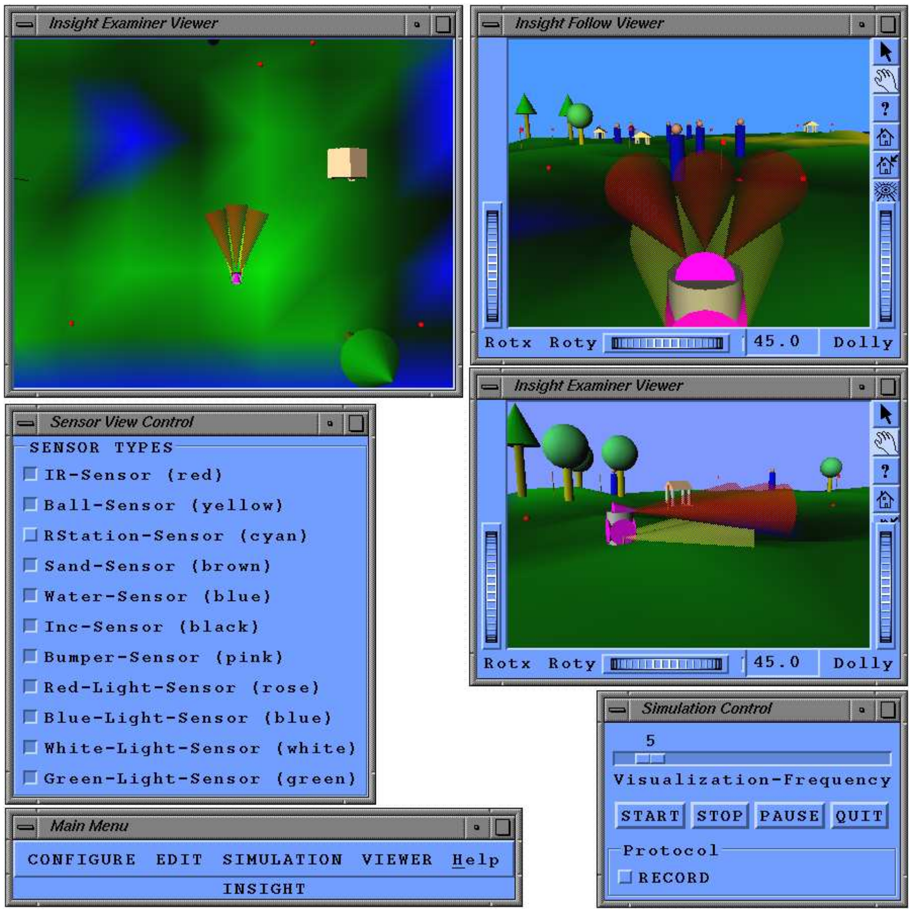

Abbildung 2.2: Das virtuelle Labor INSIGHT: eine Entwicklungsumgebung für behaviorbasierte mobile Agenten.
Es werden mehrere Sichten auf das System generiert (z.b. Aufsicht, Sicht aus der Perspektive des Agenten, Beobachtung von der Seite). Diese können unabhängig voneinander verändert werden. Über einfach zu bedienende Auswahlleisten können Sensortypen festgelegt werden. Die Simulationsgeschwindigkeit ist modifizierbar und Simulationsläufe können aufgezeichnet und per Playback zu einem späteren Zeitpunkt wiederholt werden.

zu gestalten, wird auf die so generierte Fläche ein Muster mit kontinuierlichen Regionen unterschiedlicher Oberflächeneigenschaften projiziert. Der Projektionsformalismus ist dabei so ausgelegt, daß keine topologisch invaliden Strukturen auftreten können (z.B. Wasser auf der Kuppe eines Hügels). Die mobilen Roboter, welche innerhalb des Szenarios agieren können, sind mit zwei unabhängig steuerbaren Antriebsmotoren ausgestattet. Hierdurch wird es möglich, die Bewegungsrichtung, den Kurvenradius und die Geschwindigkeit zu variieren.
Die sensorische Ausstattung der Agenten orientiert sich an der in realen Systemen eingesetzten. An der Basis des Roboters sind ringförmig Kontaktsensoren installiert. Über einen weiteren Sensor wird die zurückgelegte Strecke erfaßt. Ein Untergrundsensor liefert Informationen über die Qualität des aktuellen Untergrunds. Eine Reihe von kontaktfreien Entfernungssensoren mit kegelförmiger Charakteristik und beschränkter Reichweite werden genutzt, um frühzeitig auf Hindernisse reagieren zu können.

2.4.2 Der experimentelle Ablauf

Als dezidierte Experimentalumgebung unterstützt INSIGHT den Entwickler gut bei der Erstellung von behaviorbasierten Steuerungssystemen. INSIGHT läßt die Konfiguration von Agenten und Umgebung zu. Agenten können in ihrer sensorischen Ausstattung und der Initialpositionierung innerhalb der Umgebung modifiziert werden. Um Steuerungsprogramme in unterschiedlichen Environments testen zu können, werden diese vollautomatisch generiert. Für die Entwicklung behaviorbasierter Steuerungsprogramme wird in der vorliegenden Version ausschließlich der dynamische Ansatz von Steels unterstützt. Ein Interpreter für die von Steels entwickelte Programmiersprache PDL ([72]) ist in das System integriert.
Quantities werden in PDL zur Koppelung der Behaviormodule, dem Einlesen von Sensorwerten und der Ausgabe von Aktuatorwerten verwendet. Ein wesentlicher Teil der Dynamik des Systems ist anhand der Werteschwankungen der Quantities identifizierbar. Aus diesem Grund ermöglicht INSIGHT den externen Zugriff auf die Quantities. Um dem Entwickler ein Gefühl für die Dynamik des zugrundeliegenden Steuerungssystems zu geben, lassen sich die Werte der Quantities in einem zweidimensionalen Zeit/Wertdiagramm anzeigen.
Die jeweils aktuelle Experimentalkonstellation läßt sich abspeichern und wieder laden, was eine schnelle Wiederholung von Experimentläufen unter unterschiedlichen Parametersettings unterstützt.

2.5 Eigene Vorarbeiten

Auf den folgenden Seiten werden zwei Systeme erläutert, welche im Verlauf dieser Arbeit entwickelt worden sind. Es handelt sich um die Simulation Cyrano und den mobilen Roboter SAMSA.
Hauptziel bei der Entwicklung der Systeme war es, festzustellen, inwieweit ein hybrider Ansatz, bestehend aus einer deliberativen Komponente und einer reaktiven Komponente, tragfähig ist, um zielgerichtetes und reaktives Verhalten sinnvoll miteinander zu kombinieren.
Weiterhin lagen zum Zeitpunkt der Entwicklung keinerlei Erfahrungen mit der Spezifikation und Realisierung behaviorbasierter Steuerungssysteme vor. Die Entwicklung der Prototypen lieferte die Grundlage für eine vernünftige Abschätzung, welcher der in Kapitel 2.8 beschriebenen behaviorbasierten Ansätze (bzw. ein entsprechend modifizierter Ansatz) verwendet werden kann.
Besonders die Ergebnisse, welche hinsichtlich Systemarchitektur, Repräsentationen und Steuerungssystem in der Simulation Cyrano entwickelt wurden, waren ausschlaggebend für die weitere Arbeit. Ein vergleichbarer, wenn auch komplexerer Ansatz wird im Projekt D1 "Kommunizierende Agenten" verfolgt ([28]).

2.5.1 Cyrano

Zielsetzung bei der Entwicklung von Cyrano war die Untersuchung, inwieweit eine hybride Steuerungsarchitektur, bestehend aus einer deliberativen und einer reaktiven Komponente, genutzt werden kann, um eine integrierte Analyse handlungssteuernder Anweisungen zu ermöglichen. Die Analyse derartiger Anweisungen stellt den Computerlinguisten vor eine Vielzahl von Problemen. Man stelle sich folgendes Szenario vor: Zwei Personen, ein Instruktor und ein Agent, sollen aus 3 verschiedenfarbigen Blöcken einen Turm bauen. Der Instruktor gibt ausschließlich Anweisungen und plant die Aktion. Der Agent verfügt über einen Basissatz an möglichen Aktionen (z.B. er weiß, wie ein Block zu greifen ist und versteht die Anweisungen des Instruktors). Ein Beispieldialog zwischen der anweisenden und der durchführenden Person könnte so aussehen:

1. Instruktor: Verwende den roten Block als Basis des Turms.
2. Agent: OK.
3. Instruktor: Hebe den anderen Block auf den roten Block.
4. Agent: Welchen der Blöcke ? Ich sehe einen grünen und einen gelben Block.

5. Instruktor: Nimm den gelben.
6. Agent: OK.
7. Instruktor: Und jetzt das gleiche mit dem letzten Block.
8. Agent: Ich kann den letzten Block nicht auf den roten Block legen.
9. Instruktor: Lege den letzten Block auf den oberen Block.
10. Agent: OK.

Schon an diesem Dialog lassen sich eine Anzahl linguistischer Probleme ausmachen:

- Referenzprobleme
- Ellipsen

Der Agent kann seine Aufgabe nur deshalb lösen, weil er über situatives Wissen verfügt. Er nimmt die aktuelle Weltsituation wahr und reagiert dynamisch auf diese. Sprachliche Anweisungen und Situationswissen müssen also gleichzeitig verarbeitet werden.
Aktuelle sprachverarbeitende Systeme arbeiten stark stratifiziert. Eine Rückkoppelung zwischen den Einzelkomponenten findet nicht oder nur sehr eingeschränkt statt. Hierdurch sind diese Systeme in Bereichen mit einer sich dynamisch verändernder Umwelt kaum einsetzbar. Ähnliches gilt für Entwicklungen im Bereich der Robotik. Realisierte Roboter versuchen, durch Analyse der Sensordaten eine interne Repräsentation der Außenwelt zu generieren. Diese wird dann von nachgeschalteten Komponenten verarbeitet. In der Regel ist der Aufbau einer Repräsentation ein zeitraubender Prozeß, so daß die realisierte Repräsentation nicht mehr der aktuellen Weltsituation entspricht. Durch diese Inkonsistenz kann der Roboter nicht adäquat auf die Situation reagieren.
Bereits 1981 stellt Norman [62] eine Systemarchitektur vor, die den stratifizierten Ansatz aufgibt. Ein intelligentes System soll aus einer kognitiven Komponente und einer reaktiven Komponente bestehen (Abb. 2.3).
Die kognitive Komponente arbeitet auf internen Weltrepräsentationen. Sie ist zuständig für die Planung von Aktionen. Die reaktive Komponente analysiert sensorische Daten und steuert die Aktuatoren. Beide Teilkomponenten interagieren miteinander.
Der *behaviorbasierte* Ansatz von Brook ([11]) verzichtet vollständig auf eine kognitive Komponente, die auf internen Repräsentationen arbeitet. Brooks

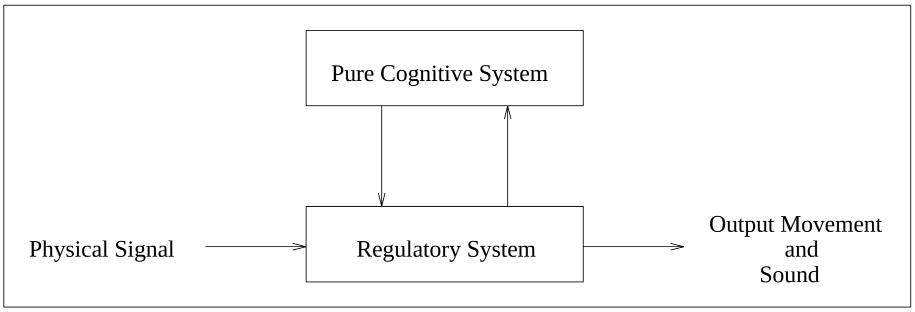

Abbildung 2.3: Systemarchitektur nach Norman. Während das *Pure Cognitive System* von der Welt entkoppelt ist und lediglich auf einer Repräsentation dieser arbeitet, wird im *Regulatory System* eine nicht-kognitive unmittelbare Verknüpfung von Signal und Effekt realisiert. Das Weltmodell des *Pure Cognitive System* kann durch Informationen aus dem *Regulatory System* angepaßt werden, kognitiv motivierte Entscheidungen fließen zurück in das *Regulatory System* und beeinflussen so mögliche Effekte in der Welt.

geht davon aus, daß intelligentes Verhalten nicht durch die Verarbeitung einer symbolischen Repräsentation der Welt entsteht, sondern er fordert, daß "die Welt ihre eigene Repräsentation" sei. Der Agent soll zu jedem Zeitpunkt die Welt wahrnehmen und interpretieren. Anhand der so gewonnenen Informationen kann er dann situationsangepaßt reagieren. Realisiert wird dieser Ansatz durch *behaviors*. Ein Behavior ist ein Schaltkreis, der unmittelbaren Zugriff auf Sensoren und Aktuatoren hat. Durch diese direkte Koppelung von Input und Output kann das Behavior schnell reagieren. Es wird keine interne Repräsentation aufgebaut. Die Behaviors sind hierarchisch geordnet und gekoppelt. Diesen Architekturansatz nennt Brooks *Subsumptionarchitektur*.

Der behaviorbasierte Ansatz wurde von David Chapman in dem System Amazon zur Steuerung des Agenten Sonja verwendet [14]. Der Agent Sonja kann sich selbständig durch die dynamische Umwelt bewegen und mit ihr interagieren. Neben dem sensorischen Input läßt Chapman sprachliche Anweisungen zu. Diese werden jedoch nicht im klassischen computerlinguistischen Sinne analysiert, sondern lediglich mit einem gespeicherten Datenbestand verglichen (Pattern-Matching) und an die Behaviors übergeben. Dies hat den Effekt, daß Sensordaten und Sprache gekoppelt sind. Unsinnige Anweisungen werden als nicht-situationsrelevant erkannt und ignoriert.

Systemarchitektur

Ausgehend von den geschilderten Überlegungen wurde ein behaviorbasiertes Architekturkonzept entwickelt, welches im wesentlichen aus folgenden Komponenten besteht (siehe auch [44]):

- natürlichsprachliche Komponenten: Parser und Generator
- aktionssteuernde Komponente: Behaviormodul
- visuelle grafische Ausgabe: Grafikmodul

Zusätzlich muß eine Umgebung entwickelt werden, die die Simulation einer künstlichen Welt realisiert. Innerhalb dieser kann sich der Agent bewegen und sie mittels seiner Sensoren wahrnehmen. Die Simulation dieser Welt ist bewußt einfach gehalten. Ziel des Projektes Cyrano ist nicht die Entwicklung komplexer Sensorauswertungsmechanismen, wie etwa Algorithmen der Bilderkennung, sondern es soll gezeigt werden, daß ein sinnvolles Zusammenspiel von sprachlicher Anweisung und Sensordatenaufnahme intelligente Sprachverarbeitung überhaupt erst ermöglicht.
Abbildung 2.4 zeigt die Systemarchitektur in der Übersicht. Aufgabe des Parsers ist es, die syntaktische Wohlgeformtheit der handlungssteuernden Anweisung zu überprüfen. Außerdem wird die Anweisung in eine geeignete (einfache) semantische Repräsentation überführt. Diese ist, zusammen mit den aktuellen sensorischen Daten, der Input des Behaviormoduls. Der Input muß nicht vollständig spezifiziert sein. Man geht vielmehr davon aus, daß die aktuellen Sensordaten zur Vervollständigung der funktionsrelevanten Informationen des angesprochenen Behaviors genutzt werden können. Sollte dies nicht der Fall sein, muß das angesprochene Behavior die entsprechenden Informationen vom Benutzer des Systems abfragen. Dementsprechend wurde ein (einfacher) Generator implementiert, welcher die Generierung der natürlichsprachlichen Rückfragen realisiert. Der Benutzer tritt also in einen Dialog mit dem künstlichen Agenten. Dieser ist in seinem Diskurs stark eingeschränkt. Es ist in der Regel lediglich eine Rückfrage nötig, möglich sind aber beliebig viele. Verfügt das Steuerungssystem über ausreichende Informationen, um eine Aktion zielgerichtet durchzuführen, bedeutet dies, daß genau ein Behavior aktiv werden kann. Da es sich bei dem System um ein dynamisches handelt, können Situationen auftreten, für die das aktuell aktivierte Behavior nicht zuständig ist. In diesem Fall wird in der nächst tieferen Ebene der Behaviorhierarchie geprüft, ob ein zuständiges Behavior aktiviert werden kann. Ist dies der Fall, wird dieses Behavior angewendet, und der Prozeß kehrt zur Ursprungsebene zurück. Ist dies nicht der Fall, so wird

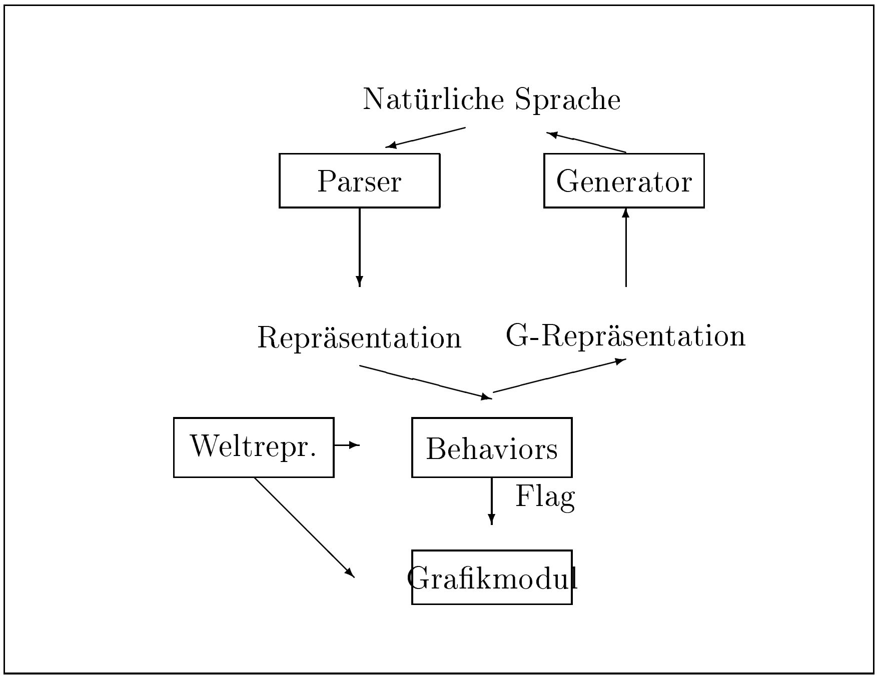

Abbildung 2.4: Übersicht Systemarchitektur
Die hybride Architektur von Cyrano besteht aus drei Subsystemen: dem höheren System (Parser/Generator und Zwischenrepräsentationen), dem Behaviorsystem und der Simulation. Parser/Generator bilden natürlichsprachliche Instruktionen auf Attribut/Wertstrukturen ab, welche die behaviorbasierte Steuerungskomponente in Kommandos an den Agenten umsetzt. Das Grafikmodul bildet in Kombination mit der Weltrepräsentation die Simulation von Cyrano.

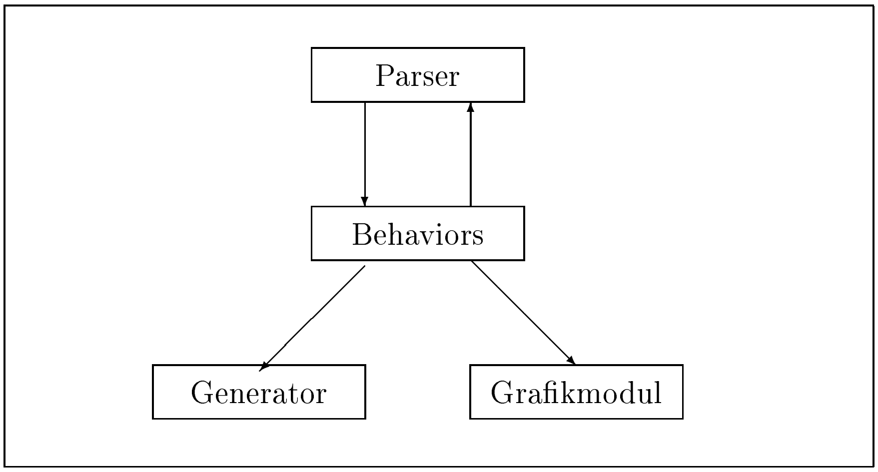

Abbildung 2.5: Der Kontrollfluß in Cyrano läuft vom Parser zu der Behaviorkomponente. Diese entscheidet, ob eine Rückfrage generiert werden muß (Generator) oder ob die Aktion ausgeführt wird und der neue Zielzustand visualisiert werden muß (Grafikmodul). Anschließend geht die Kontrolle zurück an den Parser.

eine weitere Ebene nach unten verzweigt. Durch diesen Mechanismus wird eine kontrollierte, sinnvolle Reaktion auf dynamische Veränderungen garantiert. Der Kontrollfluß der Behavior-Aktivierung entspricht dem traditionellen Aufruf einer Prozedur. Da jedoch jede Behaviorhierarchie eine Menge von parallelen Prozeduren beschreibt, kann man eher von einem parallelen Prozeduraufruf sprechen. Hat sich nach der Aktivierung eines Behaviors der Status der Welt geändert, visualisiert das Grafikmodul den Zustand und gibt anschließend die Kontrolle an die Behavior-Komponente zurück. Ist die aktuelle Handlungsanweisung durchgeführt, geht die Kontrolle zurück an das Parsingmodul, welches auf die nächste Anweisung wartet. (siehe Abb. 2.5)

Das Szenario von Cyrano

Das gewählte Szenario gleicht dem des Videospiels Sokoban. In diesem Spiel steuert der Benutzer einen Lagerarbeiter durch eine zweidimensionale Darstellung einer Lagerhalle. Mehrere Kisten sind auf verschiedene Positionen verteilt. Ziel des Spieles ist es, die Kisten so zu verschieben, daß eine ausgezeichnete Zielstellung der Kisten erreicht wird. Der Lagerarbeiter ist dabei ausschließlich in der Lage, die Kisten zu *schieben*. Hierdurch können Züge

nicht rückgängig gemacht werden. Abbildung 2.6 zeigt eine mögliche Startkonfiguration des Spieles.
Der Lagerarbeiter (im folgenden Agent) verfügt über einen Satz an Basisfähigkeiten:

- Bewegung in die vier "Himmelsrichtungen"
- Steht der Agent unmittelbar neben einem Block, so kann dieser in Laufrichtung verschoben werden, jedoch nicht falls sich hinter dem Block ein weiterer Block oder ein Hindernis befindet.
- Minimale Sensorik zur Feststellung, ob ein Hindernis/Block im Weg ist.

Das gewählte Szenario erfüllt eine Reihe von Anforderungen, die eine Realisierung des Projektes ermöglichen:

- einfache Struktur der möglichen Aktionen des Agenten
- hinreichende Komplexität der von dem Agenten durchführbaren Aktionen
- leichte Erweiterbarkeit erlaubt die Simulation mehrerer Agenten
- Programmierung auf einem PC (Verfügbarkeit von Hard- und Software)

Simulation der dynamischen Welt

Der Agent soll in einer dreidimensionalen Welt agieren und diese aus einer Perspektive entsprechend seiner aktuellen Position in dieser Welt wahrnehmen. Hierbei soll sich die Simulation der Sensorik an Erfahrungen anlehnen, die ein Mensch, der in einer ähnlichen Situation agiert, erfährt. Entfernte Objekte sollen durch nähere verdeckt werden; der Agent verfügt über einen Horizont. Er nimmt immer nur Teilausschnitte der Welt wahr. Außerdem verfügt seine Sensorik über eine festgelegte Ausrichtung. Er nimmt also nur vor ihm liegende Objekte wahr.
Das Szenario läßt 3 Arten von Objekten zu:

- statische Objekte, die Teil des Labyrinths sind
- dynamische Objekte, die als Blöcke verschiebbar sind
- dynamische Objekte, die als Feindagenten Aktionen beeinflussen

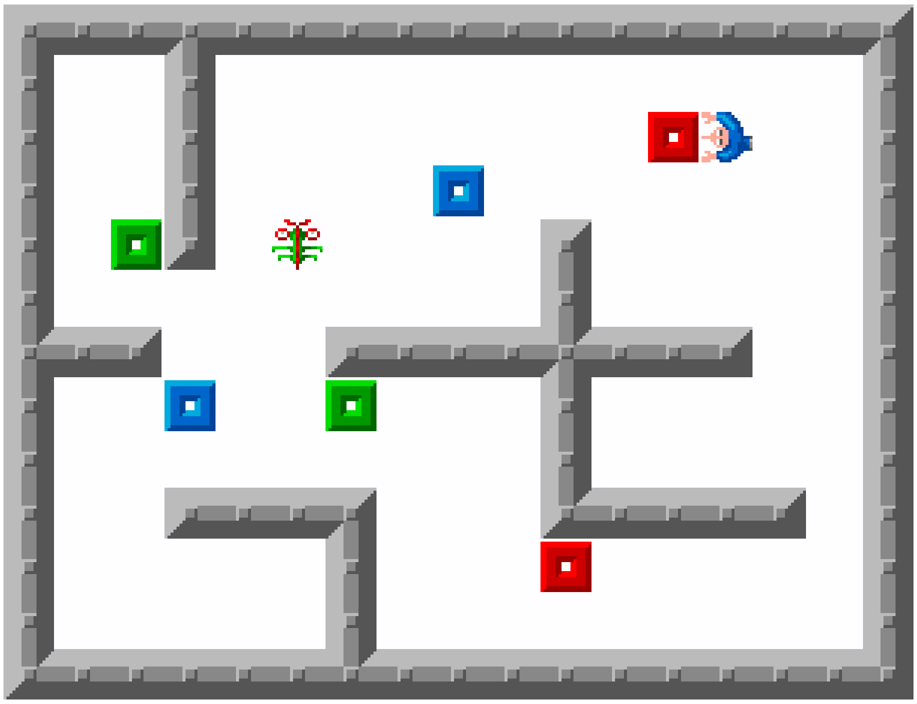

Abbildung 2.6: Eine mögliche Startkonfiguration von Cyrano. Der teilautonome Agent Cyrano verschiebt, als Reaktion auf eine natürlichsprachliche Anweisung, die farbigen Blöcke. Er plant Handlungsfolgen und setzt diesen Plan reaktiv, unter Einbeziehung sensorischer Daten, um. Seine Wahrnehmung (optisch, wie auch akustisch) ist gerichtet und eingeschränkt. Der feindliche Agent Bert (die Biene) bewegt sich in einem zufälligen Muster durch das Labyrinth. Cyrano nimmt Bert lediglich akustisch wahr.

Ein besonderes Problem der Kommunikation stellt die situationsangepaßte Wahl der raumbezogenen Ausdrücke dar. Bei menschlicher Kommunikation führt der Sprecher in aller Regel eine Transformation des Bezugssystemes durch, so daß der Hörer in seinem Bezugssystem über absolute Raumangaben verfügt. Zusätzlich werden eine Vielzahl von Voraussetzungen gemacht, welche eine redundanzfreie Kommunikation ermöglichen:

- menschliche Handlung ist raum- und zeitgebunden
- was Menschen wahrnehmen und sprachlich ausdrücken, setzt das Vorhandensein eines Raumes voraus
- sprachlich formulierbare Rauminformationen sind durch eine Reihe kognitiver Beschränkungen bedingt

Entsprechend muß der menschliche Instruktor, der den künstlichen Agenten steuert, sich in das Bezugssystem des Agenten versetzen und raumbezogene Ausdrücke anpassen.
Hierbei stellt man fest, daß Schwierigkeiten bei der Transformation auftreten. Offensichtlich bereitet die Perspektive (eine vertikale Aufsicht), mit der das System visualisiert wird, den meisten Instruktoren Probleme. Als Konsequenz ergibt sich für Systeme, die künstliche Agenten einsetzen, die Forderung, die Visualisierung derart zu gestalten, daß möglichst wenige, insbesondere keine "unnatürlichen", Transformationen durchgeführt werden müssen.

Die Basisaktionen

Die Simulation läßt 2 Aktionen des Agenten zu.

- Drehung um die eigene Achse, zur Veränderung von Blick- und Bewegungsrichtung. Vier Positionen sind möglich (N,S,E,W).
- Bewegung in die aktuelle Bewegungsrichtung, jeweils ein Schritt.

Aus diesen Basisaktionen sind alle weiteren zusammengesetzt.

Der Visionoperator

Der Visionoperator greift auf die Weltrepräsentation zurück und liefert einen Ausschnitt aus dieser. Dabei ist eine Voreinstellung der Größe des Ausschnittes durch den Benutzer möglich. Das Sichtfeld des Agenten ist ein Rechteck, welches vor ihm liegt (in Richtung der "Nase"). Er nimmt also vor ihm und unmittelbar seitlich von ihm liegende Objekte wahr. Vom Standpunkt des

Agenten aus werden lineare Trajektorien zum Rand des Sichtfeldes verfolgt. Mit ihrer Hilfe wird berechnet, welche Objekte sichtbar sind und welche durch näher am Agenten liegende Objekte verdeckt sind.

Der Höroperator

Um auf "Feinde" reagieren zu können, verfügt der Agent über die Fähigkeit, innerhalb eines definierten Radius zu "hören". Er stellt also fest, ob sich ein Feind in seiner Nähe befindet und kann so auf ihn reagieren. Eine prototypische Reaktion ist die Flucht vor dem Feind. Entsprechend kann die Richtung, aus der sich der Feind nähert, festgestellt werden. Der Agent verfügt also über ein gerichtetes Gehör.

Die Parser/Generator Komponente

Die Parser/Generatorkomponente hat die Aufgabe, den Dialog mit dem Agenten mittels natürlicher Sprache zu ermöglichen. Es wird ein herkömmlich realisierter Chart-Parser verwendet, der eine kontextfreie Phrasenstrukturgrammatik parst. Die aufgebaute syntaktische Struktur wird anschließend in eine Frame-Struktur übersetzt. Die Umsetzung erfolgt anhand einer Tabelle, die für die in der syntaktischen Beschreibung auftretenden Symbole ein semantisches Äquivalent enthält. Zusätzlich enthält die Frame-Struktur ein Flag, welches anzeigt, ob eine Anweisung von dem angesprochenen Behavior verstanden wurde ist oder nicht. Das Behavior filtert aus den angebotenen Informationen diejenigen heraus, die relevant für seine Ausführung sind. Sind nach der Filterung noch Ambiguitäten feststellbar, wird die Struktur an den Generator übergeben, welcher eine entsprechende Nachfrage generiert. Mit Hilfe der Antwort wird die Struktur dann vervollständigt. Sind die gegebenen Informationen immer noch nicht ausreichend, wiederholt sich der Zyklus (Abb. 2.7). Erst nach erfolgreicher Ausführung der Anweisung durch das Behavior wird die Frame-Struktur "weggeworfen". Diese Frame-Struktur repräsentiert also ein minimales Diskursgedächtnis.

Die Testsätze

Es werden drei Stufen von Anweisungen unterschieden:

- *einfache Anweisungen*: "gehe nach vorn" durch direkte Koppelung an Behaviors der untersten Stufe. Keine Rückkoppelung zwischen Parser/Generator und Behaviors
- *nicht quantisierte Anweisungen*: "schiebe den roten Block nach vorn", die bei referentiellen Ambiguitäten Rückfragen zur Folge haben

- *quantisierte Anweisungen*: "schiebe den linken hinteren blauen Block drei Schritte nach links"

Der folgender Beispielsatz ist eine quantisierte Anweisung:
"Schiebe den linken hinteren Klotz drei Schritte nach rechts"

Das Ergebnis seiner syntaktischen Analyse ist:

```
((VP (VT SCHIEBE)
     (NP (DET DEN) (AP (A LINKEN) (AP (A HINTEREN))) (N KLOTZ))
     (DGR (NUM DREI) (GN SCHRITTE))
     (PP (PREP NACH) (ADVDIR RECHTS)))) 1
```

Hieraus ergibt sich die semantische Umsetzung:

```
(SCHIEBEN (BLOCK ((X-KOORDINATE LINKS) (Y-KOORDINATE HINTEN)))
          (SCHRITTE 3)
          (RICHTUNG RECHTS)
)
```

Die Behaviorkomponente

Die Behaviorkomponente stellt die zentrale Steuereinheit des künstlichen Agenten dar. In ihr fließen alle Informationen zusammen. Sie steuert die Ankoppelung der sprachlichen Daten an die Sensordaten und ermöglicht durch "geschicktes" Zusammenspiel der Behaviors eine sinnvolle, situationsangepaßte Reaktion des Agenten. Der Agent verfügt über 9 Behaviors, die auf 5 Hierarchiestufen verteilt sind. Behaviors höherer Stufen können dabei Behaviors niedrigerer Stufen beliebig anstoßen.

- weglaufen: steuert das Ausweichen vor dem Feindagenten Bert. Es konkurriert mit den durch eine Anweisung getriggerten Behaviors.
- zu Block: steuert die Bewegung zu einem eindeutig identifizierten Objekt innerhalb des Sichtbereiches.
- schieben: steuert das Verschieben eines Blockes in Bewegungsrichtung, solange dieses möglich ist.
- zu Ort: bewegt den Agenten zu einer ausgezeichneten Position innerhalb der simulierten Welt. Es wird von "zu Block" aktiviert.
- n Schritte: bewegt den Agenten n Schritte in Bewegungsrichtung.

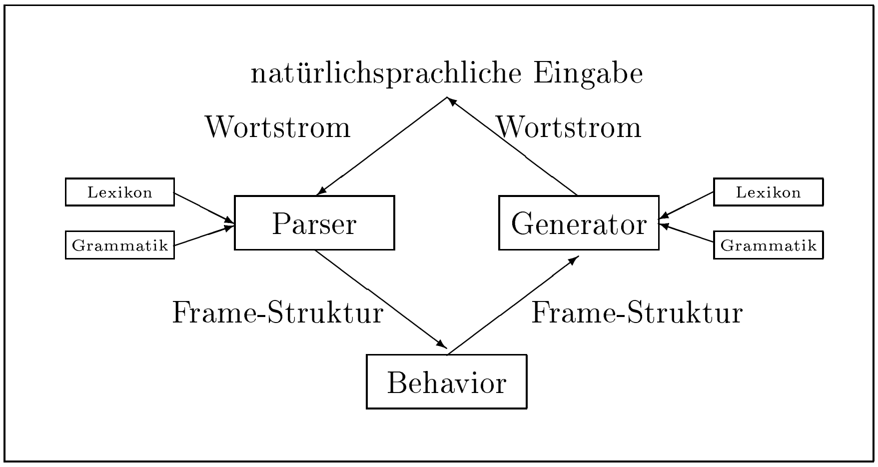

Abbildung 2.7: Die Eingabeschleife von Cyrano ermöglicht die einfache natürlichsprachliche Interaktion mit dem System. Der Parser überführt die Eingabe in eine interne Repräsentation, welche eine Aktion spezifiziert. Liegen ausreichend Informationen für die Durchführung der Aktion vor (sprachlich oder sensorisch), wird sie ausgeführt. Andernfalls werden über die Generierungskomponente solange Rückfragen erzeugt bis ausreichende Informationen vorliegen.

- umdrehen: verändert die Blick- und Bewegungsrichtung um 180^0.
- links: verändert die Blick- und Bewegungsrichtung um 90^0 entgegen dem Uhrzeigersinn.
- rechts: verändert die Blick- und Bewegungsrichtung um 90^0 im Uhrzeigersinn.
- vorwärts: bewegt den Agenten einen Schritt in Bewegungsrichtung, sofern dies möglich ist.

Konfliktauflösung bei konkurrierenden Behaviors

Tritt zwischen den konkurrierenden Behaviors eine Situation auf, in denen beide aktiviert werden, muß entschieden werden, welches sich durchsetzt. Zur Konfliktauflösung wird jedem Behavior ein Aktivationsgrad zugeordnet. Anhand dieses Wertes wird entschieden, welches Behavior prominent ist. Unterschiedliche Steigerungskoeffizienten sorgen dafür, daß sich "lebenserhaltende" Behaviors durchsetzen (etwa das Fluchtbehavior *weglaufen*).

Die Grafikkomponente

Der Grafikkomponente kommt die Aufgabe zu, den aktuellen Status der simulierten Welt zu visualisieren. Neben der Darstellung der Welt, des Agenten, des Feindagenten und der verschiebbaren Blöcke bietet sie dem Instruktor Einsicht in die derzeit aktiven Behaviors. Außerdem wird das jeweils aktuelle Sichtfeld des Agenten gezeigt. Dies ermöglicht sozusagen einen Einblick in das Innere des "Gehirnes" des Agenten. Mit Hilfe dieser Zusatzinformationen ist ein experimentelles Kommunizieren mit dem Agenten möglich.

2.5.2 Samsa - ein mobiler Roboter

Um die Testergebnisse, die mit Hilfe der Simulation Cyrano erarbeitet wurden, zu bestätigen, wurde zusätzlich der mobiler Roboter Samsa [2] entwickelt. Das System besteht aus der *mobilen Basiseinheit*, dem *Steuerungsinterface*, dem *Sensorinterface* und dem *Steuerungsrechner*.

Die Basiseinheit

Die mobile Basiseinheit ist ausschließlich aus Teilen der Firma *fischertechnik* realisiert. Tragendes Element des Roboters ist die Grundplatte. Motoren, Sensorinterface und Steuerungsinterface sind auf der Unterseite montiert. Auf der Oberseite befindet sich der Steuerungsrechner und die Energieversorgung.
Das System verfügt über zwei unabhängig steuerbare Motoren. Initiale Laufungenauigkeiten der Motoren werden durch Potentiometer ausgeglichen. Sie verhindern, daß der Roboter "hinkt". Weiterhin ermöglichen sie in einem weiten Bereich eine Feinregulierung der Maximalgeschwindigkeit des Roboters.

Verwendete Sensorik

Samsa verfügt über eine ausgesprochen einfache Sensorik. Es werden Taster der Firma *fischertechnik* verwendet. Diese dienen zum einen als digitale Kontaktsensoren, zum anderen werden sie als Impulsgeber zur Messung der Motorbewegung verwendet.
Im vorderen Bereich sind die Kontaktsensoren hinter einem flexiblen, halbkreisförmigen Kunststoffring angebracht. Dieser dient gleichzeitig als Stoßdämpfer. Durch geschickte Positionierung der Kontakttaster wird eine genauere Einschätzung der Kontaktrichtung möglich. An den Seiten und an der Rückseite steht jeweils ein Kontaktsensor zur Verfügung.
Die an den Motoren angebrachten Kontaktgeber liefern 2 Impulse pro Raddrehung. Damit kann die Position auf wenige Zentimeter genau festgestellt werden. Positionierungsfehler durch mangelhafte Traktion können nicht festgestellt werden.
Als kontaktfreie Entfernungssensoren werden IR-Fototransistoren/Dioden verwendet. Sie sind ringförmig auf Höhe der Grundplattenkante angebracht. Die Lichtmenge wird analog gemessen und stellt ein Maß für die Entfernung eines Objektes dar. Es wird also nicht das herkömmliche Echolotprin-

[2] Samsa = *S*ituated *A*rchitecture for a *M*obile *S*pecial *A*gent

Abbildung 2.8: Samsa: Seitenansicht
Der Roboter wird durch den auf der Montageplatte angebrachten Computer gesteuert. Hinter dem Steuerungsrechner ist die Stromversorgung untergebracht. Das Motor/Sensor Interface ist auf der Unterseite zwischen den Motoren installiert. Im vorderen Teil erkennt man den Kontaktsensorring, hinten links ist ein Lichtsensor sichtbar.

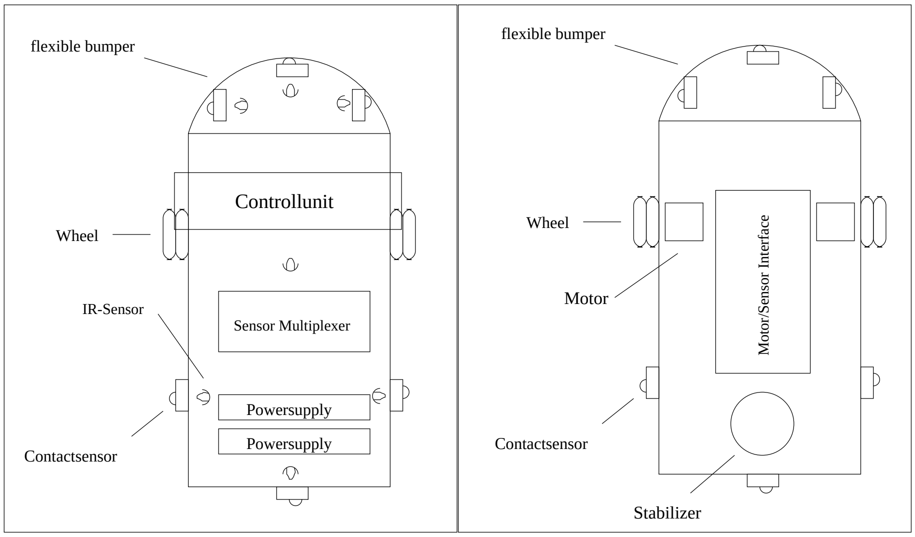

Abbildung 2.9: Samsa: Ansicht von oben und unten
Die Skizze zeigt die eingesetzten Komponenten und ihre physikalische Anordnung auf dem System. Steuerungsrechner, Energieversorgung und Sensormultiplexer liegen auf der Oberseite des Systems. Das Motor/Sensor-Interface und die Antriebsmotoren sind auf der Unterseite angebracht. Der Hauptteil der Sensorik ist in Fahrtrichtung ausgerichtet. Zusätzlich sind Sensoren an den Seiten und am Heck des Systems installiert.

zip zur Entfernungsmessung verwendet. Die Entfernungsmessung orientiert sich an der "normalen" Funktionsweise des Auges und ist dadurch biologisch adäquater. Zur Senkung der Fehlerwahrscheinlichkeit wird parallel zum eigentlichen Meßwert ein unabhängiger Referenzwert gemessen. Wenn die prozentuale Änderung von Referenzwert und Meßwert gleich ist, wird davon ausgegangen, daß kein Objekt detektiert wurde, sondern lediglich eine allgemeine Absenkung der Lichtintensität vorliegt. Die Anordnung der Sensoren zeigt Abbildung 2.9.

Der Steuerungsrechner

Als Steuerungsrechner wurde ein Standard-PC eingesetzt. Das verwendete Gerät ist eine Kleinstcomputer mit nur sehr beschränkter Rechenleistung. Entsprechend konnten nur einfache Steuerungshierarchien getestet werden. Für die Programmierung wurde auch hier wieder ein modifiziertes LISP ver-

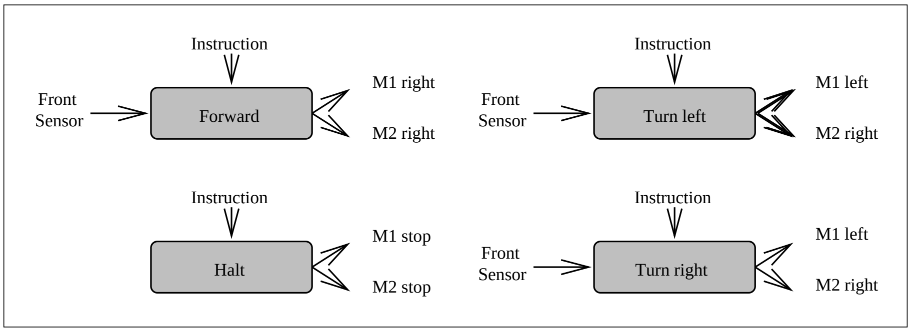

Abbildung 2.10: Basisbehaviors
Die Basisbehaviors von SAMSA (`forward`,`halt`,`turn right` und `turn left`) sorgen für die Bewegung des Systems unter Berücksichtigung von Sensordaten.

wendet, welches um Routinen, die den Zugriff auf die externe Interfacehardware zuließen, erweitert wurde.
Auf LISP-Ebene wurde ein Scheduler implementiert, welcher die Aktionsselektion ausführt. Entscheidend für die Aktionsauswahl sind der Sensorzustand und das durch die sprachliche Instruktion vorgegebene Ziel. Im wesentlichen wurde die innerhalb von Cyrano verwendete Architektur übernommen.

Die Behaviors

Die Steuerungshierarchie von SAMSA besteht aus 4 Basisbehaviors (forward, halt, turn left, turn right) und 4 komplexeren Behaviors (move left, move right, turn left on light, turn right on light). Sie sorgen dafür, daß SAMSA sich durch seine Umwelt bewegen kann. Durch taktile Rückkoppelung über die Sensoren weicht der Agent Hindernissen aus. Die Lichtsensoren sorgen für die Ausrichtung und Bewegung zu einer starken Lichtquelle hin. Dieses *homing*-Verhalten ist jedoch nicht als abstraktes Behavior vorgegeben, sondern wird durch Überlagerung von Basisbehaviors erzeugt.
Alle vorgegebenen Behaviors sind durch sprachliche Instruktionen ansteuerbar. Sprachdaten werden, wie bei Cyrano, als Sensordaten aufgefaßt. Ihre Verarbeitung und Interpretation unterscheidet sich nicht von der Verarbeitung der weiteren Sensordaten.
Die Instruktionsvorverarbeitung sorgt für die Aufbereitung der Instruktionen. Das Ergebnis steht dem Behaviorsystem zur Verfügung und jedes Behavior interpretiert die aktuelle Anweisung. Dabei werden von jedem Behavior nur die relevanten Daten analysiert.

Beispielinstruktion: `fahre nach vorne`

Die Vorverarbeitung liefert: `(action move direction forward)`

Entsprechend wird das `forward` Behavior aktiviert.

2.6 Maschinelles Lernen in der Robotik

Das folgende Kapitel gibt einen Überblick über Standardverfahren des Maschinellen Lernens. Der Schwerpunkt liegt auf Verfahren, welche im Bereich der autonomen Agenten und der Robotik eingesetzt werden.
Prinzipiell können zwei Ansätze unterschieden werden: auf der einen Seite Verfahren der klassischen KI (*symbolische Verfahren* und *logische Verfahren*), andererseits Ansätze, die neueren Strömungen der KI/Robotik folgen, hier im besonderen *Reinforcement Lernen*.
Symbolische und logische Verfahren orientieren sich in der Methodik an der Künstlichen Intelligenz bzw. an der klassischen Mathematik. Sie unterliegen damit auch den gleichen Beschränkungen. Grundlage der symbolischen Verfahren der KI ist eine Weltrepräsentation, welche den genauen Zustand des modellierten Weltausschnittes zu einem Zeitpunkt `t` repräsentiert. Durch eine Menge von Regeln werden Übergänge in der Welt bestimmt, so daß nach Vorgabe eines Start- und eines Zielzustandes eine mögliche Folge von Übergängen berechnet werden kann. Die so gewonnene Übergangssequenz beschreibt eine Folge von Handlungen, welche in der Welt durchgeführt wird, um vom durch den Startzustand repräsentierten Weltzustand in den durch den Zielzustand repräsentierten Weltzustand zu gelangen. Die vorgestellten induktiven Verfahren ID3 und ID5R basieren auf einer partiellen Weltrepräsentation durch Merkmalsbäume. Das Verfahren leitet aus den bereits repräsentierten Beispielen Merkmalsmengen für noch nicht repräsentierte Beispiele ab und ordnet diese in die Repräsentation ein. Das AQ-Verfahren wird stellvertretend für die Gruppe der Regel-Induktions-Verfahren vorgestellt.
Werden symbolische Lernverfahren eingesetzt, so wird auf der Grundlage von Trainingsläufen eine Menge von Testdaten generiert, welche symbolisch repräsentiert werden. Die so symbolisch repräsentierten Daten werden durch das Lernverfahren analysiert (i. A. offline). Durch Klassifikation der Daten werden induktiv oder deduktiv Regelhaftigkeiten abgeleitet, welche als Ergebnis des Lernverfahrens dem Robotersystem verfügbar gemacht werden.
Die Tabelle 2.1 gibt eine Übersicht über Robotiksysteme, welche klassische Lernverfahren einsetzen (s.a. [58], [59], [36], [7], [4], [15], [21], [32], [33], [41], [65], [78], [66] [39] [40] [68] [67] [16] [92]).

ID3: Bei dem ID3 Verfahren ([64]) wird Top Down ein Entscheidungsbaum aufgebaut, welcher die Testbeispiele klassifiziert. Die Trainingsbeispiele müssen hierzu klassifiziert werden. Interne Knoten des Baumes repräsentieren den Test bezüglich eines spezifischen Attributes. Jedes mögliche Attribut erhält einen Unterbaum; die Blätter repräsentieren den Klassennamen.

	Induktiv		Deduktiv	
	Benutzer	Automatisch	Benutzer	Automatisch
Robot Programs	Heise 89 Sato und Hirai 87	Duffay und Latombe 84	Selfridge und Levas 83	
Action Plans	Langley et al 89 Tan 90		Segre 88	
Operators	Kadie 91 Laird et al 90			Bennett 90
Situation-Action-Rules	Bartenstein und Inoue 87 Chen 86	Langley et al 89 Christiansen 90		Mitchell 90
Error Diagnostics			Zheng und Daneshmend 91	Zercher 92
Classifiers		Langley et al 89 Tan 90		

Tabelle 2.1: Klassifikation lernfähiger Systeme
Übersicht existierender lernfähiger Systeme im Bereich Manipulatoren/Robotik (nach [36]). Die Systeme verwenden symbolische Lernverfahren. Je nach System wird die Bewertung der Lernleistung durch den Benutzer bzw. automatisch vorgenommen. Die Systeme arbeiten i.d.R. ohne aufwendige Sensorik und auf stark eingeschränkten Domänen.

Der Baum wird schrittweise aufgebaut, wobei das Attribut gesucht ist, welches die beste Partition (Aufteilung) der Beispielmenge liefert. Die beste Unterteilung ist dabei definiert als diejenige, welche einen maximalen Informationsgewinn bezüglich eines Attributes liefert. Mögliche Erweiterungen des ID3 Verfahrens erlauben die Verwendung von verrauschten Attributen und unbekannten Attributwerten.

ID5R: Das ID5R Verfahren ([85]) ist eine Erweiterung des ID3, wobei hier der Entscheidungsbaum inkrementell aufgebaut wird. Im Endresultat, d.h. nach der Präsentation aller Trainingsbeispiele, unterscheiden sich der vom ID3 Verfahren und der vom ID5R Verfahren konstruierte Entscheidungsbaum nicht. Beim ID5R Verfahren wurde weiterhin die Datenstruktur der Knoten derart erweitert, daß der Informationsgewinn der Attribute gespeichert werden kann. Diese Information wird bei der Restrukturierung des Entscheidungsbaumes während der Trainingsphase genutzt. Diese tritt auf, wenn das aktuell getestete Attribut nicht das ist, welches den maximalen Informationsgewinn liefert. Der Baum wird dann in zwei Schritten restrukturiert:

1. Durchlaufe alle Unterbäume und suche das Attribut t, welches maximalen Informationsgewinn liefert.
2. Tausche aktuelles Testattribut mit t.

AQ Verfahren: AQ Verfahren sind von Michalski [58], [59] eingeführt worden. Das Verfahren gehört zu den Regel-Induktions-Verfahren. Der Algorithmus generiert eine Menge von Klassifikationsregeln (jeweils eine Regel pro Klasse). Die Regel besteht aus der Bedingung (cover), die eine Disjunktion von sogenannten *Komplexen* ist. Jeder Komplex ist eine Konjunktion aus *Selektoren*, wobei ein Selektor der Test eines Basisattributes ist.

Um ein neues Objekt zu klassifizieren, werden die vorhandenen Regeln überprüft. Ist die Bedingung genau einer Regel vollständig erfüllt, so ist das Beispiel abgedeckt und fällt damit in die beschriebene Klasse. Werden die Bedingungen mehrerer Regeln erfüllt, so wird die allgemeinste abgedeckte Klasse gewählt. Ist die Bedingung keiner Regel erfüllt, so wird das Beispiel in die am häufigsten verwendete Klasse einsortiert.

Die Regeln für die einzelnen Klassen werden sequentiell aufgebaut. Im ersten Schritt wird ein nicht-klassifiziertes positives Beispiel eingefügt. Durch Angabe eines komplett abdeckenden Covers *star*, also einer Menge von Komplexen, die genau das positive und kein negatives Beispiel abdecken, wird das Cover initialisiert. Im nächsten Schritt wird das Cover durch Hinzufügen eines weiteren Complexes erweitert. Das Auswahlkriterium für das Complex ist dabei problemabhängig zu definieren. Durch die Erweiterung des Covers wird die Menge der klassifizierten positiven Beispiele, welche zu der beschriebenen Menge gehören, (möglicherweise) vergrößert. Entsprechend werden diese Beispiele einsortiert und aus der Menge der Trainingsbeispiele entfernt. Eine Erweiterung des AQ-Verfahrens ist das CN2 Verfahren ([17]). Dieses ermöglicht die Klassifizierung unscharfer Daten (positiv/negativ).

2.7 Reinforcement Lernen für situierte Agenten

Bevor die Lernverfahren für situierte Agenten in 2.7.2 beschrieben werden können muß in einem kurzen Exkurs auf die wesentlichen Eigenschaften von situierten Agenten eingegangen werden.

2.7.1 Eigenschaften situierter Agenten

Nach Kaelbling [35] sind wesentliche Eigenschaften eines *situierten Agenten* rauschende Sensorik, nicht deterministische Aktionen, Reaktivität, Inkrementalität, begrenzte Trainingszeiten und die Verankerung in der Umweld (grounded agent). Der Entwickler kann nur Wissen kodieren, welches auch zur Entwicklungszeit zur Verfügung steht. So ist es etwa bei der Entwicklung eines explorativen Systems, dessen Ziel es ist, Karten von unbekanntem Terrain anzulegen, nicht möglich, Hindernisse und Pfade a priori festzulegen. Die Bewegungssteuerung muß zur Laufzeit auf die Gegebenheiten des zu explorierenden Terrains angepaßt werden.
Durch die Lernfähigkeit des Agenten ergibt sich für den Entwickler des Steuerungssystems eine Arbeitserleichterung. Es ist schwer, Wissen zu kodieren und dem Agent einzuprogrammieren. Besonders motorische Vorgänge sind komplex in ihrer Beschreibung. Man denke z.B. an den Vorgang des "Schleifebindens" (siehe auch Tabelle 2.2). Eine textuelle Beschreibung dieses Vorgangs ist kompliziert. Die Beschreibung dieses Vorgangs erreicht bei weitem nicht den Detaillierungsgrad, welcher notwendig ist, um ihn prozedural zu beschreiben. Greifvorgänge, Beschreibungen von komplexen Teilhandlungen, Endbedingungen, Attributbeschreibungen usw. sind unscharf definiert bzw. hängen stark von der aktuellen Situation ab. Die Beschreibung ist nicht universell; sie stellt nur eine von vielen möglichen Varianten des Schleifebindens dar. Aus diesem Grund wird Handlungswissen in der Regel durch

1. Man nimmt in jede Hand ein langes Ende des Schnürsenkels, überkreuzt die beiden und zieht das eine Ende unter dem anderen durch. 2. Dann werden die beiden Enden so auseinandergezogen, daß sich der entstandene Knoten festzieht. 3. Während der Knoten festgedrückt wird, bildet man aus dem linken Schnürsenkelteil eine ca 5 cm lange Schlaufe. Das linke Schnürsenkelende wird zur linken Seite gelegt. 4. Um das untere, offene Ende der Schlaufe legt man den rechten Schnürsenkelteil einmal herum und schiebt ihn durch die entstandene Lücke zwischen Knoten, Schlaufe und rechtem Schnürsenkelteil, so daß eine zweite ca. 5 cm lange Schlaufe entsteht. 5. Schließlich zieht man den neuen Knoten fest, indem beide Schlaufen an ihrem geschlossenen Ende auseinandergezogen werden.

Tabelle 2.2: Die deskriptive Darstellung eines komplexen Handlungsverlaufes reicht i.d.R. nicht für eine prozedurale Kodierung des Handlungsvorganges aus. Handlungen sind im allgemeinen durch "Vormachen" leichter vermittelbar.

Demonstration (in Verbindung mit textueller Beschreibung) weitergegeben.
Das Lernen numerischer Funktionen, also die Parameteranpassung, ist das zur Zeit am besten untersuchte Verfahren bei autonomen Systemen. Der Agent verfügt über eingebautes Wissen, welches seine Aktionen bestimmt. Dieses vordefinierte Verhalten wird durch das Lernverfahren nur geringfügig geändert. Es werden lediglich Parameter der einzelnen Verhaltensweisen an nur bis zu einem vordefinierten Maß von der erwarteten Situation unterschiedliche Situationen angepaßt (getuned). Einen häufig verwendeten Ansatz stellen hier ANN (artificial neural networks) dar.
Die Verfahren, die anhand eines vordefinierten Weltmodells lernen, stellen die Standardverfahren der KI dar (z. B. ([38], [55]). Der Vorteil ist hierbei,

daß die Repräsentation durch den situierten Agenten "gegrounded" werden, also in der Welt verankert sind. Es zeigt sich jedoch, daß das Erlernen neuer Verhaltensweisen auch ohne die Zwischenstruktur einer abstrakten Repräsentation möglich ist. Des Weiteren treten alle Nachteile auf, welche sich aus dem repräsentationsbasierten Ansatz ergeben (Konsistenzhaltung, aufwendige Verfahren, damit keine Situiertheit mehr).
Das Lernen der Koordination von Behaviors ist ansatzweise gelöst, etwa durch RL Verfahren ([49] [53]). Hat jedoch die Steuerungshierarchie eine gewisse Komplexität überschritten, treten alle Standardprobleme des RL auf.
Der Erwerb neuer Behaviors ist zur Zeit beschränkt auf die Sequenzbildung von einfachen Behaviors. Weitergehende Lösungen, etwa die Bildung von generalisierten Verhaltensweisen ist noch nicht gelöst, es existieren auch noch keine Verfahren. Reinforcement Lernen ist asymptotisch komplett und ist in der Lage, abstrakte Behaviors aufzubauen. Jedoch muß bei neu auftretenden Situationen jeweils von neuem gelernt werden. Der Erwerb längerer Ketten von Behaviors ist extrem zeitaufwendig, da das Verfahren nur langsam konvergiert.

Unsicherheit

Das größte Problem bei der Entwicklung von realen Agenten ist der Umgang mit Unsicherheiten. Unsicherheit entsteht dabei nicht nur durch fehlerhafte Modellierung der Umwelt und dynamische Weltänderungen, sondern auch durch die verwendete Sensorik und Aktuatorik. Sensordrift und Aktuatorungenauigkeit bewirken eine unvorhersehbare Modifikation von deterministischen Aktionen. Hierdurch entstehen Inkonsistenzen zwischen erwartetem und erreichtem Zustand.
Herkömmliche Steuerungsprogramme arbeiten mit einer idealisierten Weltsicht. Durch eine extrem genaue Fahrtplanung und durch leistungsfähige, sehr gut positionierbare Steuermotoren und Greifhände sollen Ungenauigkeiten und Unsicherheiten aufgefangen werden. Entsprechend werden auch die Simulationssysteme entworfen. Für die einfache Fahrtplanung eines Manipulators können damit noch ausreichend exakte Ergebnisse simuliert werden. Bereits bei Fügeprozessen sind die Ergebnisse so unrealistisch, daß sie kaum noch Relevanz für die Entwicklung haben.
Unsicherheiten ergeben sich für den Agenten immer dann, wenn Informationen mittels Sensorik gewonnen werden. Hierbei erzeugt zum einen die begrenzte Auflösung (und Genauigkeit) der Sensoren Unsicherheit in den gewonnenen Daten. Weiterhin muß für die Verarbeitung der Sensor-Rohdaten eine Umwandlung des analogen Sensorsignales in einen zugeordneten digita-

len Meßwert erfolgen, welcher ebenfalls ungenau ist.
Nach Brooks [12] ergeben sich folgende Möglichkeiten der Behandlung von Unsicherheit:

- Ignorieren von Unsicherheit
- Vermeidung von Unsicherheit
- Einbeziehung von Unsicherheit

Der erste Ansatz führt zu leicht zu steuernden, jedoch extrem unflexiblen Agenten, welche in statischen Block-Welten agieren.
Um Unsicherheit zu vermeiden, wurden Robotersysteme mit extrem akkuraten und sensiblen Sensoren ausgestattet. Durch aufwendige Analyseverfahren wurde versucht, alle Unsicherheiten zu erkennen und sie zu eliminieren. Die grundlegende Annahme war, daß das Sensorsystem den Aufbau einer exakten Repräsentation der Umwelt zuließ.
Im Gegensatz hierzu wird bei der Entwicklung von behaviorbasiert gesteuerten Robotern von vorne herein die Unsicherheit der Welt als relevant angenommen. Damit sind die Ergebnisse der Simulation direkt für reale Agenten nutzbar. Die so entwickelten Roboter sind sowohl flexibler, als auch kostengünstiger und schneller zu entwickeln.
Nach diesem Prinzip ist auch das von Malcolm/Smithers [54] entwickelte System SOMASS konstruiert worden. Es steuert einen Manipulator, welcher aus hölzernen Grundelementen komplexe geometrische Konstruktionen aufbaut. Ziel des Systems ist es, wie in herkömmlichen Ansätzen, die planende Komponente mit idealisierten Daten arbeiten zu lassen, um so die Komplexität zu reduzieren. Hierzu wurde ein hybrides System aus einer planenden und einer subkognitiven, behaviorbasierten Komponente aufgebaut. Die subkognitive Komponente wird als virtuelle Maschine aufgefaßt, die die Schritte der planenden Komponente ausführt und die Behandlung von Unsicherheit und Ungenauigkeiten übernimmt. Malcom/Smithers ([54]) unterscheiden 4 Typen von Unsicherheit:

- Unsicherheit der initialen Position des verwendeten Teils
- Abweichung in der (geometrischen/physikalischen) Form des Teils
- Ungenauigkeiten bei der Positionierung der Teile während der Montage
- Ungenauigkeiten, die Folgeschritte in der Montage unmöglich machen, z.B. durch Schwerkraft bedingte Dimensionsänderung, also Unsicherheit bzgl. der Montageabfolge

2.7.2 Verfahren des Reinforcement Lernens

Wesentlich für die Kategorisierung von Reinforcement Verfahren ist es, in welcher Weise die Trainingsdaten (überwachtes oder unüberwachtes Lernen) und in welcher Weise der Feedbackwerte generiert werden (s. a. [76]). Man unterscheidet zwischen

- skalarem Feedback (policy-only)
- Kontrollfeedback (Reinforcement Comparison)
- analytischem Feedback (AHC-Learning, Q-Learning)

Beim skalaren Feedback berechnet eine vordefinierte Bewertungsfunktion den skalaren Feedbackwert. Skalares Feedback läßt sich, in Abhängigkeit von der Anzahl der errechneten Feedbackwerte, in weitere Unterklassen subkategorisieren. So wird bei Sutton ([76], [77]) der Feedbackwert lediglich einmal errechnet. Andere Ansätze erzeugen den Feedbackwert, sobald ein Fehler auftritt, z.B. Barto [75] beim Erlernen des Balancieren eines Stabes. Bei Singh [71] wird der Feedbackwert in Zwischenstadien berechnet, während z.B. Maes und Brooks [10] nach jedem Schritt einen Feedbackwert generieren (auch Pierce und Kuipers, 1991), Watkins [86],[87]).
Kontrollfeedback erweitert den Ansatz durch die Verwendung eines Reward-Predictors. Dieses System macht Handlungswissen, welches beim skalaren Feedback nur implizit in den Steuerungseinheiten kodiert ist, explizit und liefert in Abhängigkeit von der aktuellen Situation eine Vorhersage über die Höhe des zu erwartenden Feedbacks. Durch den Vergleich der beiden Werte wird es möglich zielorientierte Vorgaben in den Lernschritt einfließen zu lassen (Asada und Young, 1989).
Verfahren mit analytischem Feedback erweitern den Ansatz so, daß die Vorhersageheuristik des Reward-Predictors sich in Abhängigkeit von dem tatsächlichen Feedback schrittweise anpasst. Dadurch wird es einerseits möglich, daß Zielvorgaben Einfluß auf das Lernergebnis haben, andererseits passen sich die Zielvorgaben (z.B. bei einer drastisch veränderten Umweltsituation) den Gegebenheiten schrittweise an ([87]). In diese Klasse fällt auch das von Wengerek ([88]) eingesetzte Verfahren. Man kann 3 Klassen von Lernverfahren unterscheiden:

- Q-Learning
- Classifier Systems
- Modellbildende Systeme

2.7.3 Q-Learning

Q-Learning stellt einen Basisansatz bei der Entwicklung lernfähiger Agenten dar (s.a.[76]).

Q-Learning: Hauptmerkmale des Q-Learnings sind:

- Für jedes Aktions-Situations Paar bewertet der Agent die durchgeführte Aktion.
- Zu Beginn wird eine Matrix initialisiert, welche jede Kombination von Aktion und Situation aufnehmen kann.
- Nach jedem Schritt wird die Matrix aktualisiert. Es wird der maximale kumulative Reward, welcher für eine Aktion innerhalb der Situation möglich ist, berechnet und in der Matrix abgelegt.
- Zeitlich nähere Werte werden höher bewertet als zeitlich entferntere.

Beim Q-Learning wird diejenige Aktion gewählt und ausgeführt, welche den höchsten Wert hat. Lernen findet statt durch die Ausführung einer Aktion bei anschließender Bewertung und Aktualisierung der Wertematrix. Die Explorationsstrategie ist nicht festgelegt; in der Regel wird nach der Methode verfahren, von Zeit zu Zeit eine zufällige, also nicht optimale, Aktion durchzuführen und diese zu bewerten.

Als Vorteil des Q-Learning ergibt sich damit, daß der Agent nach endlicher Zeit eine optimale Lernstrategie hat und dadurch die optimale Aktionsauswahl findet.

Nachteilig ist, daß keine zeitvarianten Ziele vorgegeben werden können. Wird das Ziel geändert, so muß der komplette Lernvorgang wiederholt werden. Der Zustandsraum ist extrem groß. Längere Aktionssequenzen brauchen viel Zeit, um gelernt zu werden bzw. werden nicht erlernt. Voraussetzung für das Verfahren ist, daß der Agent zu jedem Zeitpunkt weiß, in welchem Zustand er sich befindet. Initialwissen ist nur schwer einzubringen. Das Verfahren ist schwer zu erweitern, falls parallele Aktionen möglich sind. [3]

[3] Sutton definiert hierzu in [77] zwei Beispielprobleme, die durch Q-Learning nicht gelöst werden können. Unter der Voraussetzung, daß ein Agent mittels Q-Learning den optimalen Weg durch ein Labyrinth erlernt hat, ergeben sich das *Shortcutproblem* und das *Blockingproblem*. Beim Shortcutproblem wird nach der Lernphase ein Hindernis entfernt. Der Agent ist nicht in der Lage, dies festzustellen, sondern folgt weiterhin dem gelernten (jetzt suboptimalen) Weg zum Ziel.

Die von Sutton entwickelte DYNA Methode des Q-Learnings integriert modellbasiertes Lernen und Q-Learning. Das Verfahren konvergiert wesentlich schneller.

Classifier Systems: Classifier Systems sind ein Spezialfall des RL. Das System besteht aus einer Menge an Regeln (Classifier) und Daten über die Regelausführung. Die Aktionsauswahl erfolgt über den Test (partielle Übereinstimmung) der Vorbedingung des Classifiers (interne und externe Daten). Classifier, deren Vorbedingung erfüllt sind, werden ausgewählt.

Gelernt wird durch Weitergabe der Aktivation von erfolgreichen angewendeten Classifiern an (gerade vorher) aktivierte Classifier. Der Bucket-Brigade Algorithmus behandelt das Creditassignment Problem: falls eine Aktion erfolgreich ausgeführt wurde, wird diese bewertet und anschließend die Aktivationsstärke des Classifiers erhöht.

Die Explorationsstrategie des Ansatzes ist einfach. Die Anzahl der Classifier ist konstant. Von Zeit zu Zeit (z.B. nach einer vom Entwickler vorgegebenen Anzahl von durchgeführten Aktionen) werden Classifier mit niedriger Stärke entfernt und durch mutierte/rekombinierte erfolgreiche ersetzt.

Als Vorteil ergibt sich hier, daß eine etwas verbesserte Explorationsstrategie angewendet wird, welche nicht nur zufällige Punkte des Zustandsraumes wählt. Die zugrundeliegende Hypothese ist hierbei, daß eine kleine Änderung in einer (guten) funktionierenden Lösung zu einer verbesserten Lösung führt. Durch die Verwendung von "#" (*don't care*) ergibt sich ein besseres Generalisierungsverhalten, welches es erlaubt, den Zustandsraum auf unterschiedlichen Abstraktionsniveaus zu untersuchen. Damit kann die maximal abstrakte Repräsentation eines Classifiers gefunden werden, die eine Lösung des Problems liefert. Das Ausschalten erfolgloser Classifier verbessert das Aktionsauswahlverhalten.

Nachteilig bei diesem Verfahren ist, daß ständig das aktuell verfolgte Ziel geändert wird. Das Lernverfahren startet aus einer "tabula rasa" Situation heraus. Es ist gedächtnislos, und somit können Mutationen ständig wiederholt getestet und wieder verworfen werden.

Beim Blockingproblem wird ein neues Hindernis eingefügt. Der Agent lernt jetzt den kompletten Weg neu, obwohl lediglich das neue Hindernis umwandert werden müßte und dann der alte Weg weiterverfolgt werden könnte. Hier zeigt sich das Grundproblem des Q-Learnings: als nicht modellbasierte Methode wird keinerlei Abstraktion geleistet.

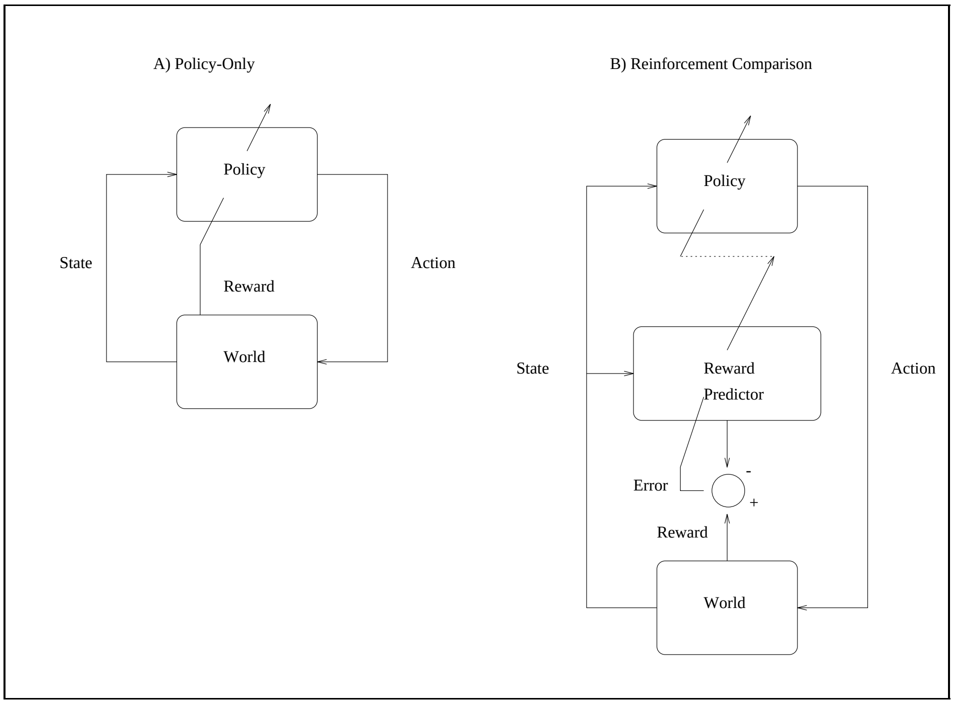

Abbildung 2.11: Reinforcement Strategien
a) Beim Policy-Only RL wird der Reward ausschließlich von Situationsinformationen abgeleitet. Damit geht in das Lernverfahren kein abstrahiertes Handlungswissen ein (außer das mittelbar in den Behaviors kodierte).
b) Beim RL mit Reinforcement Comparison wird der aktuelle Rewardwert mit einem berechneten Rewardwert verglichen, um so eine Abschätzung zu erhalten, inwieweit die aktuelle Situation mit der erwarteten Situation übereinstimmt. Im Reward Predictor sind Ausschnitte des Handlungswissens kodiert, so daß in Abhängigkeit vom Zustand des Systems, Voraussagen über den zu erwartenden Reward gemacht werden können.

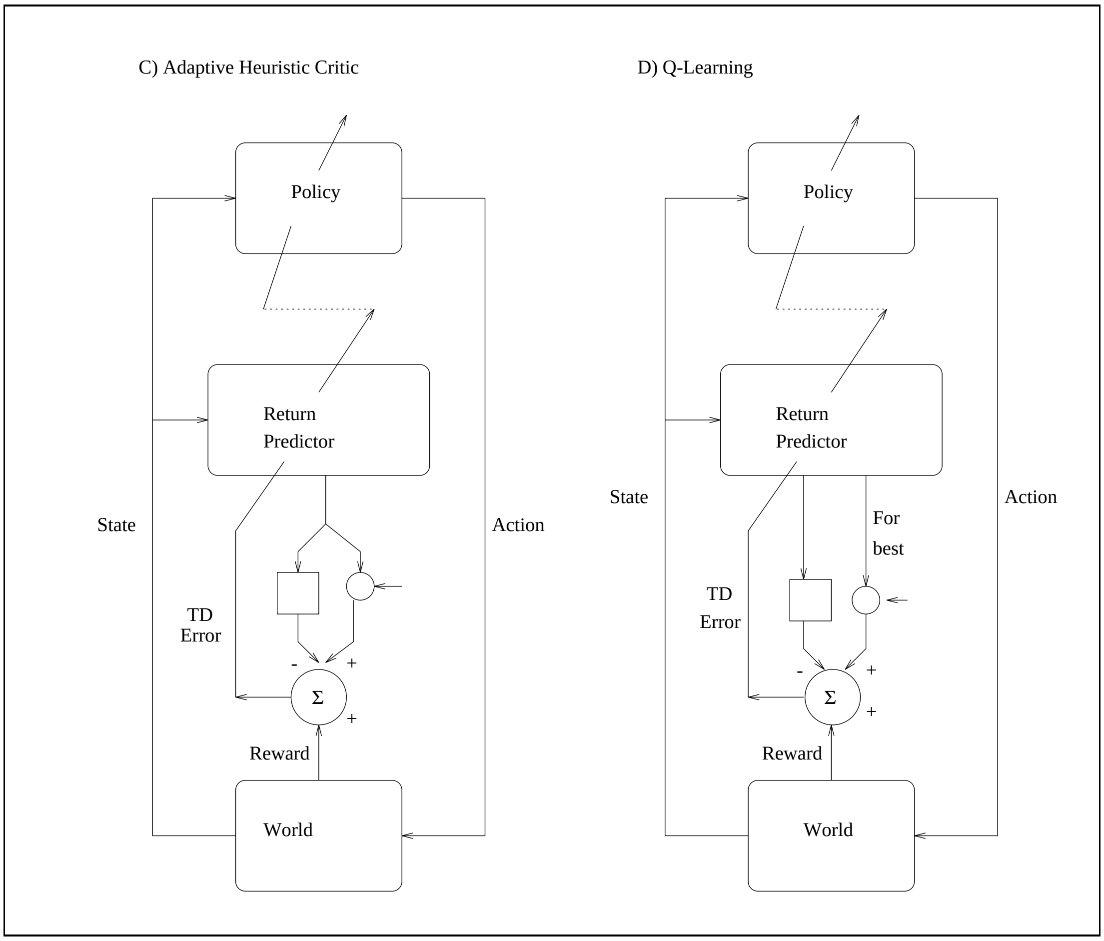

Abbildung 2.12: Reinforcement Strategien
c) Beim RL-Verfahren Adaptive Heuristic Critic (AHC) wird die durch den Reward Predictor berechnete Voraussage gewichtet. Das Verfahren versucht dabei im Laufe des Experiments die Gewichtung so anzupassen, daß erwarteter und tatsächlicher Reward sich annähern, bzw. übereinstimmen.
d) Q-Learning führt eine Gewichtung der Reward Voraussage durch, wobei der jeweils beste Wert stärker in die Gewichtung eingeht, als suboptimale Voraussagen.

Modellbildende Systeme: Modellbildendes Lernen stellt den komplexesten Ansatz des Reinforcement Lernens dar. Mit modellbildenden Verfahren soll ein Kausalmodell für die Abhängigkeit von Situation und Aktion aufgebaut werden. So nutzt beispielsweise Drescher ([20]) die Theorien Piagets, um ein probabilistisches Modell für die Effekte von spezifischen Aktionen innerhalb bestimmter Situationen aufzubauen. Dieses Modell wird wiederum verwendet, um zu entscheiden, welche Aktion innerhalb einer neuen Situation optimal ist. Aktionsauswahl und Lernverfahren sind hier entkoppelt. Das Lernverfahren kann dadurch in unterschiedlichen Aktionsauswahlverfahren eingesetzt werden. Zu bemerken ist, daß die meisten Verfahren lediglich "wesentliche" Aspekte einer Situation lernen, also beispielsweise solche, welche für die Aktionsauswahl relevant sind.

2.7.4 Credit assignment

Das wichtigste zu lösende Problem beim RL (allgemein beim Lernen) ist das Problem des Credit assignments, also die Zuordnung des durch einen Reinforcementwert definierten Effekts zu einer stattgefundenen (Sequenz von) Aktion(en). Je nachdem wie diese Zuordnung stattfindet, hat das Lernverfahren ein unterschiedliches Generalisierungsverhalten.
Connell ([19]) formuliert das CA Problem wie folgt (siehe auch Abb. 2.13): Zu einem bestimmten Zeitpunkt t erhält der Agent einen Reward. Die Frage ist nun, bis zu welchem Zeitpunkt im Generalisierungsvolumen G dieser Reward gültig ist.
Der in Grafik 2.13 aufgespannte Raum wird definiert durch die Achsen *Zeit, Kontrollparameter* und *Sensorwerte*. Ein Punkt definiert also die gewählten Kontrollparameter, die durch die Sensorik wahrgenommene Situation und den Zeitpunkt, an dem dies stattgefunden hat. Sowohl Kontrollparameter wie auch Sensorwerte lassen sich nochmals zu einem n-dimensionalen Raum aufspannen, was die Komplexität des CA weiter erhöht.
Man kann drei Arten von CAP definieren: das *strukturelle* CAP, das *temporale* CAP und das *Kontroll* CAP.
Das strukturelle CAP beschreibt die Problemstellung, wie der Reward über den Zustandsraum verteilt ist, d.h. wie groß ist die Wahrscheinlichkeit, daß bei unterschiedlichem Anfangszustand eine Sequenz von Aktionen zu dem gleichen Ergebnis, also zu dem gleichen Reward führt. Das temporale CAP beschreibt die Problemstellung, welche Folge von Aktionen innerhalb der Aktionspyramide G verantwortlich ist für den erhaltenen Reward. Das Kontroll-CAP schließlich beschreibt die Problemstellung, ob eine geringfügig andere Sequenz von Aktionen zu dem gleichen Ergebnis, also dem gleichen Reward,

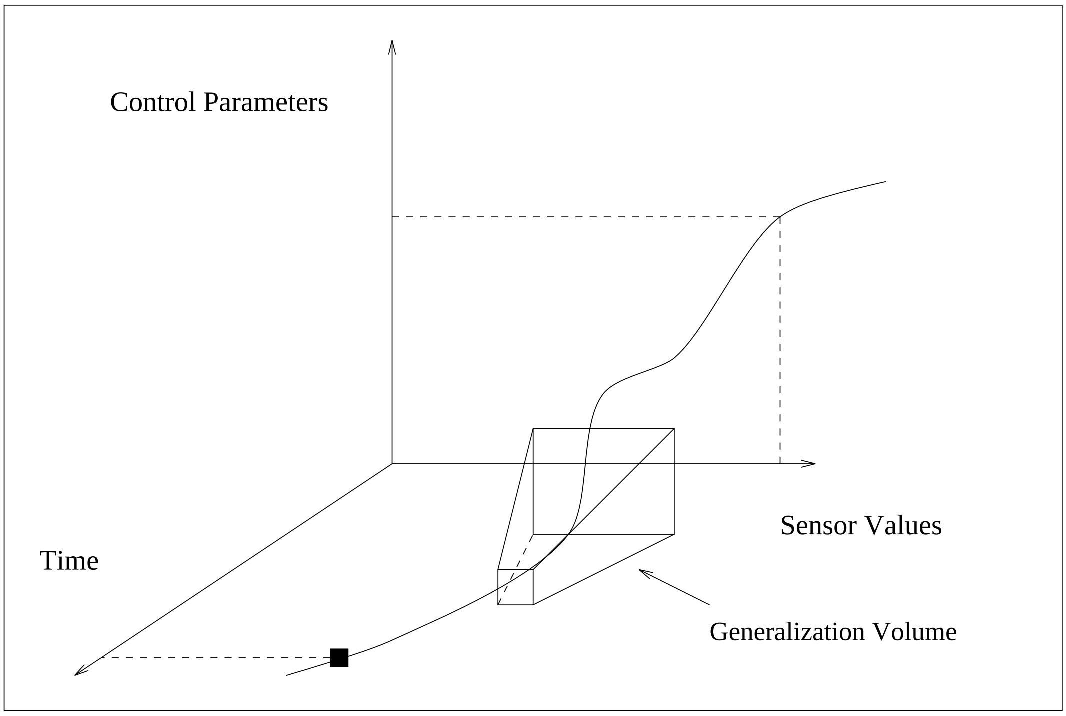

Abbildung 2.13: Innerhalb des Kontrollraumes (definiert durch Zeit, Kontrollparameter und Sensordaten) muß derjenige Zeitpunkt `t` identifiziert werden, der innerhalb einer definierten Varianz (Generalisierungsvolumen) ursächlich ist für eine Handlungssequenz (Abbildung nach Sutton, 1984).

führt.

2.7.5 Strukturelle CA

Ansätze, die das strukturelle CAP lösen, basieren auf der Idee der Funktionsapproximation. Es wird versucht, die Folge von Punkten im Zustandsraum durch eine funktionale Beschreibung anzunähern. Man kann zwischen dem *regionbasierten* und dem *punktbasierten* Ansatz unterscheiden.
Beim regionbasierten Ansatz wird der Zustandsraum in Regionen aufgeteilt, wobei Punkte der gleichen Region die gleiche Aktion spezifizieren (z.B. kd-trees (Friedmann et al, 1977, Moore, 1990)). Beim punktbasierten Ansatz wird über die k nächsten Punkte interpoliert.

2.7.6 Temporale CA

Die meisten Lösungsansätze basieren auf dem *bucket brigade* Verfahren (Samuel 1959, Holland et al. 1986). Hierbei wird der Reward schrittweise an die jeweils nächste aktivierte Kontrolleinheit (hier Behavior) weitergegeben. Dabei können Ausweichsequenzen, welche nicht für den Reward relevant sind, definiert werden. (Q-Learning, Watkins und Dayan, 1992 und Sutton, 1988)

2.8 Vergleich von Ansätzen zur reaktiven Steuerung

Eine Vielzahl von Ansätzen sind zur Realisierung von reaktiven Systemen vorgeschlagen worden. Im folgenden werden die drei relevantesten Ansätze beschrieben. Es handelt sich um

- die Subsumptionsarchitektur von Brooks
- den dynamischen Ansatz von Steels
- die Aktionsselektion (AS) von Maes

Eine weitergehende Darstellung von Ansätzen behaviorbasierter Systeme findet sich in Tyrell ([84]), Brooks ([12]), [8], Steels ([72]),Maes ([50]), [52], [51], [53], [48], [49].

Subsumptionsarchitektur: Der Ansatz von Brooks ist der erste Versuch, behavior-orientierte KI zu realisieren. Er stellt damit den 'Klassiker' der vorgestellten Ansätze dar [11]. Grundlage des Ansatzes ist die Subsumptionsarchitektur. Das Steuerungsprogramm wird hierbei durch eine Menge von endlichen Automaten (augmented finite state machine = AFSM) realisiert. Diese sind hierarchisch geordnet. Die AFSMs beschreiben dabei eine situationsabhängige Verhaltenweise (Behavior) des Agenten. Jeder Automat arbeitet dabei ausschließlich auf den für ihn relevanten Sensordaten. Es wird kein Weltmodell aufgebaut.

Innerhalb einer Schicht können sich die Behaviors durch Inhibition und temporale Kontrolle beeinflussen. Hierdurch wird z.B. die Sequenzbildung von Verhaltensweisen unterstützt.

Ein Nachteil der Subsumptionsarchitektur ist ihre statische Struktur. Zur Laufzeit können keine neuen Behaviors hinzugelernt werden.

Ein Beispiel eines mit AFSMs gesteuerten mobilen Roboters ist das System Herbert von Jonathan Connell [18]. Dieser mobile Roboter ist in der Lage, in einer unstrukturierten, sich dynamisch verändernden Umwelt (Labor) eine Aufgabe (Einsammeln leerer Getränkedosen) zu erfüllen. Abbildung (2.14) zeigt die dafür notwendige Behaviorstruktur.

Dynamischer Ansatz: Steels definiert ein Behavior als ein *ausschließlich von außen beobachtbares Verhalten, welches nicht unmittelbar mit einer eindeutig innerhalb des Agenten zu lokalisierenden Routine übereinstimmt.* Wesentliches Merkmal intelligenter Agenten ist damit die *Dynamik* der

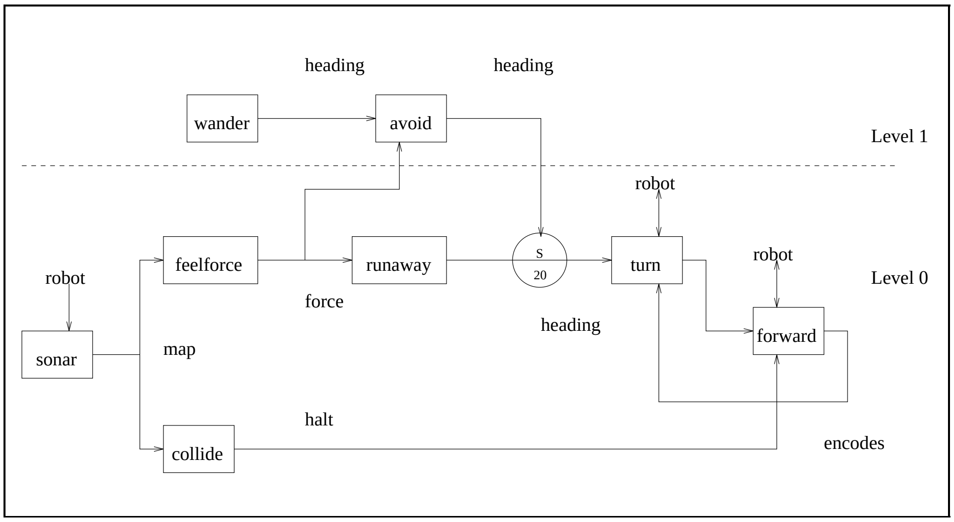

Abbildung 2.14: Ein Ausschnitt aus der Behaviorstruktur von Herbert. Der mobile Roboter ist in der Lage, Getränkedosen innerhalb einer unstrukturierten, dynamischen Umwelt zu lokalisieren, zu greifen und zu einem vordefinierten Ablagepunkt zu transportieren.

Systeme. Diese Definition eines Behaviors entspricht der von McFarland [56] gegebenen.

Die Modellierung der Steuerungsstruktur ist bei Steels nicht hierarchisch. Alle Behaviors sind innerhalb einer Ebene nebenläufig. Jedes Behavior wird dabei als klassischer Regelkreis (Behaviormodul) realisiert, welcher durch den sensorischen Input beeinflußt werden kann. Die Behaviors sind aus *Routinen* aufgebaut. Eine Routine ist eine Sequenz von Anweisungen, welche den sensorischen Input auf den aktuatorischen Output abbildet. Zur Beschreibung von Behaviormodulen definiert Steels die *Process Description Language, PDL*, eine einfache, deklarative Sprache, welche in den Zielcode der verwendeten Steuerungsprozessoren kompiliert wird. PDL ermöglicht die schnelle Entwicklung einfacher Steuerungssysteme. Leider unterstützt sie den Entwickler nicht bei methodologischen Problemen, die bei der Konstruktion behaviorbasierter Systeme auftreten. Die Programmierung erfolgt auf einer nicht hinreichend abstrakten Ebene.

Um auf die Sensoren zugreifen zu können, führt Steels *Inputquantities* ein. Dies sind definierte Speicherstellen, auf welche die Sensoren

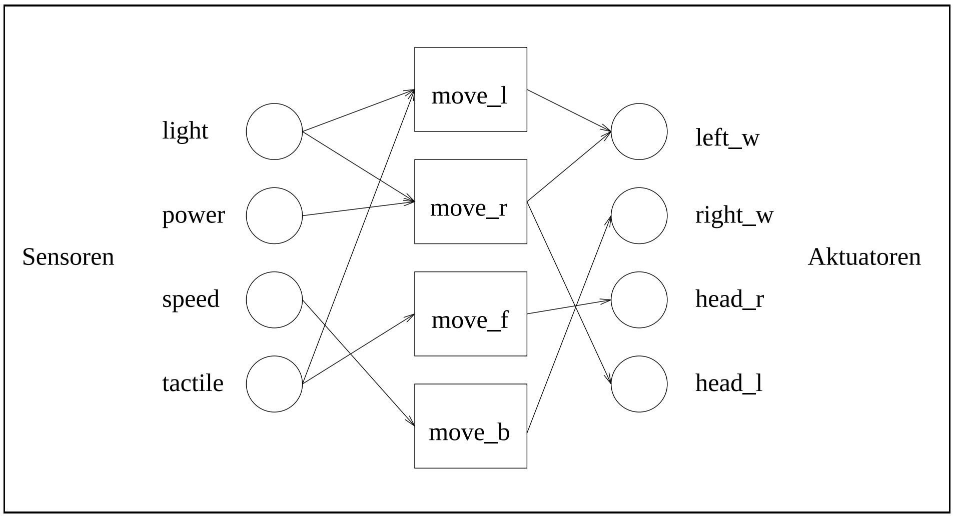

Abbildung 2.15: Ein Behaviorsystem nach Steels mit Zugriff auf Input- und Outputquantities. Die Behaviormodule (rechteckige Kästchen) haben lesenden Zugriff auf die Inputquantities und schreibenden Zugriff auf die Outputquantities. Alle Behaviormodule sind parallel nebenläufig.

schreibend und die Behaviorroutinen lesend zugreifen können. Zwischen Inputquantities und Sensoren besteht eine 1:1 Abbildung. Der Zugriff auf die Aktuatoren erfolgt über die *Outputquantities*. Diese entsprechen in ihrer Funktion den Inputquantities, wobei die Behaviorroutinen schreibend und die Aktuatoren lesend zugreifen können. Sofern eine Synchronisation zwischen den Behaviorroutinen notwendig ist, erfolgt diese über *internal quantities*. Auf diese können die Behaviorroutinen sowohl lesend wie auch schreibend zugreifen.

Aktionsselektion: Ein weiterer Ansatz für die Entwicklung behaviorbasierter Systeme stammt von Maes. [50]. Beim *Action Selection*-Verfahren (AS) ist das Behaviorsystem ein verteiltes, rekurrentes, nicht hierarchisches Netzwerk. Die Knoten repräsentieren Verhaltensweisen, sind jedoch strukturell einfach gehalten. Es existieren drei Typen von Kanten: Vorgänger-, Nachfolger- und Konfliktlösungskanten. Das Netzwerk erhält Eingaben über externe Stimuli (durch die Sensoren) und interne Stimuli (Motivation). Nachdem das Netzwerk über die Eingabekanäle Aktivation erhalten hat, wird diese schrittweise über das

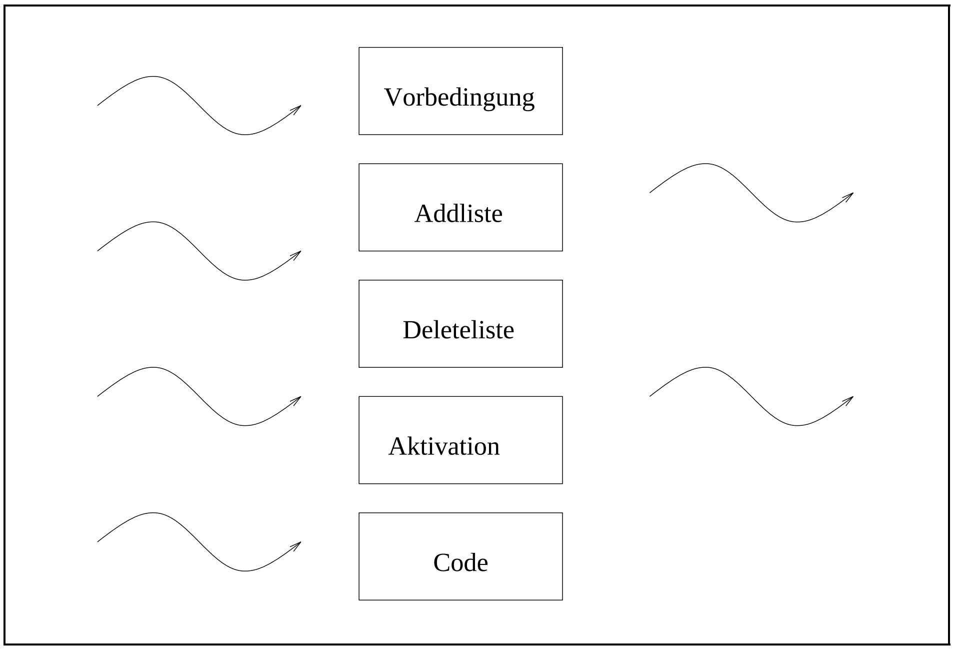

Abbildung 2.16: Der Knoten innerhalb eines Action-Selection Netzwerkes besteht aus fünf Komponenten, erhält Input über vier Kantentypen und erzeugt Aktivation, die über die Ausgabekanten weiterverteilt wird.

Netzwerk verteilt. Jeder Knoten innerhalb des Netzwerks besteht aus fünf Komponenten:

- Vorbedingung
- Addliste
- Deleteliste
- Aktivation
- ausführbarer Code

Das Aktionsselektionsverfahren arbeitet in vier Schritten. Es verfährt nach der Methode *Winner-takes-all*. Falls möglich, wird genau ein Knoten ausgewählt, der dann die Aktionskontrolle über das System erhält. Zuerst wird die Aktivation, welche sich durch externe und interne Stimuli ergibt, in das Netzwerk eingegeben. Anschließend wird die Erregung über die Kanten in das Netzwerk verteilt. Die eigentliche Aktionsauswahl erfolgt im dritten Schritt. Es wird anhand der

Vorbedingung und des Aktivationswertes geprüft, ob Knoten aktivierbar sind. Falls dies der Fall ist, wird der Knoten mit der höchsten Aktivation ausgewählt und dessen Code ausgeführt. Ist kein Knoten ausführbar, wird der globale Aktivationsschwellwert herabgesetzt und das Verfahren beginnt wieder mit dem ersten Schritt. Zur genaueren Einstellung des Gesamtverhaltens eines durch Aktionsselektion gesteuerten Systems existieren eine Anzahl globaler Parameter:

- der mittlere Aktivationswert
- der Startwert des globalen Aktivationsschwellwerts, welcher nach jedem Schritt um Δ Prozent herabgesetzt wird
- Konstante zur Gewichtung der externen Stimuli
- Konstanten zur Gewichtung der Kantentypen (Vorgänger, Nachfolger, Konfliktlöser)
- Konstante zur Gewichtung der Zielspezifikation

Ein wesentliches Problem bei der Entwicklung behaviorbasierter Systeme nach diesem Ansatz liegt in dem schwer nachzuvollziehenden Einfluß der globalen Parameter auf das Gesamtverhalten. Schon kleinere Änderungen eines Globalparameters können bei völlig unveränderter Netzwerktopologie und gleichem Experimentalsetting zu einem völlig veränderten Gesamtverhalten führen.

Völlig ungeklärt ist auch, wie bei der Entwicklung derartiger Netzwerke systematisch vorzugehen ist. Da durch die Modellierung als flaches Netzwerk keine explizite hierarchische Ordnung vorgegeben ist, muß der Entwickler ein sehr genaues Verständnis über die gewünschten Fähigkeiten des Systems haben. Soll in einer Situation eine Sequenz von Verhaltensweisen durchgeführt werden, so müssen die entsprechenden Gewichtungen der Vorgänger- und Nachfolgerkanten gesetzt werden. Dies ist bei einigen wenigen Behaviors noch einfach, stellt sich aber bei zunehmender Anzahl als immer komplizierter dar. Ein weiteres Problem sind die Aktivationsbedingungen. Jede Vorbedingung ist an genau ein Behavior gekoppelt und löst, bei Vorhandensein des externen Stimulus, genau eine Reaktion aus. Dies beschränkt das Behavior auf eine mögliche Aktion. Wünschenswert wäre die flexible Definition der Vorbedingung, so daß höhere Verhaltensweisen Behaviors auf niedriger Stufe parametrisieren können. Dies hätte zur Folge, daß Handlungskompetenz von höheren Schichten in die zugrundeliegenden tieferen Schichten übertragen würde.

Die Behaviorhierarchien innerhalb der geschilderten Ansätze werden jeweils von den Entwicklern definiert. Während der Laufzeit sind sie nicht veränderbar. Die so konstruierten Systeme sind in der Lage, innerhalb einer unstrukturierten Umwelt zu agieren und durch die vorgegebenen Verhaltensweisen auf dynamische Weltveränderungen zu reagieren; sie können jedoch diese Verhaltensmuster nicht verändern.

2.9 Zusammenfassung

Es zeigte sich, daß die in Cyrano und SAMSA verwendete Steuerungsarchitektur, bestehend aus einer linguistischen, einer deliberativen und der zentralen behaviororientierten Komponente in den Experimenten erfolgreich einsetzbar ist und das gewünschte Ergebnis erzielt werden konnte.
Die Agenten sind (eingeschränkt) natürlichsprachlich instruierbar und in der Lage, die Instruktionen situativ zu verarbeiten. Als handelnde Systeme sind sie durch die reaktive Komponente mit einem Basisverhalten ausgestattet, welches situationsangepaßtes Handeln ermöglicht.
Besonders bei dem physikalischen System stellt sich das Problem der Wiederholbarkeit von Experimenten. Die verwendete Hardware läßt es im Prinzip nicht zu, ein Experiment unter den gleichen Bedingungen exakt zu wiederholen. Damit sind Ergebnisse aus den Experimentalläufen, welche zur Analyse der Leistungsfähigkeit des beschriebenen Lernformalismus dienen sollen, nicht sinnvoll verwendbar.
Aus der Betrachtung der innerhalb der Robotik eingesetzten Verfahren zum Handlungslernen zeigt sich, daß die klassischen Verfahren nicht den geforderten Bedürfnissen entsprechen. Die Trennung von Lern- und Auswertungsphase, der Bedarf einer externen Repräsentation des Weltzustandes und die Nutzung einer externen globalen Lernfunktion, welche über spezielles Domänewissen verfügt ist nicht mit dem angestrebten Steuerungsansatz vereinbar.
Besser geeignet erscheinen hier die Reinforcement Verfahren. Besonders Q-Learning ist ausgerichtet auf den Erwerb von Handlungsfolgen in situierten Agenten.
Dies bestätigt auch die Arbeit von Wengerek, die zeigt, daß es mit Verfahren des Reinforcement Learning möglich ist, komplexere Handlungsfolgen aus Basisreflexen aufzubauen und diese dem Steuerungsprogramm eines Roboters verfügbar zu machen. Leider sind die Grundvorrausetzungen unter denen die eingesetzte Simulation entwickelt wurde zu strikt. Durch den zu kleinen Parameterraum reduziert sich das Lernproblem in zu starkem Maße. Dies gilt sowohl für die stark eingeschränkte Handlungsfähigkeit des Systems,

wie auch für die beschränkten sensorischen Fähigkeiten. Sollen komplexere Handlungsfolgen erlernt werden wird der Parameterraum wesentlich größer, wodurch das von Wengerek verwendete Verfahren (Q-Learning) nicht mehr zu gebrauchen ist. Weiterhin ist bei der Entwicklung der Simulation die Distanz zum realen System zu groß. Die Verwendung unrealistischer Sensortypen erschwert die unmittelbare Übertragung der Ergebnisse. So mußte in der anschließenden Fallstudie erst ein komplexes Kamerasystem entwickelt werden, welches in der Lage ist, zur Simulation vergleichbare Sensordaten zu liefern.

Die Arbeit von Strippgen ist in dieser Hinsicht vorbildlich. Die Repräsentation der Umwelt ist kontinuierlich und hinreichend komplex um Experimente durchzuführen, deren Ergebnisse auf reale Systeme übertragbar sind. Durch die gute Benutzerführung wird der Entwickler im Designprozeß unterstützt. Einziger Nachteil von INSIGHT ist, daß die Simulation sich auf mobile Roboter beschränkt. Diese sind in ihren Handlungsfähigkeiten zu beschränkt.

Entsprechend ergibt sich für die Entwicklung des lernfähigen behaviorbasierten Agenten die Forderung, die Simulation von vorneherein abgestimmt auf das simulierende reale System zu entwickeln. Dies gilt für Sensorik und Aktuatorik. Das grafische Benutzerinterface sollte eine mit INSIGHT vergleichbare Qualität aufweisen.

Da sich das bei Cyrano verwendete Steuerungskonzept bewährt hat, fiel die Wahl bei der Realisierung reaktiven Komponente auf den Ansatz von Brooks. Hier ist vor allem die hierarchische Ordnung der Behaviors nützlich, welche die einfachere Integration von neu erlernten Verhaltensweisen begünstigt. Die Programmierung in LISP hat weiterhin den Vorteil, daß eine Erweiterung der Behaviorhierarchie zur Laufzeit durch den Funktionsumfang von LISP gut unterstützt wird. Die Modifikationen an dem für Cyrano entwickelten Schedulingsystem betreffen lediglich den Lernmechanismus. Dies läßt eine Verkürzung die Entwicklungszeit für den lernfähigen Agenten erwarten.

Kapitel 3

Der lernfähige Agent

Der folgende Teil beschreibt die Entwicklung des behaviorbasierten, lernfähigen Agenten. Bei der Entwicklung wurde versucht, die Anforderungen bzgl. Simulation, reaktivem Steuerungssystem und Lernverfahren umzusetzen.
Bei der eingesetzten Simulation handelt es sich um ein System, welches einen Manipulator vom Typ PUMA 260/560 simuliert. Als konsequente Eigenentwicklung wurde versucht, das System so auszulegen, daß es durch eine behaviorbasierte reaktive Steuerung kontrollierbar ist. Dies bedeutet unter anderem, daß die simulierte Sensorik (Handkamera, taktile und telemetrische Sensoren, Gelenkwertsensorik) realitätsnahe Werte liefert. Gleiches gilt für die Simulation der Aktuatorik, die die Handlungsmöglichkeiten eines echten Roboters realistisch abbilden muß. Aus diesem Grund wurde, neben der direkten Ansteuerungsmöglichkeit der verfügbaren Freiheitsgrade, auch eine konventionelle Kinematikkomponente entwickelt, welche eine "klassische" Bahnplanung erlaubt. Hierdurch sind mit dem Simulationssystem reaktive Steuerungssysteme entwickelbar, welche verhältnismäßig unproblematisch für die Steuerung des realen Pendants zu verwenden sind.
Bei der Entwicklung des Steuerungssystems mußte im ersten Schritt eine Beschreibungssprache für die zu nutzende Behaviorhierarchie definiert werden. In Syntax und Sprachumfang orientiert sich die Sprache an der von Brooks definierten Subsumptionssprache, wobei die Spezifikation der einzelnen Behaviors so erweitert wurde, daß der aufgesetzte Lernmechanismus abstrakte Behaviors definieren und in die Hierarchie eingliedern kann. Durch Einführung des Konzepts der *reaktiven Zone* wird der Brook'sche Ansatz so verändert, daß eine zielgetriebene Vorselektion potentieller Kontrollbehaviors möglich wird. Ein in der Behaviorsprache definiertes Steuerungsprogramm wird durch einen dazu entwickelten Interpreter evaluiert. Kernstück des Interpreters ist der Schedulingformalismus, welcher für die Selektion des kontrollierenden Behaviors sorgt und gleichzeitig die (für den Lernmecha-

nismus zentrale) Distributionsnachricht schrittweise verteilt. Durch Verwendung von LISP als Implementationssprache des Behaviorsystems wird eine der Komplexität entsprechende Implementation erst möglich. Die Daten/Programmäquivalenz von LISP bewirkt, daß die Konzeption und Realisierung von Interpreter und Behaviorsprache verhältnismäßig einfach durchzuführen ist, wobei sich als zusätzlicher Vorteil ergibt, daß der Behaviorsprache der volle Umfang an Kontrollstrukturen von LISP zur Verfügung steht. Unter Verwendung der Behaviorsprache ist im nächsten Schritt eine Behaviorhierarchie zur Steuerung des simulierten Manipulators entwickelt worden. Durch die Definition einer Anzahl von grundlegenden Behaviors und deren Kombination zu abstrakten Behaviors wurden drei Steuerungssysteme programmiert, welche unterschiedliche Verhaltenscharakteristika des Systems erzeugten. Zusätzlich sind Routinen zur Vorverarbeitung der generierten Sensorrohdaten entwickelt worden.
Das auf dem Behaviorsystem aufsetzende Lernverfahren ist voll in das Scheduling integriert. Die Bildung und Eingliederung abstrakter Behaviors findet kontinuierlich statt, so daß das System zur Laufzeit seine Fähigkeiten erweitert. In Abhängigkeit von der Umweltsituation, dem zugrundeliegenden Basisverhalten und der Beeinflußung des Systems durch den Experimentierenden, entstehen so komplexe behaviorbasierte Steuerungssysteme. Die Schlüsselidee für die Konstruktion des Lernverfahrens orientiert sich an Beobachtungen, welche bei der Gestaltentwicklung von mehrzelligen Lebewesen gemacht wurden. Die dynamische Überlagerung von wachstumskontrollierenden Transmitterstoffen bewirkt hier eine räumlich lokale Differenzierung des Wachstumsverhalten der Einzelzellen. Im implementierten System sorgt eine Distributionsnachricht für die schrittweise Weitergabe der Information, welches Behavior zu einem bestimmten Zeitpunkt aktiv ist. Durch Ausnutzung der Struktur der Behaviorhierarchie wird es möglich beim Zusammentreffen zweier Distributionsnachrichten ein abstraktes Behavior zu konstruieren, welches eine durchgeführte Handlungsfolge beschreibt. Dieses wird in die Hierarchie eingebettet und in folgenden Handlungen vom Scheduler mitbearbeitet.
In Analogie zur Struktur des Überblickskapitels wird in den drei folgenden Teilen auf die Entwicklung des Simulationssystems, des Steuerungssystems und des Lernverfahrens eingegangen. Im letzten Teil dieses Kapitels beschreibe ich einige mit dem System durchgeführte Lernexperimente.

3.1 Das Simulationssystem

Das Experimentalsystem, welches die Grundlage dieser Arbeit ist, besteht aus einem Agenten (Manipulator), welcher in der Lage ist, einfache Objekte (Baufixteile [1]) zu manipulieren. Die Entwicklung des Simulationssystems baut auf den Erfahrungen bei der Entwicklung der bereits beschriebenen Systeme Cyrano und SAMSA auf.
Beide Systeme sind mit der gleichen reaktiven Steuerungsarchitektur versehen, bestehend aus einer (einfachen) deliberativen Komponente, einem instruktionsverarbeitenden natürlichsprachlichen System und einer behaviorbasierten Komponente, welche eine zentrale integrierende Rolle spielt.
Ziel bei der Entwicklung des Simulationssystems war es nicht, den Manipulator durch eine hybrides System zu kontrollieren. Andererseits sollte das entstehende Steuerungssystem wenigstens prinzipiell in eine hybride Gesamtarchitektur eingebettet werden können.

3.1.1 Ein reaktiv steuerbare Armsimulation

Die Entwicklung von reaktiven, zielgerichteten hybriden Steuerungssystemen setzt eine Entwicklungsumgebung voraus, die sowohl die schnelle Erstellung unterschiedlicher Anwendungssituationen erlaubt, wie auch die schnelle und einfache Veränderung der Agentenmorphologie und seiner Sensortopologie ([30], [80]). Nur durch Auswertung des Verhaltens in unterschiedlichen Anwendungssituationen kann der Systementwickler ein ausreichendes Maß an Reaktivität sichern und damit die korrekte Funktion des Steuerungssystems belegen. Zeigt es sich, daß der entwickelte Agent in seiner Form in Anwendungssituationen nicht adäquat reagiert, muß es möglich sein, seine Gestalt bzw. die Sensortopologie zu verändern.
Das Simulationssystem soll zwei Teilaufgaben erfüllen:

1. Unterstützung der Entwicklung einer behaviorbasierten Steuerungskomponente für einen Manipulator des Typs PUMA 560/260 durch die Nutzung von Sensorik und Aktuatorik

2. Unterstützende Experimentalumgebung während des Entwicklungsprozesses durch Integration von Werkzeugen (grafisches Benutzerinterface zur einfachen, ergonomischen Interaktion mit dem System, Rekorder zur Aufnahme und Wiedergabe von Aktionssequenzen,)

[1] Baufix ist ein Konstruktionsspielzeug, welches Holzelemente wie Schrauben, Mehrlochleisten, Schraubwürfel etc. enthält.

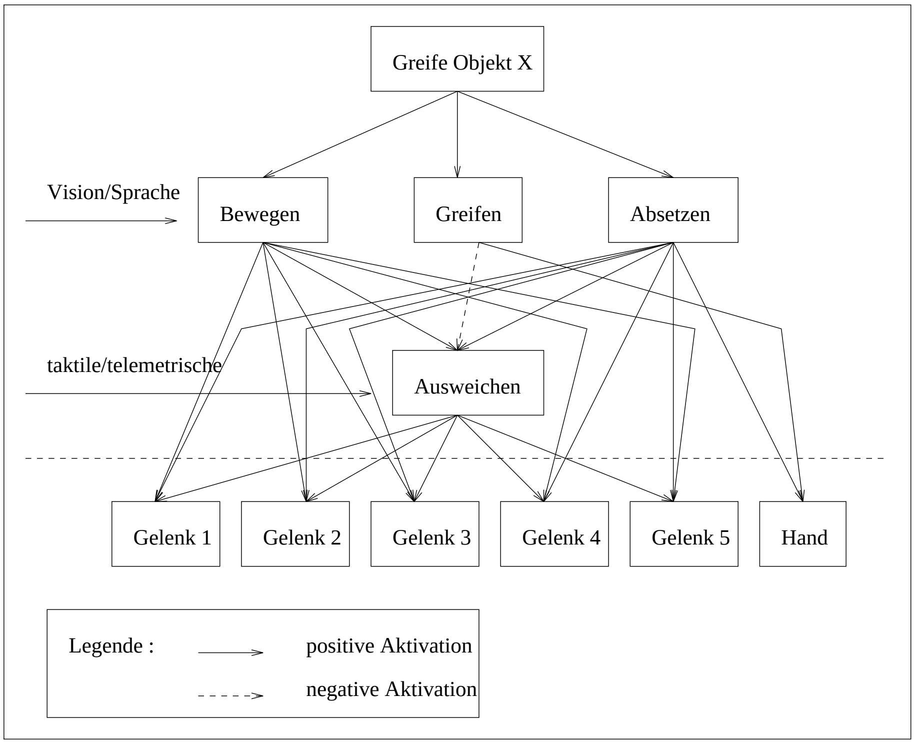

Abbildung 3.1: Behaviorhierarchie Objekt greifen
Ein mögliche Zerlegung der Funktion `Objekt greifen` umfasst drei Ebenen: das abstrakte Behavior `Objekt greifen` koordiniert die drei Behaviors `Bewegen`, `Greifen` und `Absetzen`. Auf der untersten Ebene arbeitet das Behavior `Ausweichen`, welches jedoch von `Greifen` unterdrückt werden muß.

Aus diesen beiden Grundaufgaben ergeben sich die im folgenden aufgeführten Systemanforderungen:
Zu 1): Grundlage der Steuerungskomponente ist der behaviorbasierte Ansatz. Behaviors werden verstanden als Prozesse welche eine direkte Koppelung zwischen Sensorik und Aktuatorik herstellen. Das Simulationssystem muß daher in der Lage sein, die Steuerungskomponente mit Sensordaten zu versorgen; das Simulationssystem ist also ein Generator für Folgen möglichst realistischer Sensordaten. Das Simulationssystem stellt zur Zeit

- einen Kamerasensor, montiert auf der Roboterhand
- binäre, taktile Sensoren
- telemetrische Sensoren
- Gelenkstellungssensoren

zur Verfügung.
Zu 2): Das Simulationssystem soll den Entwickler aktiv unterstützen. Dies geschieht durch die Bereitstellung einer grafischen Benutzerschnittstelle. Dabei stehen dem Entwickler die üblichen Elemente (Fenster, Menüs, Mausbedienung, Buttons, Texteingabefelder) zur Verfügung. Er ist damit in der Lage, den Betrachtungsstandort zu wechseln, Instruktionen einzugeben und Teilkomponenten, wie etwa den Datenrekorder, zu starten. Durch Repositionierung der Baufixteile können Testszenarien verändert werden.

Das entwickelte Simulationssystem stellt die Grundlage für schnelles und reproduzierbares Testen des Lern- und Steuerungssystems dar. Im Zeitrahmen dieser Arbeit war es nicht möglich, eine allgemeine Simulationsumgebung zu entwickeln, welche den oben genannte Anforderungen in allen Punkten entspricht. Vielmehr wurde pragmatisch vorgegangen und die Simulation den Anforderungen des anschließend zu konstruierenden Roboters anzupassen. Merkmale der Simulation sind:

- monolithisches System
- funktionale Beschreibung von Sensoren, Effektoren (binäre taktile, analog photometrische, analoge E-Motoren)
- Granularität in Zeit/Raum beliebig einstellbar
- wiederholbare pseudozufällige Simulationsläufe
- funktionale Trennung von Simulation und Steuerung

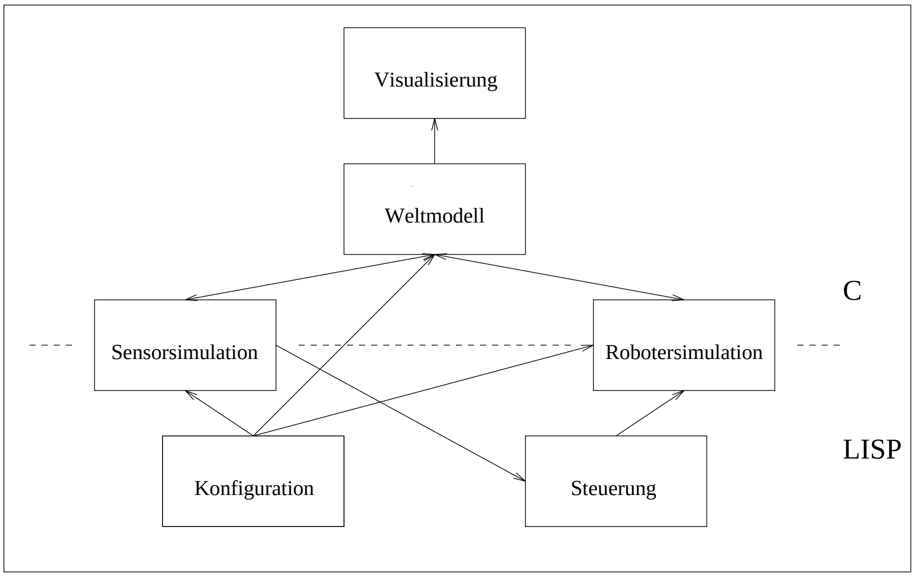

Abbildung 3.2: Implementationsschichten der Simulation. Das Simulationssystem ist weitgehend in C realisiert. Durch Schnittstellen zu LISP kann das Steuerungssystem funktional adäquater und wesentlich effizienter entwickelt werden.

- Morphologie des Roboters frei einstellbar (Anzahl Sensoren, Ausrichtung, Positionierung an Grundplatte usw.)

Die Ausgangssituation der Simulation ist in weiten Maßen frei konfigurierbar. Die Testläufe sind dabei pseudozufällig und beliebig reproduzierbar. Abbildung 3.2 zeigt die Simulationsarchitektur. Das System besteht aus 5 Komponenten:

- Visualisierung
- Robotersimulation
- Sensorsimulation
- Steuerung
- Konfigurierung

Die Simulation wurde in einem LISP-Dialekt entwickelt, wobei Teilkomponenten in C programmiert sind. Hierdurch sind zum einen die Vorteile

der funktionalen Sprache nutzbar (Programm/Daten Äquivalenz und damit Veränderung des Steuerungssystems zur Laufzeit), wie auch der Geschwindigkeitsvorteil der prozeduralen Sprache (etwa bei der Visualisierung).
Weiterhin ist der verwendete LISP-Dialekt (XLisp von David Betz) auf den wichtigsten Plattformen lauffähig. Zur Portierung des Systems müssen daher lediglich die Grafikroutinen angepaßt werden.
Die Morphologie (Größe, Masse, Sensortyp - anzahl und -platzierung, Anzahl Aktuatoren usw.) des zu testenden Agenten können beliebig gewählt werden. Für die einzelnen Sensortypen und Aktuatortypen liegen funktionale Modelle vor. Die Fehlerrate ist voreinstellbar und kann in Abhängigkeit von Zeit, Temperatur und weiteren Faktoren geändert werden.
Das Szenario sieht einen aufrecht montierten Manipulator vor, welcher mit einem Zweifingergreifer ausgestattet ist. Der Manipulator verfügt über 6 Freiheitsgrade, 5 Hauptachsen und den Greifer, der in kontinuierlichem Übergang geöffnet und geschlossen werden kann.
Der Roboter ist ebenerdig auf der Montageebene montiert. Neben dem Roboter sind Baufixteile auf der Arbeitsfläche verteilt. Die Baufixteile sind geometrisch exakt modelliert und insofern idealisiert, als keine relativen Größenschwankungen auftreten. Es ist jedoch möglich, die Prototypen beliebig zu skalieren, so daß man unterschiedliche Ausprägungen von per se gleichen Bauteilen realisiert. Diese Bauteile können von dem Greifer gegriffen und in ihrer Raumlage verändert werden.
Um den einfachen Zugang zum Simulationssystem zu gewährleisten, ist eine grafische Benutzerschnittstelle entwickelt worden. Sie ermöglicht zum einen die einfache Bedienung des komplexen Systems. Der Arm kann mittels grafischer Schiebeelemente in beliebige Positionen gebracht werden. Der Standpunkt der Benutzerkamera ist frei wählbar, wodurch Aktionssequenzen in unterschiedlichen Perspektiven visualisiert werden können. Auf der anderen Seite passt sich das System in weiten Bereichen an die Anforderungen eines Benutzers oder die Experimentanforderungen an. Konfigurationen können extern abgelegt werden. Dies erlaubt eine Neukonfiguration in kürzester Zeit.
Der Simulation liegt eine explizite Weltmodellierung zugrunde. Sämtliche Objekte werden im Weltmodell durch ihre Position und ihren Typ repräsentiert. So besteht der Roboterarm aus 13 Objekten. Jedes einzelne dieser Objekte ist passiv. Erst durch das kinematische Modell (siehe [29]) des Roboters werden sie in ihrer Raumlage verändert. Durch die Trennung von Weltmodell und Kinematik ist gewährleistet, daß unterschiedliche Typen von Robotern simulierbar sind. Dies betrifft besonders den Robotertyp PUMA 560, welcher alternativ zum 260 Verwendung findet. Entsprechend sind die kinematischen Daten dieses Systems vorbereitet worden. Als weitere Objekte existieren die

zu manipulierenden Baufixteile und die Arbeitsfläche. Die Simulation erfolgt diskret in der Zeit und kontinuierlich in der Position.
In Anlehnung an biologische Systeme soll der Agent über einen Satz funktionstüchtiger Sensoren verfügen, ähnlich wie ein Kleinkind über verläßliche Sinnesorgane verfügt. Das Kleinkind kann diese einsetzen, ohne bereits konzeptuelle Repräsentation oder sprachliche Zuordnungen der aufgenommenen Sensordaten zu besitzen. Aus der Wahl des Steuerungsansatzes ergibt sich Anforderung, daß das System in der Lage sein muß, realitätsnahe Sensoren zu simulieren. Derzeit sind folgende Sensoren realisiert:

- ein Kamerasensor, montiert auf der Roboterhand
- binäre taktile Sensoren
- telemetrische Sensoren
- Gelenkstellungssensoren

Dies entspricht einer Sensorausstattung, wie sie in aktuellen Projekten eingesetzt wird. Um das System netzwerktransparent von externen Anwendungen steuerbar zu machen, wurde eine auf dem Kommunikationstool DACS ([23]) basierende Schnittstelle definiert und implementiert [2].
Das System ist unter ausschließlicher Verwendung von Standardkomponenten entwickelt worden und damit leicht auf andere Betriebssystemplattformen zu portieren.

Visualisierung und grafische Oberfläche

Zur Darstellung der durchgeführten Handlungsabläufe mußte eine echtzeitfähige Visualisierung für die Simulation entwickelt werden. Grundlage der Visualisierung ist VOGLE, eine Bibliothek mit OpenGL kompatiblen Grafikroutinen. VOGLE ist als reine Softwarelösung unabhängig von zusätzlicher teurer Hardware.
Um eine möglichst einfache Portierung auf das angestrebte Zielsystem zu ermöglichen, wurden nur einige wenige Routinen der Bibliothek verwendet. Neben der Initialisierung und der Definition der verwendeten Farben wird als Zeichenroutine lediglich die Darstellung geschlossener, ebener Polygone eingesetzt.

[2] In Zusammenarbeit mit dem Projekt D3 des SFB 360 'Situierte künstliche Kommunikatoren'

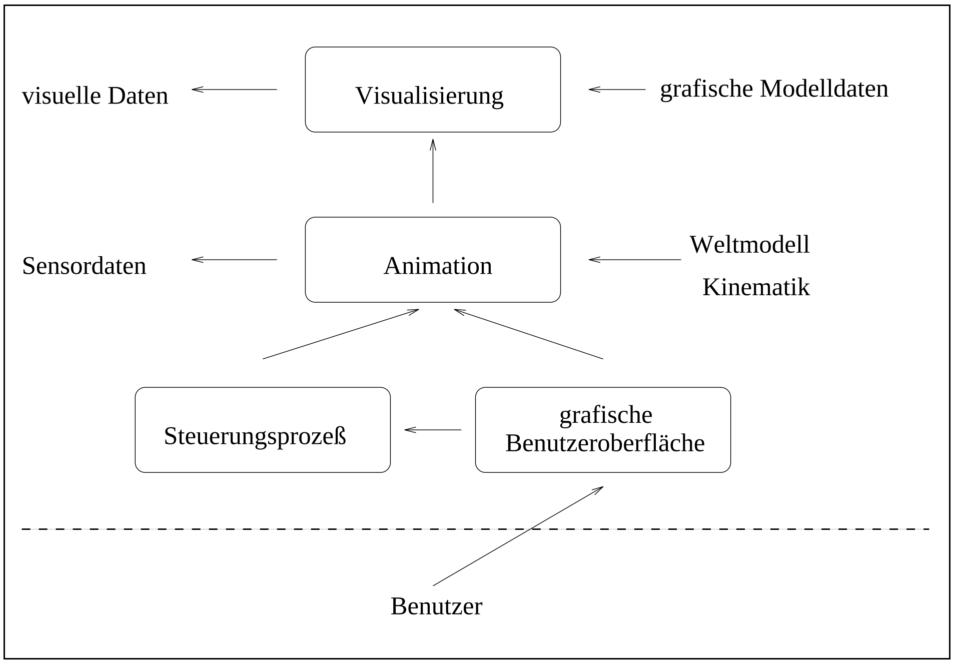

Abbildung 3.3: Datenfluß innerhalb der Simulation
Ausgehend von einer Benutzerinteraktion oder einem Steuerungsbefehl erzeugt die Animationskomponente Daten des Weltmodells und kinematische Daten um in Sensordaten (Gelenkstellung, taktile und telemetrische Sensorik). Die Animationskomponente steuert die Visualisierung welche unter Verwendung der grafischen Modelldaten die aktuelle Szene anzeigt. Durch die Visualisierung in einem zweiten Fenster werden so die Kameradaten generiert.

Die grafische Modellierung der Objekte erfolgt ausschließlich über derartige Polygone. Bei einer Portierung muß entsprechend lediglich diese Zeichenroutine vorhanden sein, um eine vergleichbare Darstellung zu gewährleisten. Innerhalb der Visualisierung werden keine komplexen Objekte aufgebaut. Nach der Berechnung der Zielposition werden die grafischen Daten aller Objekte als Polygone in den Darstellungsspeicher übertragen, der in einem zweiten Schritt dargestellt wird. Die Visualisierung und die Repräsentation der Welt sind dadurch vollständig voneinander getrennt.

Ein Nachteil dieses Ansatzes ist, daß für eine konsistente Datenhaltung zwischen Weltmodell und grafischen Daten gesorgt werden muß. Dies gilt im Besonderen für die physikalischen Eigenschaften der Objekte. Da die nichtvisuellen Sensoren ausschließlich auf dem Weltmodell arbeiten und hier unter anderem die Größe der Objekte festgelegt ist, muß dafür gesorgt sein, daß das visualisierte Objekt in seiner Ausdehnung dem repräsentierten Objekt entspricht.

Andernfalls tritt der Effekt auf, daß Sensoren Daten liefern, welche nicht mit dem visualisierten Weltzustand übereinstimmen. Dies hätte besonders Auswirkungen auf den visuellen Sensor, welcher ausschließlich über die Visualisierung gespeist wird. Sind die Abweichungen zwischen Weltmodell und visualisiertem Zustand zu groß, funktioniert das Zusammenspiel zwischen den restlichen Sensoren und der Kamera nicht mehr reibungslos. Das System 'sieht' Dinge, die nicht zu 'fühlen' sind und umgekehrt.

Die Steuerung der Simulation erfolgt auf zwei Ebenen. Zum einen können über die Interfaceroutinen Befehle direkt an die kinematische Komponente gegeben werden. Diese übergibt anschließend die berechnete Armstellung an die Simulation, welche schließlich die Visualisierung vornimmt.

Zum anderen kann die Simulation über die grafische Benutzerschnittstelle gesteuert werden. Die Steuerung erfolgt nicht exklusiv. Der Systementwickler kann also zu jeder Zeit in die aktuelle Handlung eingreifen. Dies hat zum Beispiel den Vorteil, daß unvorhersehbare Ereignisse in die Simulation eingebracht werden können. So kann getestet werden, ob der Steuerungsansatz robust auf Unsicherheiten der Umwelt reagiert.

Ziel bei der Programmierung der grafischen Oberfläche war es, dem Programmierer eine Arbeitsumgebung zur Verfügung zu stellen, welche die Entwicklung behaviorbasierter Steuerungssysteme maximal unterstützt. Die grafische Oberfläche ist intuitiv bedienbar. Jedes Menü kann zu jeder Zeit geöffnet werden, so daß Veränderungen der Konfiguration auch während des Simulationslaufes möglich sind. Durch weitgehende Konfigurationsmöglichkeiten sind auch komplexe Ausgangsszenarien einfach zu erstellen und zu modifizieren. Durch die Möglichkeit zur Abspeicherung von Systemkonfigurationen kann der jeweilige Entwickler die für seine Zwecke optimale Experimental-

umgebung erstellen und über einen längeren Zeitraum nutzen.
Neben der unmittelbaren Steuerung des Roboters ist es bei der Entwicklung behaviorbasierten Systeme wichtig, den aktuellen Zustand des Systems genau zu kennen. Durch die Überlagerung von unterschiedlichen Behaviors entstehen komplexe Handlungsabläufe. Diese gewünschte Eigenschaft hat bei der Entwicklung von Steuerungsprogrammen den Nachteil, daß selbst aus wenigen Behaviors bestehende Steuerungssysteme in ihrer Arbeitsweise nur schwer nachzuvollziehen sind. Erst durch genaue Kenntnis des Gesamtzustandes wird die Systemanalyse möglich. Der Gesamtzustands wird bestimmt durch

- den aktuellen Aktuatorstatus,
- den aktuellen Sensorstatus,
- die Aktivationsverteilung innerhalb der Behaviorhierarchie und
- das vorgegebene Handlungsziel.

Innerhalb der Simulation können lediglich Aktuator- und Sensorstatus direkt festgestellt werden. Aktivationsverteilung und Handlungsziel sind Teil des Behaviorsystems und werden an das Simulationssystem übertragen. Das Simulationssystem sorgt dann für die Visualisierung der Daten.
Der Aktuatorstatus wird zu jedem Zeitpunkt durch die Schieberegler zur direkten Positionskontrolle angezeigt. Diese folgen der Bewegung, auch dann wenn der Steuerungsbefehl durch das Steuerungssystem erfolgt. Damit ist (in Grenzen) eine Abschätzung der Handlungsdynamik des Systems möglich.
Der aktuelle Sensorstatus der taktilen und telemetrischen Sensoren wird durch ein Balkendiagramm angezeigt. Das Balkendiagramm zeigt jedoch lediglich den Zustand zu einem Zeitpunkt t. Um einen besseren Überblick über die letzten n Sensordaten zu erhalten, sind diese auch als Zeitverlaufshistogramm darstellbar. Taktile Sensoren, als binäre Sensoren, nehmen entsprechend die Werte eins oder null an; bei telemetrischen Sensoren wird der gemessene Wert skaliert angezeigt. Durch das Histogramm lassen sich kritische Zustände des System gut erkennen. Gerät das Systems z.B. in einen zyklischen Blockadezustand, so wird dieses durch ein sich wiederholendes Muster im Histogramm sichtbar.
Die Bilder, die das Handkamerasystem aufnimmt, werden kontinuierlich in einem Kontrollfenster angezeigt. Der Anwender hat so die Möglichkeit, den Blickwinkel des Systems einzunehmen. Die innerhalb des Kontrollfensters angezeigten Daten sind dieselben, die an das Steuerungssystem übertragen

werden. Dies hat zur Folge, daß bei sinkender Rasterung auch die Größe und der Detaillierungsgrad des angezeigten Bildes abnimmt.
Der Zustand des Steuerungssystems wird an das Simulationssystem übermittelt und dort durch einen Textbrowser angezeigt. Jeweils eine Zeile stellt den Zustand eines Behaviors dar, wobei Name, Behaviorebene, Vorbedingung und Aktivationswert angezeigt werden. Zur schnelleren Identifizierung wird das aktive Behavior gelb unterlegt. Da die Anzahl der Behaviors im Verlauf der Experimente stark schwanken kann, ist es möglich, einzelne Behaviors zu selektieren und damit ihre Ausgabe zu unterdrücken.
Die Visualisierung des Handlungszieles erfolgt ebenfalls über ein textuelles Fenster. In diesem wird das aktivierte Zielbehavior mit den spezifizierten Parametern ausgegeben.
Über die diversen Anzeigemodi ist der Benutzer in der Lage, sich einen Gesamtüberblick, sowohl über den aktuellen Systemstatus, wie auch über die während des Testlaufes angefallenen Daten, zu verschaffen. Für eine spätere Komplettanalyse ist es möglich, die Sequenz der eingenommenen Systemzustände in einer externen Datei abzulegen. Der so erstellte Trace liegt als Textdatei vor und ist damit einfach in weitere Analysewerkzeuge (z.B. Visualisierungsysteme, mathematische Komplexitätsanalyse etc) importierbar.

Grafische Benutzerschnittstelle

Die grafische Benutzerschnittstelle ist hierarchisch organisiert. Über das Hauptauswahlmenü werden Untermenüs aktiviert, die

- das Speichern und Laden von Konfigurationen (file)
- die Einstellung allgemeiner Optionen (options)
- die Editierung des aktuellen Weltzustandes (edit)
- die Anbindung des Kommunikationstools I-Space (ispace) [3]
- die Einstellung von speziellen Optionen (expert)
- und den Start der Simulation (start)

zulassen.
Es werden Standardelemente, wie Knöpfe, Schiebeelemente, Auswahlelemente oder textuelle Eingabemasken verwendet. Das verwendete Programmiertool ist XForms [91], eine frei verfügbare Funktionsbibliothek zur Erstellung

[3] Hier soll in Zukunft das Kommunikationstool DACS eingesetzt werden

von grafischen Benutzeroberflächen. Durch das in der Bibliothek mitenthaltene Programm `fdesign` wird die Erstellung der grafischen Oberflächen sehr vereinfacht. Das Programm erzeugt compilierfähigen C-Code, so daß mit fdesign erstellte Oberflächen unmittelbar in die Zielapplikation übernommen werden können. Das Tool unterstützt die klare Trennung von Applikation und Benutzeroberfläche, wodurch bestehende Applikationen leicht mit einer Oberfläche 'nachrüstbar' sind.
Im Rahmen der Zusammenarbeit mit dem Projekt D1 ergab sich als weiterer Vorteil, daß Programmteile, welche von unterschiedlichen Entwicklern programmiert worden sind, sehr viel einfacher zu integrieren waren. Dieser Effekt kommt durch das in XForms definierte Programm-Modell zustande, welches eine objektorientierte Modularisierung vorsieht. Routinen werden durch eine Interaktion mit der grafischen Oberfläche aktiviert. Der Kontrollfluß des Programms ist damit nicht mehr sequentiell, vielmehr reagiert das System per Interrupt auf die asynchronen Aktionen des Benutzers. Dieses Interaktionsmodell hat zur Folge, daß wesentlich unabhängigere Programm-Module entwickelt werden, welche einfacher zu einem Gesamtsystem verkoppelt werden können.

Weltmodell

Grundlage der Simulation ist das Weltmodell. Innerhalb des Weltmodells werden die Objekte und ihre Position in der realen Welt definiert. Jedes Objekt ist dabei ausgezeichnet durch seine Position, den Typ, die Farbe und ob es zum aktuellen Zeitpunkt vom Roboter gegriffen wird. Alle Objekte innerhalb des Weltmodells sind passiv. Sie werden erst durch die Animation in ihren beschreibenden Parametern verändert.
Im Weltmodell wird jedoch nicht die grafische Repräsentation des Objektes festgelegt. Durch die Trennung von Repräsentation und visuellen Daten wird es möglich, Objekte in unterschiedlicher Art und Weise zu visualisieren. Dies hat den Vorteil, daß innerhalb eines Simulationslaufs weniger wichtige Elemente ausgeblendet oder in modifizierter Qualität dargestellt werden können.
Neben den Objekten der realen Welt werden im Weltmodell auch die Sensoren repräsentiert. Sie unterliegen damit den gleichen Bedingungen, die auch für andere Objekte gelten: ihre Visualisierung ist unabhängig vom Sensortyp, die Position kann frei definiert werden.
Hierdurch wird es möglich, die Sensortopologie des Roboters frei zu definieren. Sensoren können an eine beliebigen Position am Roboter angebracht werden. Sie behalten die relative Position zu einem Referenzpunkt bei und folgen so den Bewegungen des Roboters. Die Ausrichtung im Raum bleibt

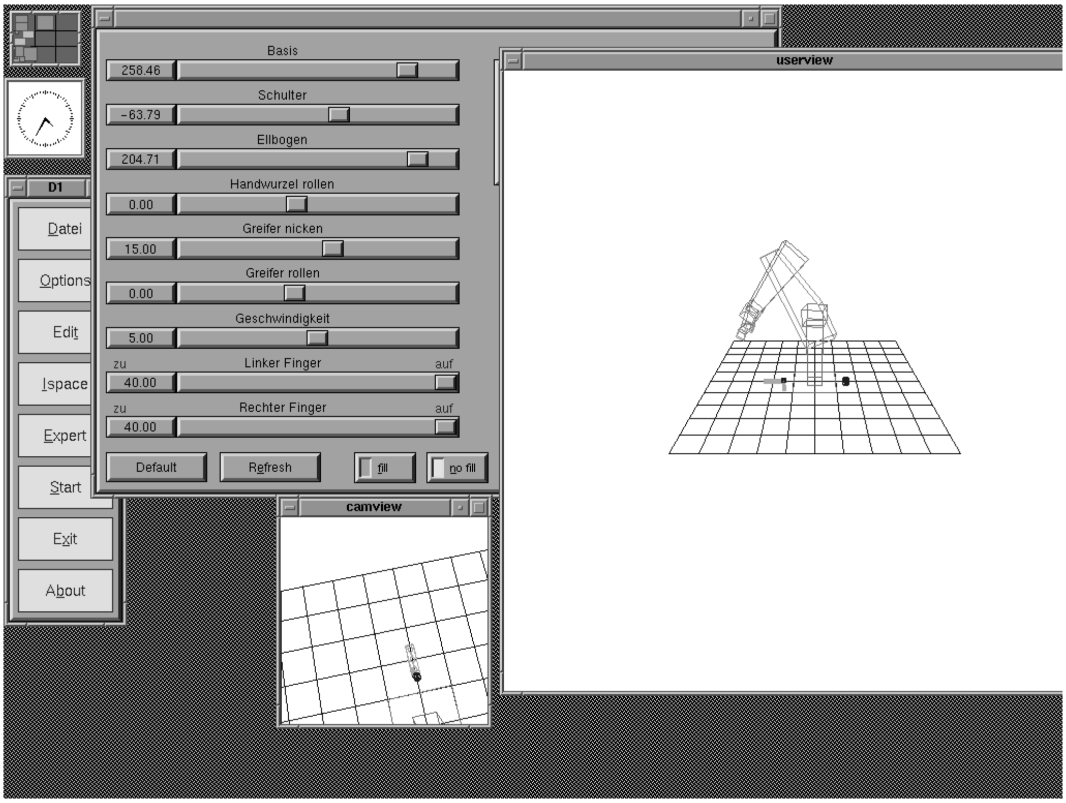

Abbildung 3.4: Grafische Benutzerschnittstelle des Simulationssystems
Die Abbildung zeigt eine frühe Version des eingesetzten Simulationssystems. Die Schiebeelemente ermöglichen die direkte Steuerung des Armes. Im Fenster `userview` wird die Gesamtszene für den Benutzer visualisiert. Das Fenster `camview` zeigt die Szene aus Sicht der Armkamera des Agenten. Im Arbeitsraum des Manipulators sind eine Reihe von Baufixteilen platziert (z.B. Schraube und Dreilochleiste).

ebenfalls erhalten.
Greifhandlungen werden innerhalb des Modells durch die Markierung des gegriffenen Objektes durchgeführt. Greifen wird insofern nicht als Simulation der physikalischen Vorgänge realisiert, sondern als Änderung der Weltrepräsentation. Das zu greifende Objekt ist als 'gegriffen' gekennzeichnet.
Damit das Objekt den Bewegungen des Roboterarmes folgt, wird der Differenzvektor zwischen dem Objektkoordinatensystem und dem Tool Center Point (TCP) des Greifers berechnet. Nach jeder Bewegung des Roboters wird über den TCP und den Differenzvektor die neue Position des Objektkoordinatensystems bestimmt. Die Positionsbestimmung erfolgt unter Berücksichtigung möglicher Rotationen des Koordinatensystems der Hand. Die Position des gegriffenen Objektes wird also als Punkt innerhalb des Handkoordinatensystems aufgefaßt.
Dies führt innerhalb der Simulation dazu, daß Objekte innerhalb der Weltrepräsentation als 'gegriffen' gekennzeichnet werden, auch wenn kein Kontakt zur Hand bzw. den Greifern vorhanden ist. Das Objekt folgt der Hand; seine Position wird durch den Differenzvektor bestimmt.
Ein sinnvoller Greifvorgang ergibt sich erst dann, wenn das Steuerungssystem, unter Berücksichtigung der Sensordaten, ein Objekt als 'gegriffen' gekennzeichnet ist. Das Steuerungssystem ist jedoch von der Weltrepräsentation komplett entkoppelt . Die Interaktion zwischen Steuerung und Simulation erfolgt ausschließlich über Sensoren und die Interfaceroutinen.
Aus Sicht des Simulationssystems ergibt sich hier die Schwierigkeit, festzustellen, welches Objekt als 'gegriffen' gekennzeichnet werden soll. Aus diesem Grund wurde eine Interfaceroutine eingeführt, die ein Objekt in Abhängigkeit vom Sensorstatus markieren kann. Die Markierung kann entweder explizit durch das Steuerungssystem initiiert werden oder automatisch dann erfolgen, wenn ein Sensor bzw. eine Menge an Sensoren einen definierten Status annimmt.
Entsteht bei der ersten Lösung noch der Eindruck, daß Steuerungssystem und Weltrepräsentation geringfügig gekoppelt sind, wird dies durch die automatische Markierung ausgeschlossen. Beim praktischen Einsatz des Systems werden extrem kleine Kontaktsensoren unmittelbar auf den Greiferbacken positioniert. Diese lösen dann bei Kontakt mit einem Objekt die Greifmarkierung aus.
Um eine Beschleunigung der Simulation zu ermöglichen, wurde bei der Implementation des Weltmodells eine Zweiteilung vorgenommen. Objekte, die kontinuierlich ihre Position ändern, werden in den aktiven Teil abgelegt. Objekte, die im wesentlichen als statische 'Hindernisse' in der Welt auftauchen, werden in dem passiven Teil gespeichert. Ein passives Objekt wird dann zum aktiven Objekt, wenn es gegriffen wird, oder wenn sich die Position der

betrachtenden virtuellen Kamera ändert. In beiden Fällen muß die visuelle Darstellung des Objektes neu errechnet werden. Da diese Berechnung besonders bei den modellierten Baufixteilen sehr aufwendig ist, wird durch die Zweiteilung des Weltmodells ein erheblicher Geschwindigkeitsgewinn erzielt.

3.1.2 Kinematische Komponente

Die kinematische Komponente wurde entwickelt, um die Simulation mit einer zusätzlichen Schnittstelle auszustatten, die Steuerungssysteme zuläßt, welche in der klassischen Robotik eingesetzt werden. Steuert man das Simulationssystems über die Komponente durch ein reaktives System, so ist eine mögliche Portierung auf reale System wesentlich einfacher.
Das Verhalten des simulierten Roboters kann durch eine Kette von Koordinatensystemen beschrieben werden. Jedes Glied des Armes besitzt ein eigenes Koordinatensystem und durch die *Denavit-Hartenberg Parameter* (DH-Parameter) können Beziehungen zwischen zwei Systemen beschrieben werden ([29]). Ausgehend von der Roboterbasis bezieht jedes Glied des Armes bis zur Hand sein Koordinatensystem auf das jeweils vorangehende. Mit den DH-Parametern wird eine homogene Transformationsmatrix aufgestellt, die zur Umrechnung von Koordinaten zwischen zwei Systemen verwendet wird und somit die Beziehung dieser beiden Systeme ausdrückt. Durch die Aneinandermultiplikation aller Matrizen des Armes ergibt sich eine Matrix, die die Beziehung der Hand bzw. des Handkoordinatensystems bezüglich der Roboterbasis bzw. des entsprechenden Koordinatensystems beschreibt. Aus ihr kann man die Position und Orientierung des Handkoordinatensystems für eine gegebene Gelenkstellung ablesen.
Für die Simulation eines Roboterarms ist also die Repräsentation der kinematischen Struktur mit DH-Parametern der Anfang. In Tabelle 3.1 sind die DH-Parameter eines PUMA 560/260-Roboters aufgeführt.
Mit den Transformationsmatrizen werden dann bei einer gegebenen Gelenkstellung die Positionen und Orientierungen der einzelnen Roboterglieder bzw. ihrer Koordinatensysteme bestimmt. Durch zusätzliche Angabe von Gelenkgeschwindigkeiten und -beschleunigungen werden Zwischenpositionen errechnet. Mit diesen Geschwindigkeiten und Beschleunigungen werden über die Vorwärtsiteration der Newton-Euler-Methode die Geschwindigkeiten und Beschleunigungen der Massenschwerpunkte der Glieder des Roboterarms bestimmt.
So wie die Beziehung von Robotergliedern mit variablen Matrizen (abhängig von der Gelenkstellung des Armes) beschrieben wird, so werden konstante Matrizen (*affixments*) zur "Befestigung" von Objekten und Sensoren bzw. Kameras verwendet. Diese Matrizen repräsentieren eine konstante Transfor-

Gelenk i	θ_i	α_i	a_i	d_i	Gelenkeinschränkung
1	0	-90	0	0	-160 bis +160
2	90	0	431.80	0	-225 bis +35
3	0	90	-20.32	149.09	-52 bis +232
4	0	-90	0	433.07	-115 bis +180
5	0	90	0	0	-100 bis +100
6	0	0	0	0	-215 bis +215

Tabelle 3.1: DH-Parameter eines PUMA 560/260-Roboters
Die DH-Parameter sind die Grundlage der kinematischen Komponente. Hierauf basierend wird die vorwärtskinematische Berechnung des Systems durchgeführt. Durch Austausch der DH-Parameter lassen sich so eine Reihe unterschiedlicher Robotersysteme simulieren.

mation zwischen Objekt- bzw. Sensorkoordinatensystem und Gelenkkoordinatensystem. Bewegt sich das Gelenk, so bewegt sich das befestigte Objekt aufgrund der Transformation entsprechend mit. Nach jedem kinematischen Schritt wird die Objektrepräsentation aktualisiert, der aktuelle Sensorstatus generiert und die veränderte Szene neu visualisiert.

Kamera

Das innerhalb der Simulation verwendete Kameramodell beruht auf einem computergrafischen Ansatz.
Hierbei wird eine Abbildungspyramide definiert, welche einen Ausschnitt der Welt umfasst. Alle innerhalb dieser Pyramide vorhandenen Objekt sind sichtbar. Mit Hilfe einer einfachen perspektivischen Transformation werden die Eckpunkte der Objekte auf eine Ebene abgebildet. Diese Abbildungsebene entspricht dabei der Retina im Auge oder dem CCD-Sensor einer elektronischen Kamera. Abbildung 3.5 zeigt die prinzipielle Arbeitsweise der Kamera.
Je nachdem wie die Transformationsmatrix definiert wird, sind unterschiedliche perspektivische Verzerrungen möglich. Bei der vorliegenden Matrix wurde eine *perspektivische* Abbildung gewählt. Diese verzerrt Winkel und Flächen, ist jedoch insofern realistisch, als weiter entfernte Objekt auch kleiner dargestellt werden.
Vor der Abbildung auf die Sichtebene werden versteckte Kanten und Flächen

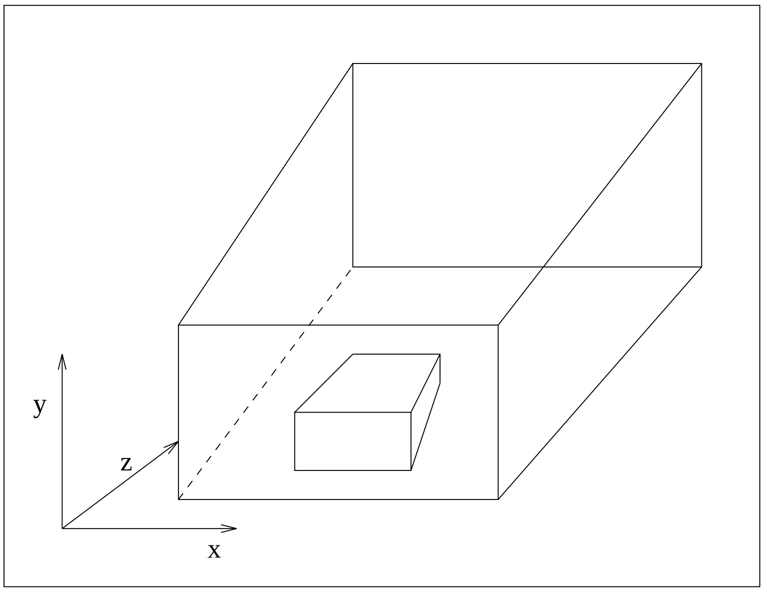

Abbildung 3.5: Funktionsweise der simulierten Kamera
Zur Simulation der Handkamera des Systems wird eine perspektivische Abbildung der simulierten Umwelt auf eine Bildfläche berechnet. Der Bildausschnitt ist dabei abhängig von der Position der Kamera, dem Horizont der Abbildungspyramide und ihrem Öffnungswinkel. Die Auflösung ergibt sich aus der Größe des verwendeten Darstellungsfensters.

entfernt. Hierdurch reduziert sich der Abbildungsaufwand erheblich.
Nach der Abbildung auf die Ebene wird das Bild aufgerastert, d.h. in diskrete Bereiche aufgeteilt. Der Farbwert eines Rasterbereiches ergibt sich als gemittelte Summe der abgebildeten Bildpunkte.
Ein so generiertes Bild wird auf dem Bildschirm dargestellt. Der Zugriff des visuellen Systems erfolgt dann über die im Grafikspeicher des Bildschirms abgelegten Daten. Das visuelles System und die Simulation sind dadurch voneinander unabhängig. Es erfolgt also *kein* Zugriff des visuellen Systems auf die internen Daten der Simulation.
Ein derartiges Verfahren läßt den Einsatz von Fehlermodellen auf der Pixelebene des generierten Bildes zu. Farbverfälschungen, Reduktion von Auflösung und Farbtiefe, Verrauschen, teilweises Löschen ganzer Bereiche des Bildes sind so ohne weiteres möglich. Somit wird eine schrittweise Annäherung an die Qualität (besser: der Fehlerhaftigkeit) von echten Kamerabildern ermöglicht ([25]).
Die vorliegende Visualisierung läßt kein Beleuchtungsmodell zu. Dadurch tritt keine Farbdifferenzierung und keine Schattenbildung auf. Um realistischere Kameradaten zu generieren, ist deshalb eine Verbesserung der Visualisierung unumgänglich. Das Modell vereinfacht die Vordergrund- Hintergrundtrennung, da Hintergrundfarben bekannt und damit einfach zu detektieren sind.
Die Visualisierung ist extrem rechenaufwendig. Hierdurch und durch die Verwendung von X11, welches in der Darstellung von Pixelbildern langsam ist, bremst die Visualisierung die Simulation. Entsprechend ist ein realistisches Echtzeitverhalten (22 Bilder pro Sekunde) im Onlinebetrieb zur Zeit nicht erreichtbar.

Berührungssensor

Der Berührungssensor misst periodisch, ob sich ein Objekt zu dem gegebenen Zeitpunkt in seinem Wahrnehmungsbereich befindet. Der Status des Sensors wird entsprechend gesetzt. Zur Berechnung des Sensorwertes wird auf die Modelldaten zurückgegriffen. Innerhalb der Simulation wird zu jedem Objekt der umschreibende Quader berechnet. Bei den taktilen Sensoren wird überprüft, ob der Messpunkt innerhalb des umschreibenden Quaders liegt.

Entfernungssensor

Die Entfernungssensoren entsprechen in ihrer Funktionsweise den taktilen. Zusätzlich wird bei den telemetrischen Sensoren die Länge des Differenzvektors berechnet und als Abstandswert zurückgeliefert.

Vorverarbeitung der sensorischen Daten

Grundidee bei der Entwicklung von Behaviors ist es, Sensoren und Aktuatoren direkt zu koppeln. Wesentlich hierbei ist die Reduzierung des Aufwandes bei der Vorverarbeitung der sensorischen Daten. Die Analyse der Sensordaten soll nur in dem Maße stattfinden, wie sie vom Behavior benötigt wird. Entsprechend fordert beispielsweise das "Ausweichbehavior" eines mobilen Systems nicht eine komplette Analyse der Struktur der Wand, die berührt wird. Es reicht die Information "Kontakt mit Gegenstand auf der rechten Seite". Daß der Kontakt rechts erfolgt, ist als Vorwissen in dem System bekannt. Es ergibt sich aus der Sensoranordnung. Das "Ausweichbehavior" ist entsprechend auf die Auswertung dieser Sensoren spezialisiert.
Innerhalb des Simulationssystems werden die erzeugten Sensordaten durch mehrere Komponenten ausgewertet. Diese aufbereiteten Daten stehen dann dem Steuerungssystem zur Verfügung.
Die aufwendigste Analyse wird beim Kamerasensor durchgeführt. Dieser liefert ein farbiges Pixelbild des durch die Handkamera sichtbaren Ausschnittes der Szene. Das Bild ist von variabler Größe und kann in unterschiedlichen Auflösungen gerastert werden. Die Standardauflösung ist mit 16 mal 16 Pixeln relativ gering. Durch die Handlungsdynamik und den beschränkten Kontext reicht sie jedoch aus, um Objekte zu lokalisieren und zu identifizieren.
Die Vorverarbeitung des Kamerabildes erfolgt in mehreren Schritten. Zunächst wird das Bild in 4 Quadranten unterteilt. Innerhalb dieser Quadranten werden die Teilflächen identifiziert, die als potentielle Objekte in Frage kommen. Durch gewichtetes Auszählen der Punkte wird ein Offsetvektor errechnet. Dieser ist ein Maß dafür, inwieweit das Bild zentriert ist und welche Korrektur in welche Richtung zur Zentrierung vorgenommen werden muß.
In einem weiteren Schritt müssen die sichtbaren Objekte identifiziert werden. Hierbei greift der Verarbeitungsmechanismus auf szenarienbezogenes Vorwissen zurück:

- Identifikation der Farbe von Objekten (grün, rot, holzfarben)
- Identifikation von Formmerkmalen von Objekten (Ausrichtung und Schwerpunkt, Länge der Hauptachsen)
- Identifikation von relativer Lage zwischen Objekten (Berührung)

Diese Merkmale reichen aus, um die innerhalb der Experimentalumgebung vorkommenden Objekte eindeutig zu identifizieren. [4]

[4] rote Schraube, grüner Würfel, roter Würfel und holzfarbene 3-Lochleiste

Zur Feststellung der Formmerkmale der Objekte werden, durch ein an den Floodfill Algorithmus ([25]) angelehntes Verfahren, innerhalb des Bildes zusammenhängende Flächen gleicher Farbe (Blobs) identifiziert. Für diese Flächen werden die Eckpunkte des begrenzenden Rechteckes errechnet. Hierzu wird, ausgehend von einem beliebigen Punkt innerhalb der Fläche, jeweils der maximal

1. obere, rechte,
2. untere, linke,
3. linke, obere
4. rechte, untere

Punkt der zusammenhängenden Fläche verwendet. Mit Hilfe dieser Eckpunkte lassen sich die Hauptachsen, ihre Länge und der Schwerpunkt der zusammenhängenden Fläche errechnen. Damit ist die Lage des Objektes innerhalb der Fläche identifizierbar.
Im letzten Schritt werden die Relationen zwischen den identifizierten Teilflächen bestimmt. Wichtig ist dabei, ob sich 2 Teilflächen berühren. Zur Verkleinerung des Berechnungsaufwandes wird zuerst festgestellt, welche Paare von Objekten innerhalb des gleichen Quadranten liegen. Anschließend wird bei den verbleibenden Objekten überprüft, ob ihre Schwerpunkte innerhalb einer vorgegebenen Distanz zueinander liegen. Bei den so bestimmten Paaren wird anschließend überprüft, ob sich ihre Begrenzungslinien berühren.
Die Auswertung der numerischen Sensoren ist vergleichsweise einfach. Taktile Sensoren liefern nur binäre Werte, telemetrische Sensoren einen analogen Wert, welche den Abstand zum nächsten feststellbaren Objekt beschreiben. Die Gelenksensoren liefern den analogen Winkelwert zurück, der Handsensor zeigt den Zustand des Zweifingergreifers als binäres Datum an.

3.1.3 Grenzen der Simulation

Folgende Vereinfachungen wurden an den Komponenten vorgenommen:

Objekte : Alle Objekte werden als Polygone modelliert. Sie lassen sich momentan nur a priori zu Verbundobjekten zusammenfügen. Die Dynamik beschränkt sich auf die Modellierung von starren homogenen Körpern.

Roboterarm : Es werden keinerlei Unsicherheiten in der Kraftübertragung berücksichtigt.

Sensoren : Auch hier werden bisher keine Fehlermodelle verwendet.

Kamera : Die Objekte werden als gefüllte Flächen dargestellt. Selbst mögliche eine Erweiterung mittels eines einfachen Shadings und die Verwendung von Texturen berücksichtigt nur unvollkommen Beleuchtungseffekte.

Berührungssensor : Modellierung als binärer punktueller Sensor.

Entfernungssensor : Das augenblickliche Modell ist eine Verallgemeinerung realer Entfernungsmesser (wie Laserscanner, Ultraschall- und Infrarotsensoren). Es gibt keinen Bezug auf die Welleneigenschaften des Signals (Reflexion u.a.) und das Laufzeitverhalten.

Auf der Systemebene ergeben sich aufgrund der Hardware- und Betriebssystemsoftwareausstattung Probleme mit dem Echtzeitverhalten des Systems. Dieses beeinträchtigt vor allem die Interaktivität des Systems.
Die meisten der oben angesprochenen Mängel lassen sich gut beheben. Besondere Schwierigkeiten ergeben sich bei der Simulation von komplexen Handbewegungen (insbesondere Schraubbewegungen). Diese machen umfangreiche Änderungen in der Modellierung notwendig bzw. sind sehr aufwendig in der Modellierung. Bei der Simulation von Sensoren ergeben sich ähnliche Probleme für die Modellierung der Kamera.
Um den Entwicklungsvorgang weiter zu verbessern, sollte eine Kopplung zu realen Robotersystemen möglich sein. Ein derartiger Ansatz findet sich bereits teilweise in einigen anderen Robotersimulationen (DiSCUS [57], ROTEX [13], siehe Kapitel 2.2). Der Entwickler kann dann die Vorteile der Simulationsumgebung (einfachere Handhabung, höhere Interaktivität, ungefährlicher und kostengünstiger) nutzen und gleichzeitig mit den realistischen Sensor- und Aktuatordaten arbeiten. Das Simulationssystem könnte so zur interaktiven Aufzeichnung und Analyse von Situationsdaten innerhalb des Szenarios genutzt werden. Damit sind innerhalb der Simulation realistische, pseudozufällige Testläufe möglich.
Um die Analyse zu verbessern, sollten sowohl Online-, wie auch Offline-Visualisierung der Situationsdaten möglich sein. Dazu sollte geprüft werden, inwieweit bereits bestehende Werkzeuge integriert werden können.

3.2 Die Behaviorkomponente

Nachdem im vorherigen Teil mit dem Simulationssystem die Basis für die (simulierte) Form des Agenten und die Funktion seiner Sensorik gelegt worden ist, wird auf den folgenden Seiten mit der Beschreibung der Entwicklung der Behaviorkomponente das Fundament für seine Handlungs- und Lernfähigkeit gelegt. Die Behaviorkomponente hat die Aufgabe die Simulation zu kontrollieren, die vom Simulationssystem generierten Sensordaten auszuwerten und durch korrektes Scheduling die Befehle der reaktiven Behaviorhierarchie abzuarbeiten.

3.2.1 Der Agent

Die definierende Eigenschaft eines Agenten ist seine Ausstattung mit Sensoren und Aktuatoren. Sie bestimmt, zu welchen Aktionen der Agent prinzipiell fähig ist, und über welchen Wahrnehmungsraum er verfügt.

Bei Lebewesen sind unterschiedliche Grundausstattungen mit Sensorik und Aktuatorik beobachtbar. Eine stark vereinfachte Regel ist, daß je niedriger die Lernfähigkeit des Lebewesens, desto mehr ist es auf eine unmittelbar zur Verfügung stehende funktionstüchtige Ausstattung mit Sensorik und Aktuatorik angewiesen. So ist eine Vielzahl von Säugetieren bereits kurz nach der Geburt in der Lage, zu laufen. Die Koordination der Beine, die Abfolge der Schritte, die Geschwindigkeit, der richtige Einsatz der Muskelgruppen, die korrekte Interpretation bzw. Reaktion auf sensorisches Feedback aus den Nerven steht nach wenigen Minuten zur Verfügung. Lernfähigkeit scheint hier nur eine untergeordnete Rolle zu spielen, und wenn, dann nur insofern, als das vorgegebene Muster "laufen" durch die ersten Erfahrungen abgerundet wird.

Eine wichtige Voraussetzung für das Überleben von Lebewesen in einer ihnen unbekannten, unsicheren Umwelt, ist die adäquate Reaktion auf sensorische Stimuli. Dies gilt sowohl für sensomotorische Eindrücke, als auch für die Wahrnehmung der externen Welt. Die Mechanismen, die diese Interpretation leisten, sind erst in einfachen Ansätzen erforscht. Man kann jedoch davon ausgehen, daß eine Analyse der sensorischen Daten, aufgrund derer ein explizites Modell der Umwelt aufgebaut wird, welches anschließend analysiert wird, um ausgehend von dem Ergebnis der Analyse eine Aktionsfolge zu bestimmen, welche in einem letzten Schritt ausgeführt wird, in den meisten Fällen fehlschlagen würde. Andererseits ist das Modell, nach dem eine nichtreflektive unmittelbare Reaktion auf einen sensorischen Reiz als ausschließliche Grundlage von Handlung angenommen wird, ebenfalls unrealistisch.

3.2.2 Die grundlegende Behaviorhierarchie

Im Folgenden sollen die in der Testphase verwendeten Behaviorhierarchien dargestellt werden. Als grundlegendes Verhaltensprogramm des Agenten legen sie fest, zu welchen Fähigkeiten der Agent überhaupt in der Lage ist. Weiterhin steuern sie, ob der Agent explorativ oder passiv, vorsichtig oder "draufgängerisch", schnell oder langsam ist. Komplexe Verhaltensweisen bauen immer auf den Basisverhaltensweisen auf. Es werden also keine neuen Grundmuster erworben, sondern ausschließlich Rekombinationen vorhandener Handlungen erlernt.
Ziel war die Entwicklung eines Steuerungsprogramms, welches den Agenten in die Lage versetzt, jeden Punkt des durch ihn erreichbaren Raumes anzusteuern. Ferner sollte es dem Agenten möglich sein, Objekte seiner Umwelt visuell zu detektieren und auf sie zu reagieren.
Die Kriterien für die Auswahl der Basisbehaviors und ihrer hierarchischen Organisation sind ausgerichtet an dieser Zielvorgabe. Durch die fehlende Methodologie für die Konstruktion von Behaviorsystemen orientierte sich die Auswahl an den mit den Experimentalsystemen Cyrano und SAMSA gewonnenen Erfahrungen. Es stellte sich heraus, daß die Übertragung der Ergebnisse dieser mobilen Systeme auf die Anforderungen des Roboterarmes nicht ohne weiteres möglich ist. Entsprechend wurde die von Schnepf/Asteroth ([2]) und von Malcolm ([54]) entwickelten Behaviorsysteme für die reaktive Steuerung eines Roboterarmes in die Planung des Behaviorsystems einbezogen.
Hierbei war besonders die von Schnepf entwickelte Behaviorhierarchie beispielhaft. Das System steuert einen Roboterarm derart, daß er in der Lage ist, einen rollenden hellen Ball auf dunklem Grund zu detektieren, diesem zu folgen und ihn schließlich zu greifen. Verliert das System den Ball aus dem Blickfeld, geht der Roboterarm in eine Grundstellung über und hält an.

3.2.3 Basisbehaviors

Die Basissteuerung des Agenten übernehmen 7 grundlegende Behaviors, welche in zwei Ebenen angeordnet sind. Die Behaviors der ersten Ebene vollziehen die unmittelbare Motorsteuerung des Agenten. Über Parameter, welche durch höherer Behaviors gesetzt werden, ist die Schrittgröße der Bewegungen des Agenten beliebig einstellbar. Das Behavior *retraction* sorgt dafür, daß der Agent keinen ungewollten Kontakt mit Objekten der Umwelt hat. Durch die tiefe hierarchische Stellung erhalten die Behaviors der ersten Ebene eine hohe Priorität innerhalb des Aktionsselektionsverfahrens.
Die Behaviors der zweiten Ebene reagieren auf visuellen Input. Sie ermöglichen

visuell geleitete, objektbezogene Handlungen. Die Motorsteuerung erfolgt auf dieser Ebene nicht mehr direkt, sondern ergibt sich durch die Aktivation der entsprechenden Behaviors (turn right, turn left, hand down, hand up) der ersten Ebene.
Ebene 1:

- *retraction*: (mit vordefiniertem Offset) Ausweichen bei Kontakt
- *turn right*: der Arm dreht sich nach rechts
- *turn left*: der Arm dreht sich nach links
- *hand down*: die Hand wird nach unten bewegt
- *hand up*: die Hand wird nach oben bewegt

Ebene 2:

- *center area*: zentriere die gefüllte Fläche im Bild
- *maximize filled area*: maximiere die gefüllte Fläche im Bild

Diese sieben Behaviors versetzen den Agenten in die Lage, innerhalb seiner Umwelt jeden Punkt zu erreichen und visuell zu erfassen. Je nachdem welche Stellung und welche Gewichtung das einzelne Behavior innerhalb der Behaviorhierarchie hat, ergibt sich ein unterschiedliches Gesamtverhalten.
Die Aktivität eines Behaviorsystems wird weitgehend durch externe Stimuli bestimmt. Dies können sowohl sensorisch perzepierte wie auch direkt in den Agenten eingebrachte Informationen sein (Anweisungen). Bleiben die Stimuli aus, so kann das System entweder passiv oder aktiv reagieren. Als passives System vermeidet es die Gefahr, sich selbst zu zerstören. Das System kann exakt über die durchzuführenden Schritte instruiert werden. Damit ist es möglich, das System schrittweise Handlungssequenzen durchführen zu lassen. Diese können so erlernt und reproduziert werden.
Als Nachteil ergibt sich, daß das System nicht in der Lage ist, seine Umwelt aktiv zu erkunden. Die sich dadurch bietenden Möglichkeiten bleiben also verschlossen, es sei denn durch externe Stimuli wird das System in vergleichbare Situationen gebracht. Gerade die aktive selbständige Exploration ist jedoch für ein lernfähiges, handelndes System von Wichtigkeit. Ein "neugieriges" System kann dabei in einer dynamischen Welt auf unbekannte Situationen treffen. Ist es in der Lage, hierauf adäquat zu reagieren, so kann dies zu einer Verbesserung des Gesamtverhaltens führen. Im Rahmen der Experimente wurden sowohl passive, wie auch aktive Systeme untersucht.

Die folgende Liste zeigt die Kodierung der Basisbehavior in einem an der Definition von LISP-Funktionen angelehnten Pseudocode. Jedes der Behaviors wird definiert durch einen Namen, eine (möglicherweise leere) Parameterliste und den Körper des Behaviors.
Innerhalb des Körpers erfolgt der Zugriff auf Variablen welche den Status von Sensoren repräsentieren (z.B. LEFT_BUMPER). Durch die Funktion `(ACTIVATE)` wird Aktivation an weitere Behaviors abgegeben. Dies kann im nächsten Schedulingschritt zu deren Aktivierung führen. Zur Motorsteuerung werden die Routinen `(increase)` und `(decrease)` aufgerufen, welche den Öffnungswinkel eines Gelenkes erhöhen oder vermindern.

```
(RETRACT (n)                    ; Offset n
        (COND
           (LEFT_BUMPER  (ACTIVATE (LIST 'TURN_RIGHT n) 100) )
           (RIGHT_BUMPER (ACTIVATE (LIST 'TURN_LEFT  n) 100) )
        )
) ; RETRACT

(TURN_RIGHT (n)
        (COND
           ((AND
                (BASE_ANGLE < MAX)
                (BASE_ANGLE > MIN)
            )
            (increase (BASE_ANGLE, n))
           )
        )
) ; TURN_RIGHT

(TURN_LEFT (n)
        (COND
           ((AND
                (BASE_ANGLE < MAX)
                (BASE_ANGLE > MIN)
            )
            (decrease (BASE_ANGLE, n))
           )
        )
) ; TURN_RIGHT

(HAND_DOWN (n)
```

```
        (COND
           ((CHECK_ANGLE_BOUNDARIES)
                (decrease (FIRST_ANGLE, n))
                (decrease (SECOND_ANGLE, n))
                (decrease (THIRD_ANGLE, n))
           )
        )
) ; TURN_RIGHT

(HAND_UP (n)
        (COND
           ((CHECK_ANGLE_BOUNDARIES)
                (increase (FIRST_ANGLE, n))
                (increase (SECOND_ANGLE, n))
                (increase (THIRD_ANGLE, n))
           )
        )
) ; TURN_RIGHT
```

3.2.4 Implementation

Bei der Implementation des Behaviorsystems zeigte sich, daß aktuelle Realisierungen von behaviorbasierten Systemen nur schwer auf das angestrebte integrierte Architekturkonzept übertragbar sind. Die meisten Systeme wurden speziell für die Anwendung entwickelt. Weiterhin waren zu Beginn der Arbeit praktisch keine Entwicklungssysteme für behaviorbasierte Systeme vorhanden. Das einzige zur Verfügung stehende System GAPPS/REX ([34] erwies sich als ungeeignet.
Ähnlich wie bei der Arbitration Macrology und MacNet (siehe [14]) handelt es sich bei GAPPS/REX um einen zweistufiges Ansatz, der zum einen aus der Schaltkreisbeschreibungs-Sprache REX und zum anderen aus der auf diese aufgesetzte GAPPS-Sprache (*Goals as a Parallel Program Specification*) besteht. Gapps erzeugt aus der Vorgabe des Top-Level Zieles des Agenten und einer Menge von Zielreduktionsregeln ein paralleles Programm zur zielgerichteten Steuerung des Agenten.
Trotz der flexiblen Art der Zielbeschreibung unterstützt GAPPS/REX den Entwickler nicht im Designprozeß der Behaviors. Es ist möglich einzelne Behaviorebenen zu definieren, die Interaktion von unterschiedlichen Hierarchieebenen ist jedoch nicht spezifizierbar. Da jedoch gerade diese für die Realisierung des Lernverfahrens von Bedeutung ist, wurde von der Nutzung von GAPPS/REX abgesehen.

Weitere Werkzeuge zur Entwicklung behaviorbasierter Steuerungssystem sind die Behaviorlanguage von Brooks ([9]) und PDL von Steels ([72]). Bei beiden handelt es sich um Spezifikationssprachen, mit welchen Behaviors und deren Vernetzung beschrieben werden können. Der Ansatz von Brooks ist als Makrobibliothek für LISP programmiert, Steels PDL wird per Crosscompiler nach C und dann per Standard C Compiler in ausführbaren Code des Zielprozessors übersetzt. Beide sind unter starker Berücksichtigung der von Brooks bzw. Steels untersuchten Forschungsziele entwickelt worden. Behaviorsysteme, welche mit Hilfe dieser Tools erstellt sind, sind nur schwer anzukoppeln an höhere Systeme. Beide Systeme sehen kein Lernen in der hier beschriebenen Art vor. Da jedoch Lernverfahren und Aktionsselektionsverfahren eng verwoben sind, sind beide Spezifikationssprachen nicht einsetzbar.
Als Konsequenz hieraus erschien es sinnvoller, eine eigene Variante eines Behaviorschedulers zu entwickeln. Der Scheduler kann an die konkreten Bedürfnisse optimal angepaßt werden:

- Auswahl eines kontrollierenden Behaviors (Winner-takes-all).
- hierarchische Anordnung der Behaviors.
- Behavior als identifizierbare Einheit.
- dynamisches Hinzufügen von neuen Behaviors zur Laufzeit.
- "Verteilung" der Distributionsnachricht über die Behaviorhierarchie.
- Auswahl des kontrollierenden Behaviors durch Überlagerung von sensor- und plangetriebenen Kriterien.
- Ankoppelung an das entwickeltes Simulationssystem.

Diese Bedingungen folgen weitestgehend der Spezifikation eines Behaviorsystems nach Brooks. Eine Erweiterung findet durch die Möglichkeit des dynamischen Einfügens von Behaviors zur Laufzeit, durch das Konzept der Verteilung von Distributionsnachrichten und durch ein, gegenüber der Subsumptionsarchitektur, verändertes Konzept der Überlagerung von sensor- und plangetriebener Auswahlkriterien statt. Die ersten beiden Erweiterungen sind notwendig, um das vorgeschlagene Lernverfahren in die Handlungsauswahl zu integrieren.
Dem Konzept der temporären Inhibition von Behaviors zur Überlagerung von sensor- und plangetriebener Kriterien der Aktionsauswahl wird bei diesem Ansatz die striktere *reaktive Zone* gegenübergestellt. Die reaktive Zone eines Behaviors ist die Menge der Behaviors, die reaktiv kompatibel zu diesem

sind. Zwei Behaviors sind reaktiv kompatibel, wenn sie zum gleichen Zeitpunkt vom Behaviorsystem gescheduled werden können, also zueinander in Konkurrenz treten können. Durch entsprechendes Setzen der reaktiven Zone wird es für ein Behavior möglich, konkurrierende Behaviors für einen Schedulingdurchlauf zu blocken. Durch Schnittmengenbildung wird bei erlernten Behaviors erreicht, daß diese Kompatibilitätsbedingungen der zugrundeliegenden Behaviors übernehmen.
Auch bei der Wahl der Programmiersprache LISP für die Programmierung des Behaviorsystems wurde Brooks gefolgt. Die Programm/Daten Äquivalenz dieser Programmiersprache erlaubt die einfache Implementation eines Schedulers in LISP, der auf in LISP kodierten Datenstrukturen arbeitet. Sämtliche Kontrollstrukturen der Programmiersprache können zur Kodierung der Behaviors verwendet werden. Ihre Interpretation erfolgt dann durch explizites Anstoßen des LISP-Interpreters mittels der `(EVAL expr)` Funktion. Da LISP eine Vielzahl von Funktionen für Operationen auf komplexen Datenstrukturen zur Verfügung stellt, ist auch die dynamische Verwaltung der Behaviorhierarchie einfacher als in einer prozeduralen Programmiersprache zu verwirklichen.

3.2.5 Das Behaviorsystem

Abbildung 3.6 zeigt eine Übersicht der implementierten Komponenten und ihrer Interaktion. Das Gesamtsystem ist auf drei programmiersprachlichen Ebenen entwickelt worden:

1. C-Ebene, Implementation zeitkritischer Komponenten (Simulation, externe Schnittstellen)

2. LISP-Ebene, Implementation des Behaviorschedulers (Lernverfahren und Aktionsselektion)

3. Behaviorebene, Behaviors als LISP-Expression, welche durch den Scheduler ausgewertet werden

Auf der C-Ebene ist das Simulationssystem und der LISP-Interpreter realisiert. Da der verwendete LISP-Interpreter (XLisp 2.1g [1]) im Quellcode vorliegt, ist es möglich, neue generische LISP-Funktionen in C zu implementieren. Diese stellen die Schnittstelle von LISP zu C dar. Eine derartige in den LISP-Interpreter integrierte Schnittstelle bewirkt eine sehr enge Koppelung von Simulation und Steuerungssystem. Beide laufen innerhalb desselben Prozesses. Der C Prozeß des Simulationssystems ist in den LISP-Interpreter eingebettet. Innerhalb von LISP muß dann dafür gesorgt werden, daß das

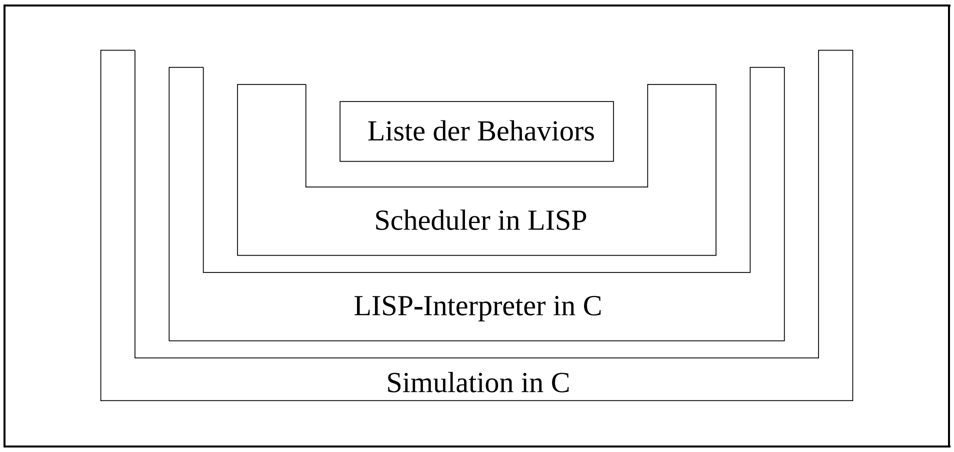

Abbildung 3.6: Ebenen der Implementation

Bei der Umsetzung der Architektur in ein Softwaresystem wurde versucht, eine der Komplexitätsebene adäquate Modularisierung vorzunehmen. Die Simulation ist aus Geschwindigkeitsanforderungen heraus in C programmiert. Darin eingebettet ist ein LISP-Interpreter, welche mittels neu definierter Primitivfunktionen mit der Simulation kommunizieren kann. Der Interpreter/Scheduler der Behaviorsprache ist als LISP-Programm realisiert, was durch die Daten/Programmäquivalenz von LISP verhältnismäßig einfach ist. Die Behaviorhierarchie schließlich wird durch eine speziell definierte Programmiersprache beschrieben, wobei der prozedurale Anteil direkt durch den LISP-Interpreter evaluiert wird.

Simulationssystem initialisiert und entsprechend gescheduled wird. Hierzu sind Funktionen implementiert worden, die es erlauben, das Update des Simulationssystems durch einen einzigen Funktionsaufruf durchzuführen.
Folgende Interfaceroutinen sind implementiert worden:

- (INIT-SIMULATION), initialisiert und startet die Simulation
- (UPDATE-SIMULATION n), führt n Scheduleschritte der Simulation durch
- (SET-DEGREE-VECTOR n1 n2 n3 n4 n5), setzt den Gelenkvektor
- (OPEN-HAND), öffnet die Hand
- (CLOSE-HAND), schließt die Hand
- (SENSOR-STATUS), liefert Liste mit aktuellem Sensorzustand

(INIT-SIMULATION) initialisiert und startet die Simulation. Das Hauptfenster der grafischen Oberfläche wird geöffnet. Maus- und Tastaturinteraktion sind möglich, werden jedoch erst im nächsten Schedulingschritt abgearbeitet.
(UPDATE-SIMULATION n) führt n Schedulingschritte der Simulation durch. Der Parameter n sorgt für eine problemloses Scheduling von asynchronen Inputevents die im Eventbuffer abgelegt worden sind, aber nicht gescheduled werden, solange das Behaviorsystem arbeitet. Dieses Problem tritt insbesondere bei großen Behaviorhierarchien auf, wo das Aktionsselektionsverfahren längere Zeit braucht, um das zu aktivierende Behavior zu bestimmen.
(UPDATE-SIMULATION) aktualisiert die Position der Objekte im Weltmodell der Simulation, führt den nächsten Schritt der kinematischen Komponente aus, berechnet die neuesten Sensorwerte und führt die komplette Visualisierung durch. Außerdem werden die Mausevents abgearbeitet. Dadurch kann zu jeder Zeit über die grafische Oberfläche in das Simulationsgeschehen eingegriffen werden.
(SET-DEGREE-VECTOR n1 n2 n3 n4 n5) setzt den Zielzustand auf die in den Parametern n_i angegebene Gelenkstellung. Mit jedem Schedulingschritt wird diese Position entsprechend der Geschwindigkeitseinstellung angenähert. Die Position wird also nicht sprunghaft angenommen; die Simulation sorgt für eine fließende Bewegung unter Berücksichtigung der Kinematik und mit ständigem Sensorupdate.
(OPEN-HAND) bzw. (CLOSE-HAND) setzt den Zielzustand des Zweifingergreifers auf maximal geöffnet bzw. voll geschlossen. Auch hier wird, wie bei (SET-DEGREE-VECTOR), der Zielzustand nicht sprunghaft eingenommen, sondern die Geschwindigkeitseinstellung einbezogen und mit jedem

Schedulingschritt ein Zwischenschritt berechnet. Nach jedem Schritt wird ein Sensorupdate durchgeführt.
(SENSOR-STATUS) liefert die Liste mit dem aktuellen Sensorstatus. Der aktuelle Sensorstatuts ist der beim letzten (UPDATE-SIMULATION) berechnete Zustand. SENSOR-STATUS ist eine reine LISP-Funktion, welche auf der Einzelabfrage der internen Sensoren aufbaut. Die Einzelabfrage erfolgt über eine Reihe von primitiveren Interfacefunktionen (siehe auch Anhang: Übersicht der Interfacefunktionen). Hierdurch ist die Reihenfolge der Statusliste frei festlegbar. Binäre Sensoren liefern dabei einen booleschen Wert, skalare Sensoren einen normierten Integerwert.
Auf der LISP-Ebene ist der Behaviorscheduler implementiert. LISP bietet für die Realisierung von quasiparallelen Prozessen eine Reihe von Vorteilen. Durch die Äquivalenz von Daten und Programmen und die Möglichkeit des expliziten Aufrufs der Evaluierung in den eigenen Programmen können die Behaviors in einfacher Weise kodiert werden. Sowohl der Header, wie auch der Body eines Behaviors sind LISP-Expressions. Die Behaviors sind innerhalb einer Liste gespeichert. Der Scheduler durchläuft diese Liste und evaluiert die LISP-Expressions. Da innerhalb der Lernphase ständig neue Behaviors generiert werden, muß das Behaviorprogramm zur Laufzeit modifizierbar sein. Derartige Modifikationen sind in LISP einfach zu realisieren, da bereits ein umfangreiches Repertoire an Funktionen zur Manipulation von Listen vorhanden ist.
Der Behaviorscheduler hat zwei wesentliche Aufgaben

1. Auswahl des kontrollierenden Behaviors

2. Ausbreitung der Distributionsnachrichten über die Behaviorhierarchie

Die Auswahl des zu aktivierenden Behaviors erfolgt über ein 2-stufiges Verfahren. Im ersten Schritt wird festgestellt, welche Behaviors innerhalb der reaktiven Zone liegen. Die reaktive Zone legt schichtenübergreifend fest, welche Behaviors aktiv werden können. Es wird eine Auswahl in vertikaler Richtung vorgenommen, beginnend bei den höheren Behaviors. Durch diese von den komplexeren Verhaltensweisen bestimmte Selektion wird festgelegt, welche Verhaltensweisen in der aktuellen Situation zugelassen sind. Die Auswahl ist "plangetrieben"; höhere Behaviors haben eine größere Handlungskompetenz. Sie definieren längere Sequenzen von Aktionen. Die Sequenz soll einen definierten Zielzustand erreichen und zwar mit einer größeren Unabhängigkeit von der aktuellen Situation.
Im nächsten Schritt wird aus der so bestimmten Menge der zulässigen Behaviors die Teilmenge der Behaviors ausgewählt, deren Vorbedingung zu diesem Zeitpunkt erfüllt ist. Nur diese Behaviors sind situationsadäquat. Die

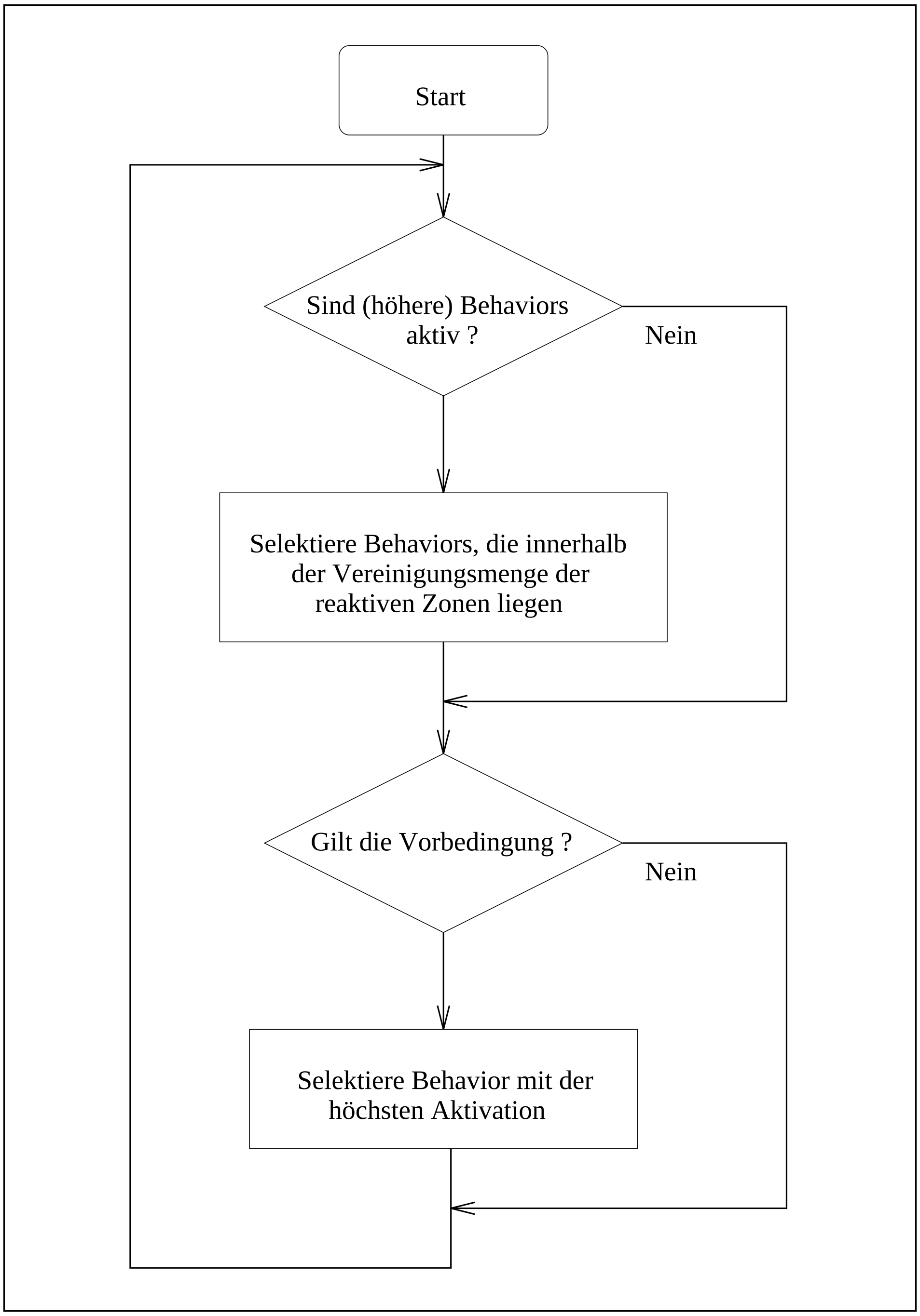

Abbildung 3.7: Ablauf des Behaviorscheduling
Die Entscheidung welches Behavior aktiv wird, basiert sowohl auf sensorischer Information, welche in der `Vorbedingung` abgeprüft wird und auf durch die Behaviorhierarchie vorgegebenen Handlungsdekompositionen, welche in den reaktiven Zonen kodiert ist. Durch Vorselektion in vertikaler Richtung (reaktive Zone) wird die Menge der Behaviors stark eingeschränkt, was zu einer erhöhten Effizienz des Systems führt.

Auswahl erfolgt jetzt "situationsgetrieben". Grundannahme ist, daß die Behaviors situationsangepaßt entwickelt worden sind und so zu einer sinnvollen Aktionssequenz führen. Durch dieses Prinzip erfolgt eine Sequenzialisierung der Aktionen über den Weltzustand; Handlungsfolgen emergieren. Dies führt beispielsweise dazu, daß unnötige Handlungen, welche ein nicht reaktiver Planer, der von einem gegebenen Startzustand aus eine Folge von Handlungen berechnet, nicht durchgeführt werden, wenn die Situation sie nicht erfordert. Das heißt, durch die reaktive Programmierung nutzt das System umweltgegebene Vorteile aus.
In einem letzten Schritt wird aus der Menge der verbleibenden Behaviors das mit der maximalen Aktivierung ausgewählt. Das Verfahren verfährt nach der Methode "winner takes all".
Nachdem das kontrollierende Behavior bestimmt ist, wird der Body evaluiert. Die im Body spezifizierten Handlungen sind entweder

- atomare Basisaktionen
- aus atomaren Basisaktion konstruierte Folgen
- Aktivation von Subbehaviors

Atomare Basisaktionen und Folgen von atomaren Basisaktionen werden durch die LISP/C Interfaceroutinen beschrieben. Als kleinste Steuerungseinheit sind sie unteilbar. Es wird jedoch nicht davon ausgegangen, daß die Basisaktionen in jedem Fall erfolgreich sind. Da nach jedem Schedulingschritt der Simulation (UPDATE-SIMULATION) der aktuelle Sensorstatus geliefert wird und die Aktionsselektion im Behaviorscheduler für das nächste zu aktivierende Behavior auf diesem Zustand aufsetzt, wird dann implizit geprüft, ob die vorangegangene Aktion erfolgreich war.
Durch die Aktivation eines Subbehaviors können Folgen von Behaviors definiert werden. Die Aktivation ist jedoch kein "Prozeduraufruf". Sie entspricht eher einer Empfehlung, welche Aktion als nächste durchzuführen ist. Der Behaviorscheduler kann entsprechend der oben definierten Auswahlkriterien eine anderes Behavior aktivieren.
Der Scheduler arbeitet auf der Behaviorhierarchie. Diese dritte Ebene ist das eigentliche Steuerungsprogramm des Agenten. Die Behaviors werden als LISP-Expressions innerhalb einer strukturierten Liste verwaltet. Abbildung 3.2 gibt die Struktur dieser Liste wieder.
Ein Behaviorprogramm ist dementsprechend eine Liste, welche eine oder mehrere Sublisten enthält. Jede Subliste beschreibt eine Behaviorebene. Jede Behaviorebene kann ein oder mehrere Behaviors enthalten. Jedes Behavior

programm	$\longrightarrow$	'(' listenelemente ')'
listenelemente	$\longrightarrow$	listelemente ebene
listenelemente	$\longrightarrow$	ebene
ebene	$\longrightarrow$	'(' ebenenelementen ')'
ebenenelemente	$\longrightarrow$	ebenenelemente behavior
ebenenelemente	$\longrightarrow$	behavior
behavior	$\longrightarrow$	'(' header body tliste ')'
header	$\longrightarrow$	'(' NAME AK AB rz ')'
rz	$\longrightarrow$	'(' namen ')'
namen	$\longrightarrow$	namen NAME
namen	$\longrightarrow$	NAME
tlist	$\longrightarrow$	'(' telemente ')'
telemente	$\longrightarrow$	telemente telement
telemente	$\longrightarrow$	telement
telement	$\longrightarrow$	'(' NAME TV ')'
body	$\longrightarrow$	LISPEXPRESSION

Tabelle 3.2: Aufbau der Behavior verwaltenden Liste

Die Syntax der eingesetzten Sprache zur Beschreibung der Behaviors bildet die Hierarchie auf eine geschachtelte Liste ab. Jede Ebene der Hierarchie entspricht einer Subliste, welche wiederum eine Liste von Behaviors enthält. Ein Behavior wird beschrieben durch header, body und tliste. Der body eines behaviors wird schließlich auf eine gültige LISP-Expression abgebildet, welche direkt evaluiert werden. Hierdurch steht dem System der volle Sprachumfang von LISP unmittelbar zur Verfügung.

ist wiederum eine Liste, welche aus dem Kopf und dem Rumpf des Behaviors besteht. Der Kopf (wiederum eine Liste) enthält

- NAME: den symbolischen Namen des Behaviors
- AK: die aktuelle Aktivation
- AB: die Aktivationsbedingung, eine LISP-Expression, welche den aktuellen Sensorstatus überprüft
- rz: die reaktive Zone, eine Liste von Behaviors, welche in der gleichen reaktiven Zone liegen.

Der Rumpf des Behaviors ist eine LISP-Expression, welche von LISP evaluiert werden kann. Die durch diese Spezifikation beschriebene Datenstruktur entspricht einem zweidimensionalen Array, welches jedoch nicht vollständig besetzt sein muß. Jede Ebene der Behaviorhierarchie entspricht einer Zeile, jedes Behavior innerhalb einer Ebene einem Spalteneintrag. Die Position der Behaviorebene innerhalb der Liste bestimmt ihr Abstraktionsniveau: je höher die Position, desto niedriger die Stellung innerhalb der Behaviorhierarchie. Durch diese Ordnung der Behaviorebenen wird für die Abarbeitung in Richtung absteigender Handlungskompetenz durch den Scheduler gesorgt.

3.3 Das Lernverfahren

Auf den folgenden Seiten soll der Mechanismus beschrieben werden, der die Fähigkeiten des reaktiven Agenten so erweitert, daß das System in der Lage ist Handlungsketten zu erlernen und in die Behaviorhierarchie einzubauen. Lernfähigkeit bedeutet hier *autonome Neuprogrammierung* des Steuerungssystem durch den Agenten selbst. Nur so ist ein Agent in der Lage, bei partiellem Ausfall vom Komponenten weiterhin robust zu agieren. Eine derartige Eigenprogrammierung bedeutet weiterhin, daß der Agent selbständig das für eine Umwelt optimale Programm erstellt und so seine Überlebenschancen maximiert. Die Anpassung an die sich dynamisch, kontinuierlich ändernde Umwelt muß ebenfalls dynamisch und kontinuierlich sein. Der Agent muß also zu jedem Zeitpunkt und in jeder Situation Lernbereitschaft zeigen. Eine Teilung zwischen *exploration/exploitation*-Phasen, wie sie in vielen Ansätzen des maschinellen Lernens vertreten wird, ist nicht haltbar.
Bei den meisten Architekturansätzen zur Realisierung eines Agenten wird (echte) Lernfähigkeit ausgeklammert. Zwar können die Systeme unerwartete Situationen erkennen und adäquat auf diese reagieren. Die Systeme sind damit in begrenztem Maße adaptiv. Bei "korrekter" Entwicklung der reaktiven Steuerung ergibt sich, daß die Systeme günstige Situation ausnutzen können. Ist es jedoch erforderlich bei einer komplexen Handlung einen ungünstigen Teilschritt durchzuführen, so sind die Systeme dazu nicht in der Lage.
Definiert man einen Agenten entsprechend den Prinzipien des behaviorbasierten Ansatzes, so muß er über *spezifisches, vordefiniertes Verhalten* verfügen. Das spezifisches Verhalten wird durch eine Menge von Aktions/Kompetenz Modulen des reaktiven Systems realisiert. Weiterhin stehen dem Agenten Daten der Sensoren zur Verfügung. Diese stellen den Status der externen und der internen Umgebung fest. Zur internen Umgebung zählen dabei Gelenkstellung, Motorstatus und die aktuell verfolgten, möglicherweise interagierenden *Teilziele*.
Mit Hilfe der auf diese Weise erhaltenen Daten kann der Agent einen zeitlichen *Erfahrungsraum* aufbauen. Der Erfahrungsraum ist ein multidimensionaler Raum, welcher über den angenommenen Zeitraum den kompletten, für den Agenten wahrnehmbaren internen und externen Zustand repräsentiert. Er kann genutzt werden, um eine verbesserte Aktionsauswahl zu treffen. Hierzu muß der Agent in der Lage sein, Aktionen/Aktionssequenzen im Erfahrungsraum zu identifizieren. Er muß die Relevanz einzelner Daten bzw. deren Änderung in der Zeit für ausgewählte Einzelaktionen und Teilziele erkennen.
Der Erfahrungsraum definiert ebenfalls den in jedem Schritt auftretenden *Feedbackwert*. Dieser stellt ein erstes, einfaches Kriterium zur Einordnung

der aktuellen Situation und der aktuell durchgeführten Aktion dar. Soll der Agent aus Erfahrung lernen, so muß dieser Feedbackwert in den Erfahrungsraum und damit in die Aktionsauswahl einbezogen werden.
Als Maß der Verbesserung wird bei klassifizierendem Lernverfahren (z.B. tree induction) der Informationsgewinn (information gain) bewertet. Auch diese Größe wird in Abhängigkeit von der gewählten Domäne und dem gewählten Lernverfahren definiert.

3.3.1 Lernen in behaviorbasierten Systemen

Die in Kapitel 2.6 dargestellten traditionellen Lernverfahren zeichnen sich durch eine gemeinsame Eigenschaft aus: Lernverfahren und Repräsentation des Gelernten sind *getrennt*. Weiterhin arbeitet das Lernverfahren *global*.
Dies steht im offensichtlichen Widerspruch zu der Grundannahme behaviororientierte KI, welche intelligentes Verhalten als das Zusammenspiel *lokaler* Prozesse versteht.
Forschungsergebnisse aus dem Bereich der Neurophysiologie, welche durch Techniken wie Kernspintomographie und PET erst möglich wurden, zeigen immer deutlicher, daß das Gehirn als Ganzes zu betrachten ist. Nicht ausgezeichnete Teilbereiche sind exklusiv für eine Funktion zuständig, sondern erst das Zusammenspiel einer Vielzahl von Hirnregionen ermöglicht die komplexen Fähigkeiten des Gehirns. Die hierdurch erreichte Dynamik des Systems Gehirn ist ein wesentlicher Aspekt seiner Funktionsweise (siehe hierzu auch [22]).
Auch bei der biologischen Gestaltbildung von Lebewesen spielen dynamische Prozesse eine entscheidende Rolle. Das befruchtete Ei ist sphärisch. Innerhalb kurzer Zeit teilt es sich, um 2,4,8 Zellen usw. zu bilden. Sind während der ersten Entwicklungsstufen noch alle Zellen gleichartig, so spezialisieren sie sich in den weiteren Schritten immer mehr. Innerhalb einer Woche ist klar, welche Zellen sich zu identifizierbaren Teilen des Körpers entwickeln werden. Das sich entwickelnde Lebewesen differenziert sich feiner in einzelne Segmente aus, wobei die relative Position innerhalb des Zellclusters definiert, zu welchem funktionalen Teil das Segment heranreift. Innerhalb jedes einzelnen Segmentes verläuft die Entwicklung andersartig. ([63], [43], [42]).
Nach dieser ersten Phase der Entwicklung, welche im wesentlichen durch homeotische Gene gesteuert ist, läuft ein dynamischer Prozeß der Differenzierung ab. Durch die Differenzierung der Einzelsegmente ist jedes Segment in der Lage, spezifische Transmitterstoffe zu entwickeln. Diese wandern in angrenzende Segmente und initiieren dort eine weitere Ausdifferenzierung. Die Überlagerung von Differenzierungsprozessen bestimmt so die endgültige Gestalt.

Die hier angebrachten Gründe sprechen gegen die Existenz einer zentralen steuernden Einheit innerhalb des Gehirns. Dies bedeutet, daß Einheiten reduzierter Intelligenz durch dynamische Überlagerung ihrer Fähigkeiten erst Intelligenz entstehen lassen. In letzter Konsequenz müssen die Einheiten auf Neuronen, als "Grundbaustein" des Gehirns, zurückgeführt werden. Dem einzelnen Neuron läßt sich keine ausgezeichnete Funktion mehr zuordnen, es ist funktionell neutral.
Hieraus ergibt sich eine neue Sichtweise auf den Begriff Intelligenz. Nicht die zugrundeliegenden Bausteine "Neuron" sind damit Träger der Intelligenz, sondern die Dynamik des Netzwerkes Gehirn definiert die Intelligenz.
In der Natur gibt es auch komplexe dynamische Systeme, welchen man keine Intelligenz zuschreiben würde. Beispiele hierfür sind Gase und Flüssigkeiten. Zwischen den Grundeinheiten dieser chemischen Stoffe (Molekülen) besteht eine komplexe Dynamik. Im Gegensatz zu intelligenten Systemen fehlt ihnen die Fähigkeit zur *Selbstorganisation*. Die fehlende Gesamtstruktur dieser Systeme läßt keine, über die lokale Effekte, hinausgehende systematische Beeinflussung einer Grundeinheit durch andere Grundeinheiten zu.

Der Lernmechanismus

Ausgangslage des Lernmechanismus ist eine vorgegebene Behaviorstruktur, welche die gewünschten Minimalfähigkeiten des Systems innerhalb einer vorgegebenen, jedoch nicht unbedingt deterministischen Umwelt zur Verfügung stellt. Die Auswahl des aktiven Behaviors erfolgt durch den in 3.2 beschriebenen Formalismus.
Der zentrale Steuerungsmechanismus des Lernverhaltens ist die zeitlich an die Aktivierung des Behaviors gekoppelte Ausschüttung von Distributionsnachrichten.
Die Distributionsnachrichten verteilen sich zeitlich gesteuert über die Behaviorhierarchie. Nach jeweils einem Schedulingschritt erfolgt in Abhängigkeit von lokalen und globalen Parametern das Weiterschalten. Die Datenstruktur der Distributionsnachricht ist eine Attribut/Wertstruktur mit Stellen für Zeitstempel, Identifikationscodes des aktivierten Behaviors und Aktivationsniveau der Nachricht.
Im ersten Schritt der Ausbreitung wird die Hierarchieebene des aktivierten Behaviors überflutet, dann schrittweise die nächst höheren. In jedem zeitlichen Schritt wird dabei das Aktivationsniveau der Distributionsnachricht verringert. Beim Erreichen des unteren Schwellwertes (in diesem Fall 0) wird die Distributionsnachricht deaktiviert. Die Verringerung des Aktivationsniveaus ist abhängig davon, ob die Distributionsnachricht innerhalb der gleichen Ebene liegt.

Treffen zwei von unterschiedlichen oder auch gleichen Behaviors ausgesandte Distributionsnachrichten aufeinander, kann sich eine Vernetzung ergeben. Diese findet jedoch nur dann statt, wenn die Zeitstempel der jeweiligen Distributionsnachricht sich maximal um ein vordefiniertes Intervall (in diesem Fall 1) unterscheiden und die Aktivationsniveaus im selben Bereich liegen. Die Vernetzung entspricht dem Aufbau eines *Aktivationskanals* zwischen den Behaviors. Der Treffpunkt der Distributionsnachricht innerhalb der Behaviorhierarchie definiert die Stelle, an der das so gebildete *abstrakte Behavior* eingefügt wird. Ein abstraktes Behavior kombiniert die zugrundeliegenden Behaviors (in Abhängigkeit von den durch die Distributionsnachrichten übermittelten Daten) zu einer Sequenz.
Weiterhin beeinflußt die zeitliche Differenz die Stärke der Vernetzung. Zeitlich nahe Aktivierung führt zu stärkerer Vernetzung als zeitlich ferne.
Durch die zeitliche Ausdehnung der Distributionsnachricht entsteht eine rasterartige Aufteilung der Behaviorhierarchie. An den Schnittstellen des Rasters werden die Verbindungen aufgebaut. Der einfache Mechanismus sorgt für eine feine Differenzierung im Aufbau von Verbindungen.
Ein weiteres Merkmal des Mechanismus ist es, näher zusammenliegende Schichten stärker zu vernetzen als weiter voneinander entfernte. Ein derartiges Verhalten des Lernmechanismus macht durchaus Sinn. Der initiale Zustand der Behaviorhierarchie ist gerade so konstruiert, daß die Behaviors zweier benachbarter Schichten eine stärkere Beziehung zueinander haben als weiter auseinanderliegende. Jede Ebene stellt eine Abstraktionsebene des Verhaltens dar. Der Aufbau engerer Verbindungen zwischen benachbarten Schichten bewirkt demnach, daß Lernen auf unterschiedlichen Abstraktionsniveaus stattfindet.
Treffen Distributionsnachrichten gleicher Behaviors wiederholt aufeinander und ist bereits eine Vernetzung der Behaviors etabliert, so wird der entsprechende Kanal verstärkt. Damit realisiert das Verfahren die *Hebb'sche Regel* [31].
Somit wird durch einen einfachen, lokalen Mechanismus eine hohe Variabilität der Differenzierung der Behaviorhierarchie erreicht. Faßt man die Eigenschaften des Lernmechanismus schlagwortartig zusammen, ist Folgendes zu nennen:

1. hohe strukturelle Variabilität der Vernetzung
2. Lernen auf unterschiedlichen Abstraktionsebenen wird unterstützt
3. Realisierung der Hebb'schen Regel
4. Integration von prozeduralem und deklarativem Wissen

5. keine Trennung von Exploration/Exploitation

6. Überdeckung (Vergessen) von falschen Aktionen

7. Einbeziehung bereits gelernter Verhaltensweisen in neu zu lernende

8. Beibehaltung der Reaktivität

9. Anschlußmöglichkeit an "höhere Systeme" und damit

10. Beeinflussung des Verhaltens von außen

3.3.2 Die Distributionsnachricht

Die Distributionsnachricht spielt in diesem Ansatz eine zentrale Rolle. Sie bestimmt maßgeblich, wie sich die abstrakten Behaviors bilden.
Grundsätzliche Funktion der Distributionsnachricht ist es, für eine Verkettung der Behaviors zu sorgen. Die transportierte Information ist damit nichts anderes als eine eindeutige Folge von "Absenderadressen". Diese spezifiziert zu einem späteren Zeitpunkt die Folge der zu aktivierenden Behaviors.
Die Distributionsnachricht ist in Form eines erweiterbaren Frames realisiert. Während der Ausbreitung über die Behaviorhierarchie wird Information hinzugefügt und bereits gespeicherte modifiziert. Die Datenstruktur der Distributionsnachricht hat folgende Attributstellen:

- Liste der bereits berührten Behaviors

- Aktivationswert, ist ausschlaggebend für die Größe der abstrakten Behaviors

- Selektionsschlüssel, definiert für welche Behaviors die Distributionsnachricht lesbar ist

- Ausbreitungsschlüssel, definiert die Art der Ausbreitung (Ausbreitungsgeschwindigkeit)

3.3.3 Der Selektionsschlüssel

Anhand des Selektionsschlüssels wird entschieden, welche Behaviors die Distributionsnachricht lesen können. Hierdurch ist eine von der Distributionsnachricht abhängige Ausbreitung über die Behaviorhierarchie möglich. Nur

Spezifikation Distributionsnachricht	
Slot	Value
Besuchte Behaviors	list of Behavior IDs
Aktivation	real
Selektionsschlüssel	skeyformat
Ausbreitungsschlüssel	dkeyformat

Tabelle 3.3: Schematische Darstellung der Distributionsnachricht
In der Distributionsnachricht werden die bereits "besuchten" Behaviors, die Aktivation der Nachricht, der Selektionsschlüssel und der Ausbreitungsschlüssel abgelegt.

wenn Selektionsschlüssel und Identifikation des aktiven Behaviors zusammenpassen, wird das Behavior in die Liste der besuchten Behaviors aufgenommen und damit vernetzt.
Die Kodierung des Selektionsschlüssels erfolgt durch einen Codestring, bestehend aus Basisidentifikatoren.

3.3.4 Der Ausbreitungsschlüssel

Der Ausbreitungsschlüssel definiert, in welche Richtungen und mit welcher Geschwindigkeit sich die Distributionsnachricht ausbreitet. Mögliche Richtungen sind dabei `rechts, links, unten und oben`, mögliche Geschwindigkeiten sind `alle und einige`. Bei "`alle`" wird jedes Behaviors einer Schicht angesprochen, bei "`einige`" werden nur alle n Behaviors angesprochen.

3.3.5 Abstraktion und Generalisierung

Durch die zeitliche Verkettung von Behaviors wird ein Kanal aufgebaut, welcher die Kommunikation zwischen den beteiligten Behaviors zuläßt.
Ist der Kanal etabliert, so erfolgt im ersten Schritt ein Austausch der behaviorrelevanten Daten. Hierzu zählen die Anwendungsbedingungen und die reaktive Zone. Die Anwendungsbedingung muß übertragen werden, um sicherzustellen, daß das gebildete abstrakte Behavior in der gleichen Situation zum Einsatz kommt, wie das erste seiner Subbehaviors. Der Austausch der reaktiven Zone sichert ein einwandfreies Scheduling der neuen Behaviors.

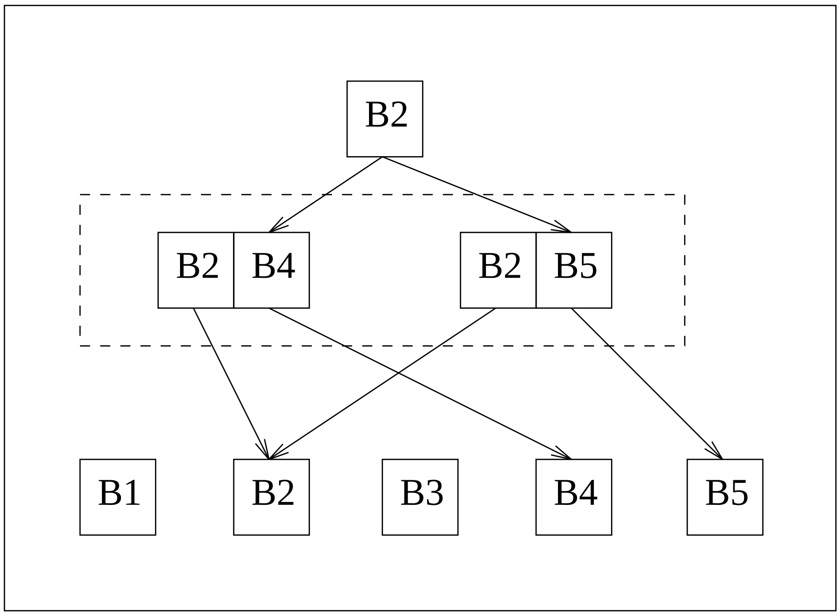

Abbildung 3.8: Zusammenlegung von Behaviors
Bei der Zusammenlegung von Behaviors wird überprüft, ob die Behaviors einer Teilfolge eines gebildeten abstrakten Behaviors die gleiche Aktivierungsbedingung enthalten. Unter dieser Voraussetzung ist es möglich, ein abstraktes Behavior auf der nächst höheren Stufe zu bilden, welches die gemeinsame Aktivierungsbedingung überprüft und die folgenden Behaviors aktiviert.

Im zweiten Schritt werden die Aktivationskanten zwischen abstraktem Behavior und den Subbehaviors aufgebaut. (siehe auch Abb.: 3.9)

3.3.6 Zusammenlegung abstrakter Behaviors

Durch den beschriebenen Formalismus können sich Behaviors mit gleichen Ausgangsbedingungen herausbilden. Diese können wiederum zusammengefasst werden (siehe auch Abb. 3.8). Die Zusammenlegung bewirkt, daß das erste Behavior des abstrakten Behaviors ausgelagert wird und Anfangsbehavior ist für eine Auswahl an möglichen folgenden abstrakten Behaviors. Die Generalisierung erfolgt innerhalb einer Schicht. Da sie auf alle Schichten angewandt wird, entstehen sukzessiv immer bessere Verallgemeinerungen von Verhaltensmustern auf allen Ebenen.

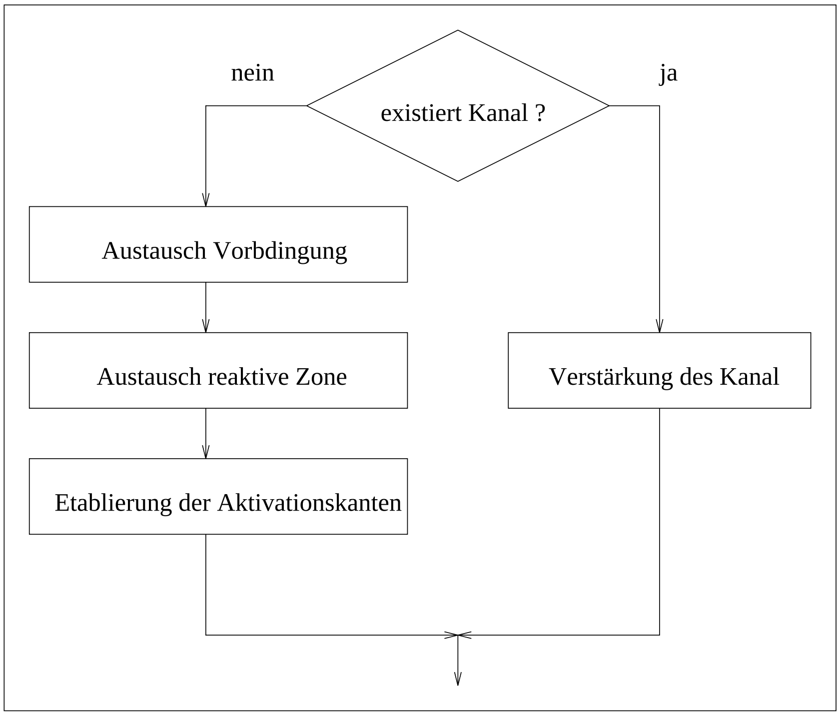

Abbildung 3.9: Vernetzungsprozedur der Behaviors
Für die Etablierung eines abstrakten Behaviors muß zunächst geprüft werden, ob bereits ein Kanal (also die Kombination von zwei, oder mehr Behaviors) existiert. Ist dies der Fall, so wird der Kanal verstärkt, indem die Selektionsschwelle des abstrakten Behaviors vermindert wird. Existiert der entsprechender Kanal noch nicht, so wird durch Austausch der Vorbedingung und der reaktiven Zone ein abstraktes Behavior konstruiert und in die Hierarchie eingeordnet.

Derartig konstruierte abstrakte Behaviors werden jeweils eine Ebene über der des höheren an der Verbindung beteiligten Behaviors in die Behaviorhierarchie eingetragen. Als Anwendungsbedingung wird die Anwendungsbedingung des für den ersten Teilschritt eingesetzten Behaviors verwendet. Die abstrakten Behaviors aktivieren nicht unmittelbar die Effektoren, sondern geben Aktivationspotential an ihre Teilkomponenten ab. Das abstrakte Behavior gliedert sich damit in den normalen Schedulingzyklus ein.

3.3.7 Parameter, die das Lernverhalten beeinflussen

Die sich entwickelnden abstrakten Behaviors sind abhängig von einer Vielzahl von Parametern. Zum einen spielt die Außenaktivation eine entscheidende Rolle. Sie ist die Grundlage für die primäre Selektion eines Behaviors und damit für eine potentielle Aktivation.
Im nächsten Schritt wählt das Aktionsselektionsverfahren aus der Menge der aktivierten Behaviors dasjenige aus, welches zu diesem Zeitpunkt die Kontrolle erhält.
Bei einem veränderten Aktionsselektionsverfahren werden entsprechend auch andere abstrakte Verhaltensweisen vom Agenten erworben.
Ein entscheidender Faktor für die Bildung der abstrakten Behaviors ist das Ausbreitungsverhalten der Distributionsnachricht. Es steuert zum einen die Größe der abstrakten Behaviors und die Abfolge der zugrundeliegenden Behaviors. Damit steuert die Ausbreitung sowohl das strukturelle CA, das temporale CA und das Control CA (siehe Abschnitt 2.7.4).
Durch das Ausbreitungsverhalten der Distributionsnachricht wird weiterhin gesteuert, in welcher Abstraktionsschicht das abstrakte Behavior eingeordnet wird.

3.3.8 Passive und aktive Ausbreitung

Jede Distributionsnachricht verfügt über "Schaltstellen". Diese bestimmen, welche Behaviors die Nachricht lesen können. Damit ist es möglich, zwischen einer passiven und einer aktiven Ausbreitung zu unterscheiden.
Bei der passiven Ausbreitung enthält die Distributionsnachricht eine Universalschaltstelle. Diese kann von beliebigen Behaviors gelesen werden und somit die Verbindung der Behaviors indizieren. Bei der Weitergabe der Distributionsnachricht tritt keine Veränderung der Schaltstelle auf.
Bei der aktiven Ausbreitung wird eine differenzierende Schaltstelle verwendet. Damit sind nur spezielle Behaviors in der Lage, die Distributionsnachricht zu lesen. Das Behavior kann dann die Schaltstelle der Nachricht mo-

difizieren und bestimmen, an welche weiteren Behaviors die so veränderte Distributionsnachricht ankoppeln kann.
Die erlaubten Modifikationen der Schaltstelle einer Distributionsnachricht sind eine dem jeweiligen Behavior inhärente Eigenschaft; sie ist also vom Programmierer vorgegeben.

3.4 Experimente

Anhand einer Reihe von Experimenten soll die Funktionstüchtigkeit des Lernverfahrens gezeigt werden. Um eine ausreichend breite Basis für aussagekräftige Ergebnisse zu schaffen wurden für die Steuerung des Manipulator drei unterschiedlich komplexe Behaviorsysteme entwickelt. Anschließend wurde versucht den Systemen folgende abstrakte Verhaltensweisen beizubringen:

- Aufrichten
- an spezielles Objekt annähern
- von Objekt wegdrehen
- spezielles Objekt betasten

In Abhängigkeit von der Eigenaktivität des Systems sollte die Verhaltensweise entweder durch autonomes Handeln erworben werden, oder bei passiven Systemen durch externe "Instruierung". Hierbei gilt, daß "Instruktion" nicht im linguistischen Sinne zu verstehen ist, sondern als direkte Manipulation des Systems durch die grafischen Schiebeelemente. Im Bereich des maschinellen Lernens sind die Experimente also dem "Lernen durch Vormachen" und dem "Lernen aus Erfahrung" zuzuordnen.
Im Rahmen dieser Arbeit war es nicht möglich eine (wünschenswerte) detaillierte mathematische Analyse der Lernergebnisse auszuarbeiten. Die Beschreibung der Ergebnisse bleibt deshalb informell und deskriptiv. Vorgreifend zum abschließenden Kapitel kann jedoch hier schon gesagt werden, daß die Dynamik der Entwicklung der Behaviorhierarchien unerwartet hoch war. Die Anzahl der abstrakten Behaviors explodierte geradezu (> 1000) und erst durch langwieriges Experimentieren in der Einstellung der Lernparameter war es möglich das gewünschte Verhalten zu erzeugen. Prinzipiell ist die Funktionstüchtigkeit des Verfahrens jedoch gesichert.

3.4.1 Der vorsichtige Roboter

Die diesem System zugrundeliegende Behaviorhierarchie steuert den Roboterarm sehr "defensiv". Der Kontakt mit externen Objekten wird vermieden. Nur durch explizite Instruktion agiert das System, es sei denn, ein Objekt nähert sich und der Arm muß ausweichen. Sind keine externen Stimuli vorhanden, verharrt das System in der aktuellen Position (Abb. 3.10 zeigt das Steuerungssystem)

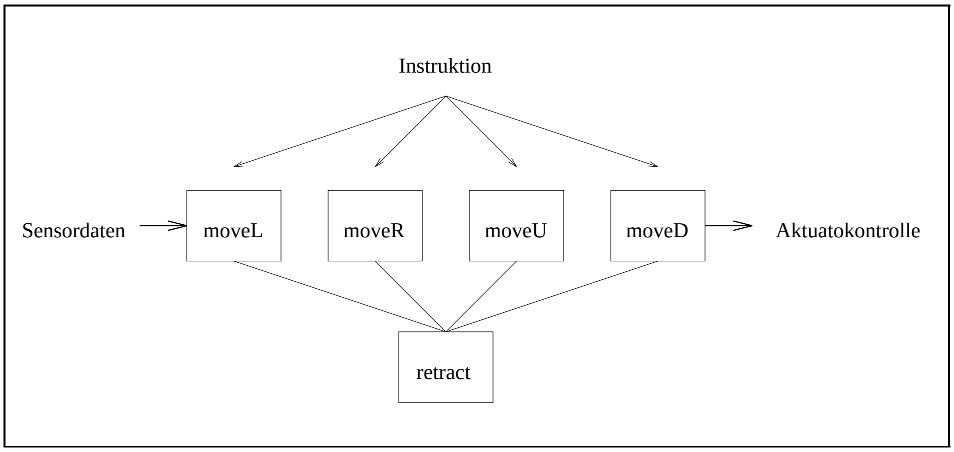

Abbildung 3.10: Behaviorhierarchie des vorsichtigen Roboters. Bei Kontakt der taktilen Sensoren mit externen Objekten wird *retract* aktiv und steuert den Arm aus dem Kontaktbereich heraus.

- hohe Aktivation von `retract`
- Aktion nur durch Instruktion oder durch dynamische Änderung des Weltzustandes
- keine Eigenaktivität

Die Behaviorhierarchie ist flach. Auf der untersten Ebene (Ebene 0) befindet sich lediglich `retract`. Auf der nächst höheren Ebene (Ebene 1) befinden sich vier Bewegungsbehaviors: `turn_left, turn_right, hand_up, hand_down`. Alle Bewegungen sind gleichwertig, können jedoch lediglich durch Instruktionen oder durch Sensordaten initiiert werden.

Die tiefe hierarchische Stellung und eine kleine Aktivationsschwelle von `retract` bewirkt, daß dieses Behavior sich immer dann durchsetzt, wenn die taktilen Sensoren im Handbereich einen Kontakt mit Objekten der Umgebung melden. Der Roboter weicht dann in die entsprechende Gegenrichtung aus.

Wird durch eine Instruktion ein Behavior der höheren Ebene aktiviert, so ist dieses nur solange aktiv, wie kein Kontakt wahrgenommen wird. Sobald dies der Fall ist, übernimmt wieder *retract* die Kontrolle; das System weicht aus und der Roboter beendet seine Bewegung.

3.4.2 Der neugierige Roboter

Das vorsichtige System ist in einem nächsten Schritt um eine höhere Behaviorebene erweitert worden. In dieser befinden sich zwei weitere Behaviors: *center* und *maximize-area*, welche auf rote Objekte (Schraube und Würfel) reagieren.

Tritt ein solches Objekt in den visuellen Bereich ein, versucht das Behavior *center*, den Roboter so zu steuern, daß das Bild des Objektes im Zentrum der visuellen Matrix liegt. Hier wird die Fähigkeit des visuellen Systems genutzt, den Offsetvektor zwischen dem Zentrum der visuellen Matrix und dem Schwerpunkt der Objektabbildung zu errechnen. Aus dieser sensorischen Information erhält das Behavior einen Hinweis darauf, wie weit und in welche Richtung das System seine Position ändern muß.

Das Behavior *maximize-area* reagiert ebenfalls auf visuellen Input. Wird ein rotes Objekt detektiert, so versucht das Behavior, die von dem Bild des Objektes auf der visuellen Matrix eingenommene Fläche zu vergrößern. Hierzu steuert das Behavior den Roboterarm ein vordefiniertes Stück nach vorn. Nach vorn heißt in diesem Fall, daß die Gesamtwinkelsumme der Gelenksensorik konstant gehalten wird. Die Gelenkwerte der unteren beiden Armsegmente (Ober- und Unterarm) des Roboters werden um jeweils ein Grad erhöht. Im gleichen Schritt wird der Winkel des dritten Segmentes (Handgelenk) um 2 Grad vermindert. Der Arm beugt sich, die Hand behält aber annähernd ihre ursprüngliche Ausrichtung bei.

Center und *maximize-area* führen keine herkömmliche auf inverser Kinematik beruhende Bahnplanung aus. Jedes der beiden Behaviors definiert lediglich eine lokale Reaktion auf den sensorischen Input. Erst durch die Überlagerung der Verhaltensweisen ergibt sich die Annäherung an das Objekt. Die Bahn, welcher der Roboter folgt, wird dabei stückweise von jeweils einem der Behaviors kontrolliert.

Die Neugier des Roboters endet, falls ein taktiler Sensor den Kontakt mit einem Objekt meldet. *retract* wird aktiv und steuert den Arm aus dem Kontaktbereich heraus. Ist das Objekt auch noch sichtbar, nachdem *retract* die Kontrolle hatte, erhalten die Behaviors der zweiten Ebene die Kontrolle zurück. Der Roboterarm nähert sich wieder an, wird jedoch bei Kontakt unmittelbar wieder in die andere Richtung gesteuert und verliert so (nach eventuellen weiteren Versuchen) das Objekt aus dem visuellen Wahrnehmungsfeld.

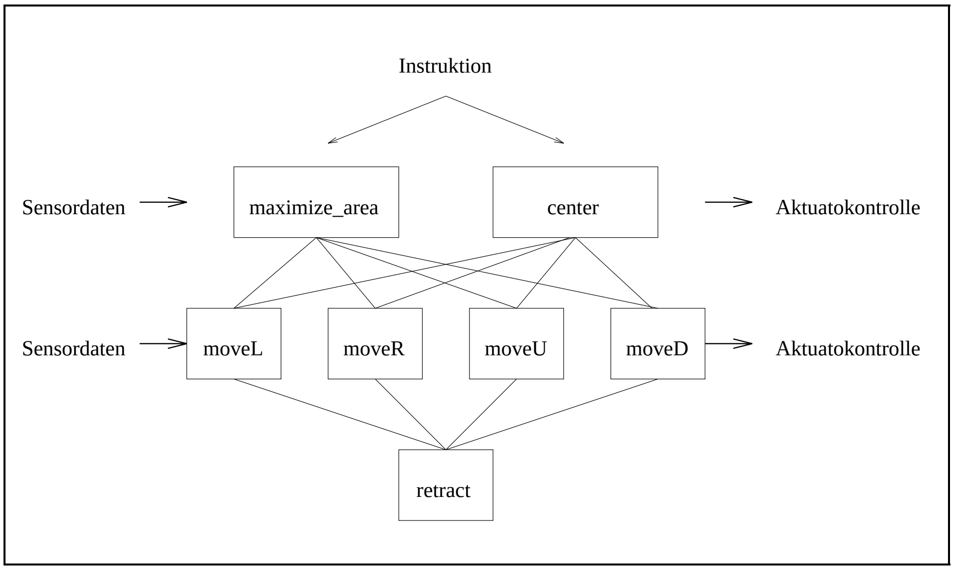

Abbildung 3.11: Die Behaviorhierarchie des neugierigen Roboters. Durch die Überlagerung der Behaviors *center* und *maximize-area* wird eine Annäherung an rotfarbige Objekte erreicht.

3.4.3 Der tastende Roboter

Der tastende Roboter verfügt über die komplexeste, initiale Behaviorhierarchie. Es sind weiterhin alle zuvor beschriebenen Behaviors vorhanden. Zusätzlich ist auf der dritten Behaviorebene das Behavior *tactile-exploration* implementiert, welches im Gegensatz zu *retract*, versucht Objekte der Umwelt taktil zu erkunden (siehe Abb. 3.12).
Genau wie bei dem neugierigen System nähert sich der Roboterarm visuell detektierten Objekten an: nachdem ein Objekt im Blickfeld erscheint, wird der Arm durch Überlagerung von Zentrierung und Flächenmaximierung an das Objekt herangeführt. Statt jedoch bei Kontakt (Übernahme der Kontrolle durch *retract*) zurückzuschrecken, sorgt *tactile-exploration* für ein Abtasten der Oberfläche. Dazu bewegt sich der Arm wahlweise nach rechts oder links, wobei der frontale telemetrische Sensor ständig einen konstanten Wert erhalten soll. Verliert das System diesen, so steuert *tactile-exploration* den Arm aus dem Objektbereich heraus und hebt den Arm an.
retract ist weiterhin aktiv, erhält jedoch nur die Kontrolle, falls sich durch dynamische Weltänderung ein Kontakt mit der Außenwelt ergibt. Durch derartige vom System nicht kontrollierte Annäherung ergibt sich eine Gefährdung der Systemintegrität, welche das System zu vermeiden sucht. Der Roboterarm "schreckt" also vor sich annähernden Objekten zurück.

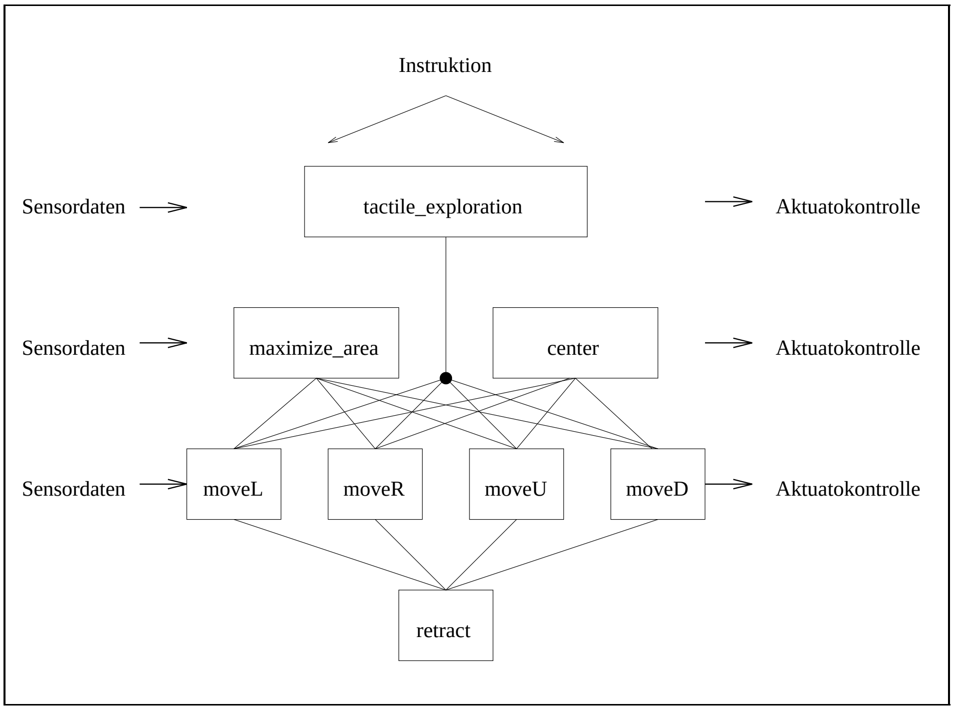

Abbildung 3.12: Die Behaviorhierarchie des tastenden Roboters. Das Behavior der dritten Ebene *tactile-exploration* untersucht erkannte Objekte aktiv mit Hilfe der taktilen Sensorik

3.4.4 Versuchsprotokolle

Die folgenden Versuchsprotokolle sollen einen Eindruck vermitteln, wie das System bei den unterschiedlichen Aufgaben reagierte. Es wird auf eine grafische Darstellung der Entwicklung der Knotenzahl in Abhängigkeit von Experimentwiederholung verzichtet. [5]

Aufrichten: Im ersten Experiment sollte das System die Fähigkeit erwerben, sich in eine vorgegebene Position zu bewegen; es sollte sich aufrichten.

Nicht vorgegeben war die jeweils beste Aktion, welche innerhalb eines Zustandes durchzuführen ist. Die Explorationsstrategie war somit zu Beginn des Experiments zufällig.

Häufig durchgeführte Teilhandlungen, erhalten in der Folge eine höhere Aktivationsschwelle. Entsprechend umgekehrt wurde bei Teilhandlungen, die selten eingesetzt werden die Aktivationsschwelle gesenkt.

Bei der Durchführung des Experiments mit einem passiven System läßt sich durch Aktivation der Behaviors *hand up* und *hand down* schnell ein abstraktes Behavior aufbauen, welches die Steuerung in die Zielposition vornimmt. Dieses Training durch Vormachen erzeugt eine Handlungssequenz, welche eine sehr niedrige Aktivationsschwelle hat und somit unmittelbar die Kontrolle übernimmt. Wird das System bei der Ausführung gestört, etwa durch einen in eine andere Richtung führenden Steuerungsimpuls, führt dies nicht zum Abbruch der Handlungskette. Nach Beendigung der unterbrechenden Handlung wird der Rest der Handlungskette durchgeführt.

Wird das Experiment mit einem aktiven System durchgeführt, ergeben sich ähnliche Beobachtungen. Zwar ist hier die Explorationsstrategie nicht so effektiv, wie bei einem instruierten System. Durch die Erhöhung der Aktivationsschwelle der nicht erfolgreichen Teilhandlungen wird jedoch erreicht, daß auch Teilsequenzen, welche erfolglose Behaviors enthalten, in geringerem Maße ausgewählt werden. Dies hat eine weitere Erhöhung der Aktivationsschwelle zur Folge, welche dazu führt, daß die abstrakten Behaviors nicht weiter ausgewählt werden.

[5] Ähnliche Darstellungen werden in der Literatur zum maschinellen Lernen immer wieder eingesetzt um die Güte der Verfahren zu demonstrieren, erwiesen sich aber häufig als optisch ansprechende, aber wenig aussagekräftige Reformulierungen der Tatsache "das beschriebene Verfahren X funktioniert und ist für die beschriebene stark eingegrenzte Domäne n/10 Prozentpunkte effizienter als Verfahren Y".

Handlungssequenzen mit erfolgreichen Teilhandlungen werden in ihrer Aktivationsschwelle weiter abgesenkt, dadurch in höherem Maße ausgewählt und somit verstärkt. Dies führt schließlich zu einem abstrakten Behavior, welches aus einer Kette von *hand up* Aktionen zusammengesetzt ist.

Das System hat somit ein komplexes Behavior aufgebaut, welches die Aufrichtung bewirkt. Das System ist weiterhin reaktiv. Unmittelbar nach der Aktivierung des Systems übernimmt das abstrakte Behavior die Kontrolle und sorgt für die schnelle Aufrichtung des Roboterarmes in die Optimalposition.

Durch Wiederholung der Experimente aus verschiedenen Ausgangspositionen heraus kommt es zur Verallgemeinerung der Aktivationsbedingung. Das abstrakte Behavior übernimmt in jeder Situation die Kontrolle. Lediglich in Kontaktsituationen greift *retract* in die Steuerung ein.

Das so konstruierte abstrakte Behavior ist in der obersten Ebene eingeordnet. Es hat die höchste Kompetenz und wird ausschließlich aus Handlungselementen der niedrigen Ebenen aufgebaut.

Annäherung an Objekte: In einem weiteren Experiment soll das System auf spezielle Objekte in einer erlernten Art und Weise reagieren. Hierzu wird das System durch den Instruktor gesteuert. Ausgangssituation war hier die aufrechte Position.

Das Training des passiven System erwies sich als außerordentlich langwierig. Zwar wurde, wie beim Aufrichten, eine Beugesequenz hin zur Tischebene relativ schnell erworben. Die notwendige Abstraktion in Bezug auf die Objekteigenschaften gelang in den ersten Experimentserien nicht. Erst durch eine Modifikation des Behaviorsystems (Einbeziehung der Kameradaten `ohne` Auswertung in der Aktivierungsbedingung von `moveD`) gelang es, die entsprechenden Parameter an das abstrakte Behavior weiterzuleiten und so zumindest eine Reaktion auf die Farbe des Objektes herzustellen.

Wesentlich einfacher war das Training der aktiven Systeme, da diese ohnehin so konstruiert waren, ein Objekt zu fokussieren und sich ihm anzunähern. Es mußte lediglich ein Behavior aufgebaut werden, welches die Beugung des Armes bis zum Sichtkontakt bewirkte. Anschließend übernahm die entsprechende Behaviorhierarchie die Annäherung. Nach einer Anzahl von Wiederholungen wurde so die neue abstrakte Handlungsweise aufgebaut, welche dann durch das System in einer entsprechenden Situation selbständig durchgeführt wird.

Von Objekt wegdrehen: Bei diesem Experiment sollte sich der Arm von einem beliebigen Objekt entfernen und zwar basierend auf den visuellen Informationen (`retract` sorgt ohnehin für die Vermeidung von physischen Kontakt, jedoch basierend auf taktilen/telemetrischen Sensordaten).

Für das passive Steuerungssystem gelten die gleichen Aussagen, wie für das vorherige Experiment. Erst nach der geringfügigen Modifikation des Behaviorsystem reagierte das System auf die Objekteigenschaften.

Bei den Trainingsläufen der aktiven Systeme mußte für die Unterdrückung der eingebauten Annäherungseigenschaft gesorgt werden. Dies wurde durch wiederholte "Störung" des Systems tatsächlich erreicht. Sobald das System sich einem Objekt visuell geleitet annäherte, wurde der Roboter so gesteuert, daß eine Rotation im Basisgelenk stattfand, welche den visuellen Kontakt beendete. Dies führte nach einer entsprechenden Anzahl von Wiederholungen dazu, daß der Roboter zwar weiterhin versuchte sich einem Objekt zu nähern, bevor die Bewegung jedoch abgeschlossen war, daß neu erworbene Rotationsverhalten übernahm und den Roboter aus dem visuellen Bereich heraussteuerte und so die Annäherung beendete.

Spezielles Objekt "betasten": Ziel dieses Experimentes war es, zu untersuchen ob es gelingt das unterste Behavior `retract` zu überdecken. Da zu erwarten war, das sich in Bezug auf das passive System die Beobachtungen aus den vorherigen Experimenten auch hier wiederholten, wurde dieses Experiment nur mit dem "neugierigen" System durchgeführt. Das "tastende" System war ohnehin darauf ausgelegt sich einem Objekt zu nähern und bei Berührung `retract` zu unterdrücken.

Der Versuchsverlauf ähnelt dem gerade beschriebenen. Via Instruktion wird das System gezwungen Kontakt zu einem Objekt herzustellen. Durch die geringe Aktivationsschwelle von `retract` gelang es aber nicht, das gebildete abstrakte Behavior häufig genug zu aktivieren. `retract` setzte sich also immer wieder durch. Erst nach einer Modifikation des Behaviorsystems gelang die Überdeckung.

Daß die Überdeckung von `retract` hier nicht gelang ist insofern erfreulich, weil sich zeigte, daß das modifizierte Behaviorsystem (ohne die Erweiterung durch die gelernten Behaviors) nicht einwandfrei funktionierte. Der Roboter war nicht mehr in der Lage in jeder Situation einem Objekt schnell genug auszuweichen.

Kapitel 4

Ergebnisse und Ausblick

Untersuchungsgegenstand dieser Arbeit ist die Entwicklung eines lernfähigen behaviorbasierten Agenten. Wesentlich für die Untersuchung der Lernverfahren war der Einsatz eines adäquaten Simulationssystems, welches unter realistischen Annahmen entwickelt worden ist. Ausgehend von den Erfahrungen mit einer einfachen Simulation eines mobilen Agenten und der Realisierung eines mobilen Roboters wurde das Konzept des Simulationssystems entwickelt.
Das Simulationssystem unterliegt einer Reihe von Anforderungen, welche sich aus dem Steuerungsansatz ergeben. Der behaviororientierte Ansatz basiert auf der unmittelbaren Verknüpfung von Situation und Handlung. Entsprechend ist Simulationssystem so ausgelegt , daß realitätsnahe Sensorwerte generiert werden. Als Entwicklungsumgebung unterstützt das System den Benutzer bei der Entwicklung von Steuerungssystemen. Konfigurationen sind veränder- und speicherbar. Die grafische Oberfläche erleichtert die Bedienung des Systems.
Für die Entwicklung der Steuerungssysteme wurde eine Behaviorsprache definiert und ein geeigneter Interpreter implementiert. Das interpretierende System wurde in LISP programmiert. Neu entwickelte Primitivfunktionen übernehmen die Steuerung der Simulation. Der Interpreter sorgt außerdem für das Scheduling der Behaviors. Die Auswahl des kontrollierenden Behaviors erfolgt unter Berücksichtigung des aktuellen Sensorzustandes, möglicher Einschränkungen durch die reaktive Zone und unter Einbeziehung von erlernten abstrakten Behaviors.
Der letzte Schritt war die Entwicklung eines in das Steuerungssystem integrierten Verfahrens für den Erwerb von Handlungssequenzen. Der zugrundeliegende Mechanismus lehnt sich an biologische Vorgänge an, wie sie bei der Gestaltentwicklung von mehrzelligen Lebewesen beschrieben worden sind.
Durch die Wahl des Steuerungsansatzes wird sichergestellt, daß der Agent in

einer dynamischen Umwelt sicher handeln kann. In Abhängigkeit von dem definierten Grundverhalten reagiert das System auf sensorischen Input durch situationsadäquate Handlungssequenzen.
Grundlage der durchgeführten Experimente ist ein Aktionsselektionsverfahren welches speziell an die Anforderungen des Lernverfahrens angepasst ist. Dieses Verfahren ist der in drei Punkten modifizierter Ansatz der Subsumptionsarchitektur nach Brooks.
Im Gegensatz zur Subsumptionsarchitektur von Brooks, bei der alle Behaviors, deren Koppelung mit der Sensorik und ihre Koppelung untereinander vom Entwickler statisch vorgegeben ist, kann ein in der entwickelten Behaviorsprache definiertes Steuerungssystem seine Fähigkeiten zur Laufzeit erweitern.
Damit ist das System sowohl in der Lage, auf unvorhergesehene Ereignisse innerhalb einer dynamischen Umwelt zu reagieren und kann außerdem aus Erfahrungen lernen und so das Verhaltensrepertoire ausbauen.
Durch die Einbindung der Distributionsnachricht in das Aktionsselektionsverfahren ergibt sich ein erweitertes Sequenzbildungsschema für aus zugrundeliegenden Behaviors aufgebaute Komplexhandlungen.
Das Verfahren läßt sich in die Gruppe der Reinforcement Lernverfahren einordnen. Das innerhalb dieser Gruppe von Lernverfahren am häufigsten eingesetzte Q-Learning wurde mit Erfolg bei behaviorbasiert gesteuerten mobilen Robotern eingesetzt.
Durch die hierarchische Ordnung der Behaviors wird es möglich, gezielt Initialwissen einzubringen. Dieses hat sowohl Auswirkungen auf das Basisverhalten, wie auch auf das Lernverhalten. Durch Basisverhalten, Umweltbedingungen und eingebrachten Initialwissen wird die Entwicklung behaviorbasierter Steuerungssysteme erleichtert.
Die so konstruierten Behaviorhierarchien sind effektiver in der Aktionsauswahl. Bei herkömmlichem Q-Learning wird eine Matrix aufgebaut, welche alle möglichen Kombination von Verhaltensweisen repräsentiert. Bei dem untersuchten Verfahren werden innerhalb der gelernten Hierarchie lediglich diejenigen Verhaltenssequenzen repräsentiert, welche häufig zum Einsatz kommen.
Die Aufhebung der Trennung von Lern- und Handlungsphase bewirkt, daß ein Vortrainieren der Behaviorhierarchie unter Idealbedingungen möglich wird. Beim Einsatz unter Realbedingungen kann dann eine "Nachbesserung" der gelernten Aktionssequenzen stattfinden.
Muß für eine korrekte Sequenzierung der Teilbehaviors durch die Q-Matrix der Zustand des Systems zu jedem Zeitpunkt bekannt sein, so ist diese Bedingung hier hinfällig. Auch nach einer durch die Fehlinterpretation von Sensordaten ausgelösten inadäquaten Handlung, bzw. durch eine Reaktion

auf dynamische Änderungen innerhalb des Weltzustandes, ist das System in der Lage eine vorher begonnene Handlungssequenz zu beenden.
Für das Verfahren ergeben sich also folgende Vorteile gegenüber Q-Learning:

- nicht alle möglichen Kombination werden berücksichtigt
- durch hierarchische Ordnung kann Initialwissen eingebracht werden
- keine Trennung von Lern- und Handlungsphase
- Lernverfahren ist den Aktionsselektionsprozeß integriert
- das Lernverfahren muß nicht zu jedem Zeitpunkt wissen, in welchen Zustand sich das System befindet

Nachteilig an dem Verfahren ist hohe Anzahl an Kontrollparametern. Dies führt schnell dazu, daß das Lernverhalten nicht mehr durchschaubar ist. Entsprechend explodiert in den ersten Experimentalsitzungen die Anzahl der abstrakten Behaviors.

4.1 Anbindung an erweiterte behaviorbasierte Systeme

Sollen behaviorbasierte Systeme über rein reaktives Verhalten hinausgehende Fähigkeiten verfügen, bietet sich eine Erweiterung durch die Ankoppelung einer klassisch realisierten deliberativen Komponente an. Diese soll in der Lage sein, längerfristige Zielsetzungen festzulegen, was sowohl durch ein klassisches Handlungsplanungssystem erfolgen kann oder aber auch in einfacherer Form als Bibliothek von instantiierbaren Planschablonen für komplexe Handlungsfolgen denkbar ist. Deliberatives und behaviorbasiertes System müssen dabei in einer Art gekoppelt sein, daß sowohl Reaktivität, als auch zielgerichtetes Handeln möglich ist.
Im Besonderen beziehe ich mich auf den folgenden Seiten auf die innerhalb des Projektes "Kommunizierende Agenten" im Rahmen des SFB 360 entwickelte hybride Systemarchitektur.
Das Steuerungssystem besteht aus 2 Komponenten: dem reaktiven Basissystem und dem deliberativen höheren System. Das reaktive Basissystem bildet die zentrale Komponente in dem hybriden Architekturansatz. Es integriert die verschiedenen Sensordaten, wählt die der Situation angemessene Aktion aus und kontrolliert die Aktuatorik. [26].
Das deliberative System besteht aus der Sprachverarbeitungs- (NLP) und der Planungskomponente. Die NLP-Komponente verarbeitet natürlichsprachlich

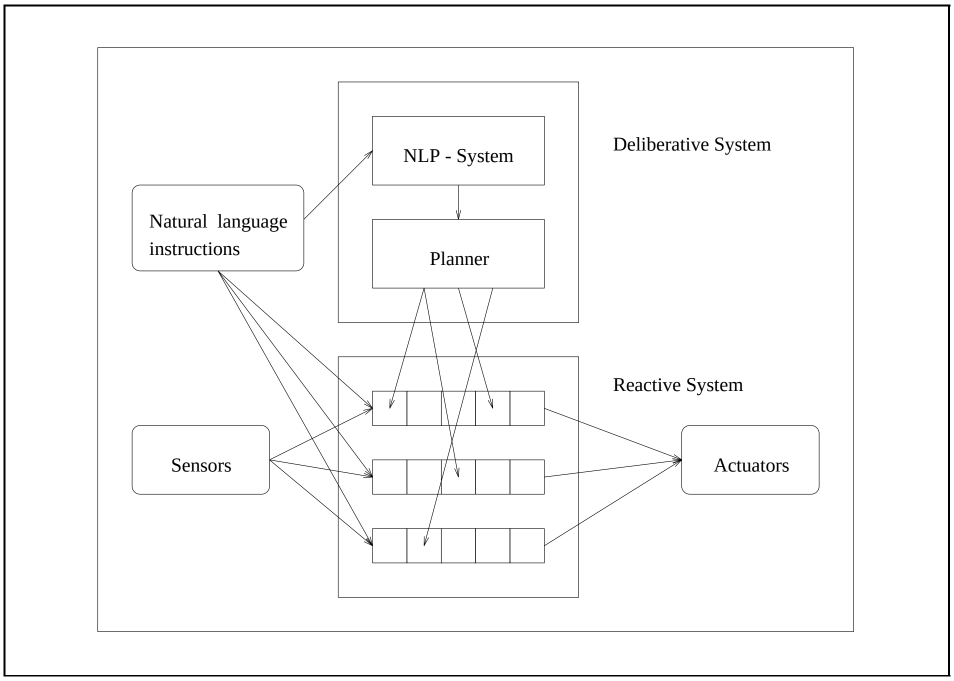

Abbildung 4.1: Blockbild des Steuerungssystems

formulierte Anweisungen. In einem ersten Schritt werden innerhalb der Planungskomponente *Aktionsschemata* verwendet ([45]). Aktionsschemata sind "Planschablonen", welche durch von der Instruktion gelieferte Informationen soweit wie möglich gefüllt werden. Dieses so spezifizierte Planfragment aktiviert anschließend das reaktive Basissystem. Dort werden, sofern nötig und möglich, fehlende Informationen mit von den Sensoren gelieferten Situationsdaten aufgefüllt.

Die Koppelung von reaktivem und deliberativem System erfolgt über eine definierte Schnittstelle. Diese Übergabebuffer speichern Information der Sensoren, Information der natürlichsprachlichen Intervention und Information, die durch die Aktionsschemata erzeugt wird.

Es existieren Schnittstellenparameter für Farbwerte, Objektbezeichnung und Aktionsbezeichnung. Von Sensoren gelieferte Farbwerte werden in dem entsprechenden Parameter eingetragen. In den gleichen Parametern wird aber auch der sprachlich spezifizierte Attributwert für die Farbe "Grün" eingetragen.

Im Gegensatz zu den Schnittstellenparametern werden die internen Systemsensoren des Basissystems zur Synchronisation der Einzelbehaviors genutzt.

Es werden also keine externen Sensordaten abgelegt, sondern ein Ausschnitt des aktuellen Aktivierungszustands der Behaviors repräsentiert (s.a. [27]).
Eine erfolgreiche Einbindung des durch das Lernverfahren aufgebauten abstrakten Behaviors in erweiterte behaviorbasierte Systeme erfordert eine Reihe von Modifikationen an der Schnittstelle zwischen der deliberativen und der reaktiven Komponente.
Das deliberative System verfügt in der vorliegenden Version über kein Metawissen bezüglich der Organisation des Basissystems. Als ungeordnete Menge sind über die Schnittstelle eine Reihe von Verhaltensweisen aktivierbar; deren interne Struktur ist dem deliberativen System unbekannt. Eine effiziente Handlungsplanung ist damit nur auf der Ebene der Grundverhaltensweisen möglich.
Werden im reaktiven Basissystem neue Handlungsfolgen als abstrakte Behaviors erlernt, so sind diese noch nicht für das deliberative System nutzbar. Die monodirektionale Schnittstelle muß entsprechend durchlässig für Informationen werden, die, aus dem Basissystem kommend, vom deliberativen System genutzt werden.
Neue Steuerungsbehaviors werden durch das Verfahren auf einem definierten Abstraktionsniveau in die Behaviorhierarchie eingeordnet. Anhand dieser Positionierung innerhalb der Hierarchie und der Anzahl zugrundeliegender Behaviors wird ihre Komplexität beschreibbar. Die Nutzbarmachung dieser Information für das deliberative System ist eine Grundvoraussetzung für die Verwendung erlernter Verhaltensweisen in Planungsprozessen der höheren Komponente. Für die Planung von effizienten Handlungssequenzen ist sowohl Wissen über die Anzahl der Teilschritte des abstrakten Behaviors, wie auch Start- und Endzustand und, zur Vermeidung von redundanten Handlungsschritten, die Erkennung relevanter Zwischenschritte notwendig. Diese zu identifizieren würde jedoch bedeuten, daß das deliberative System in der Lage sein muß, mögliche beschreibende Konzepte einer Handlungsfolge zu erkennen.
Neben einem erweiterten Informationsfluß vom reaktiven zum deliberativen System ist ebenfalls eine Erweiterung in die andere Richtung wünschenswert. Das deliberative System ist nur schwach an das reaktive System gekoppelt. Dies hat zur Folge, daß vom deliberativen System geplante Handlungsfolgen nur bei entsprechend häufiger Wiederholung vom Lernverfahren erkannt und als abstrakte Behaviors in das reaktive System aufgenommen werden. Um vom deliberativen System geplante Handlungsfolgen schnell als automatisierte Handlungsfolgen im Behaviorsystem einzugliedern, muß die Schnittstelle so erweitert werden, daß Handlungsfolgen als Spezifikation abstrakter Behaviors für das Behaviorsystem nutzbar sind. Damit nicht jede geplante Handlungsfolge automatisiert wird, sollte dies in Abstimmung mit den durch

das Lernverfahren identifizierten sinnvollen Handlungsfolgen erfolgen. Damit wäre ein Zusammenspiel von geplanten Handlungsfolgen mit vorgegebenen Zwischenzuständen und der erfahrungsbasierten Abstraktion von Handlungsfolgen möglich. Ein erweitertes behaviorbasiertes System ist damit sowohl zu einer planbasierten, wie auch erfahrungsbasierten Erweiterung seiner Handlungsmöglichkeiten fähig.

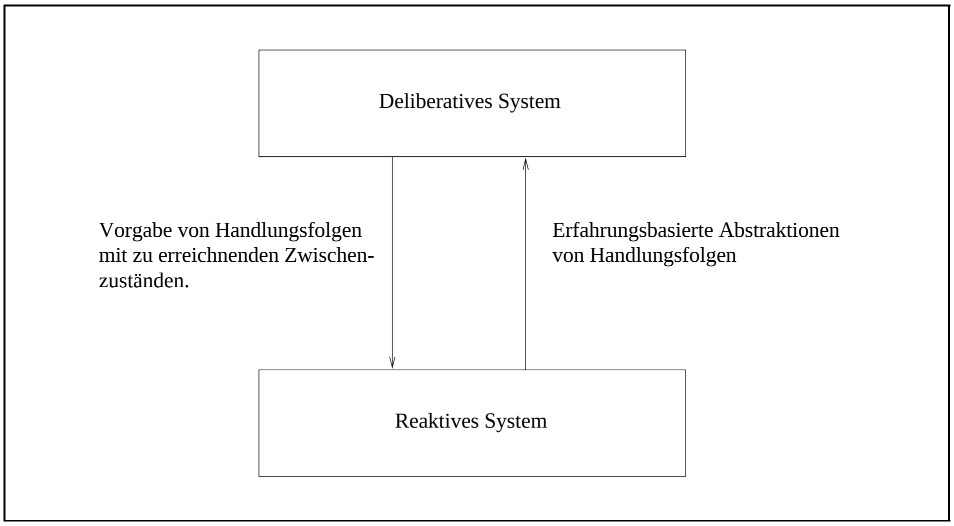

Abbildung 4.2: Datenfluß zwischen deliberativer und reaktiver Komponente

Zur Koppelung von deliberativen und reaktiven Komponenten ist es notwendig, die reaktive Komponente so zu konzipieren, daß vorgegebene Handlungsfolgen übernommen und deren Zwischenzustände durchlaufen werden können. Die deliberative Komponente muß in der Lage sein, erfahrungsbasierte Abstraktionen von Handlungsfolgen der reaktiven Komponente in den bestehenden Wissensbestand zu integrieren und für die weitere Handlungsplanung einzusetzen.

Literaturverzeichnis

[1] T. Almy. Xlisp-plus: Another object-oriented lisp. Technical report, sail.labs.tek.com, 1992.

[2] A. Asteroth. Tracking and grasping of moving objects: a behaviour-based approach. Technical report, Arbeitspapiere der GMD 0603, Sankt Augustin, 1991.

[3] N. I. Badler, B. L. Webber, J. Kalita, and J. Esakov. *Making Them Move*, chapter Animation from Instructions, Chapter 3, pages 51–93. Morgan Kaufmann, San Mateo, CA, 1991.

[4] O. Bartenstein and H. Inoue. Learning-assisted robot programming. In *Proc. Robotic Research - The fourth international symposium*, 1987.

[5] A. Barto, S. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. Technical report, TR 93-02, University of Massachusetts, Amherst, MA, 1993.

[6] A. Barto, R. Sutton, and A. C.W. *Learning and Computational Neuroscience*, chapter Learning and Sequential Decision Making, pages 539–602. MIT-Press, Cambridge, MA, 1990.

[7] S. Bennett. Reducing real world failures of approximate explanation based-rules. In *Proceedings of the 7th Int. Conf. on Machine Learning*, pages 226–234, 1990.

[8] R. Brooks. Fast, cheap and out of control. *Science*, 248:959–961, 1990.

[9] R. Brooks. A short description of the subsumption compiler. Technical report, MIT Press, Cambridge, MA, 1990.

[10] R. A. Brooks. Intelligence without reason. Technical report, MIT Artificial Intelligence Lab, Cambridge, MA, 1986.

[11] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2,No.1, March 1986.

[12] R. A. Brooks. *Autonomous mobile robots*, pages 343–363. AI in the 1980s and beyond, Grimson, W. E. L., Patil, R. S. (eds). MIT Press, 1987.

[13] B. Brunner, H. G., K. Landzettel, and J. Heindl. Multisensory shared autonomy und tele-sensor-programming - key issues in the space robot technology experiment rotex. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE/RSJ, 1993.

[14] D. Chapman. *Vision, Instruction and Action*. The MIT Press, Cambridge, Massachusetts, 1992.

[15] K. Chen. Smooth pack tracking through symbolic computations. Technical report, ISG 83-13, AI Lab, Dept. of CS, Univ. of Illinois at Urbana-Champaign, 1986.

[16] A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning reliable manipulation strategies without initial physical model. In *Proc. IEEE Robotics and Automation, Cincinatti*, pages 1224–1230, 1990.

[17] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning, 3*, pages 261–283, 1989.

[18] J. H. Connell. *Minimalist Mobile Robots*. Academic Press Inc., San Diego, CA, 1990.

[19] J. H. Connell and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, 1993.

[20] G. L. Drescher. Made-up minds. a constructivist approach to artificial intelligence. Technical report, MIT Press, Cambridge, MA, 1991.

[21] B. Dufay and J.-C. Latombe. An approach to automatic robot programming based on inductive learning. In *Proc. Robotic Research - The first international symposium*, 1987.

[22] G. M. Edelman. *Bright Air, Brilliant Fire. On the Matter of the Mind*. Basic Books, A Division of HarperCollins Publishers, Inc., 1992.

[23] G. A. Fink, N. Jungclaus, F. Kummert, H. Ritter, and G. Sagerer. A distributed system for integrated speech and image understanding. In *Proceedings of the International Symposium on Artificial Intelligence, Cancun, Mexico, Proceedings AAAI-92*, pages 117–126. MIT Press, 1996.

[24] G. D. Fischbach. Gehirn und Geist. *Spektrum der Wissenschaft, Ausgabe 11*, 1992.

[25] J. Foley and et. al. *Computer Graphics, 2nd Edition*. Addison-Wesley, Reading, Massachusetts, Nov. 1992.

[26] S. Förster. Ein instruierbares behavior-basiertes System. Technical report, Schriftenreihe des SFB 360, Universität Bielefeld, 1995.

[27] S. Förster. *Steuerbarkeit behaviorbasierter Systeme*. PhD thesis, Universität Bielefeld, 1996.

[28] S. Förster, H. Lobin, and K. Peters. Architekturkonzeption eines teilautonomen Monatgeroboters. Technical report, SFB 360, Situierte Künstliche Kommunikatoren ,Universität Bielefeld, 10 1995.

[29] K. Fu, R. Gonzalez, and C. Lee. *Robotis – Control, Sensing, Vision, and Intelligence*. McGraw-Hill, 1987.

[30] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic Institute and State University, 1991.

[31] D. O. Hebb. *Einführung in die moderne Psychologie*. Beltz-Studienbuch, 1973.

[32] R. Heise. Demonstration instead of programming: Focussing attention in robot task aquisition. Technical report, 89/360/22, Department of Computer Science, The University of Calgary, Canada, 1989.

[33] C. M. Kadie. Continuous conceptual set covering: Learning robot operators from examples. In *Proceedings 8th Workshop on Machine Learning, Evanston*, pages 615–619, 1991.

[34] L. P. Kaelbling. Goals as parallel program specification. *Proceedings of AAAI 1988*, I:60–65, 1988.

[35] L. P. Kaelbling. Foundations of learning in autonomous agents. *Robotics and autonomous systems, Vol 8, Nov. 1991*, 1991.

[36] J. Kreuziger and M. Hauser. The application of symbolic learning techniques in a robot manipulation system. Technical report, Bericht des SFB 314, Teilprojekt R, Universität Karlsruhe, Fakultät für Informatik, 1993.

[37] U. Kühnapfel. *Graphische Realzeitunterstützung für Fernhandlungsvorgänge in komplexen Arbeitsumgebungen im Rahmen eines Systems zur Steuerung, Simulation und Off-Line Programmierung* . PhD thesis, Kernforschungszentrum Karlsruhe, 1992.

[38] B. J. Kuipers. Commonsense knowledge of space : Learning from experience. In *Proceedings of the Sixth IJCAI, Stanford, CA*, 1979.

[39] J. E. Laird. Soar: An architecture for general intelligence. *Artificial Intelligence, 33(3)*, 1987.

[40] J. E. e. a. Laird. Correcting and extending domain knowledge using outside guidance. In *The 7th International Conference on ML, Austin*, pages 235–243, 1990.

[41] P. Langley, K. Thompson, W. Iba, J. Gennari, and J. Allen. An integrated cognitive architecture for autonomous agents. Technical report, 89/360/22, Department of Computer Science, The University of Calgary, Canada, 1989.

[42] P. Lawrence. *The Making of a Fly.* Blackwell Scientific Publications, Oxford, 1992.

[43] E. Lewis. A gene complex controlling segmentation in drosophila. *Nature 276*, pages 565–570, 1978.

[44] H. Lobin. Modellierung eingebetteter Kommunikatoren. *Arbeitsberichte Computerlinguistik 1-92, Fakultät für Linguistik und Literaturwissenschaft*, 1992.

[45] H. Lobin. Agentenorientierte Repräsentation von Aktionen für die natürlichsprachliche Steuerung. *DGfS/CL-93*, page 63, 1993.

[46] K. Lorenz. *Foundations of Ethology.* Springer Verlag, Berlin, 1981.

[47] K. Lorenz. *Denkwege.* R. Piper GmbH & Co KG, München, 1992.

[48] P. Maes. How to do the right thing. *Connection Science*, 1:291–323, 1989.

[49] P. Maes. Learning behavior networks from experience. In *Proceedings of IJCAI*, pages 991–997, 1989.

[50] P. Maes. A bottom-up mechanism for behavior selection in an artificial creature. In *Proceedings of the 1rst conference on adaptive behavior.* MIT Press, 1991.

[51] P. Maes. Behavior-based artificial intelligence. In *proceedings of the 2nd conference on Adaptive Behavior, MIT Press*, 1993.

[52] P. Maes. Modeling adaptive autonomous agents. *Artificial Life Journal*, 1(1/2), 1994.

[53] P. Maes and R. A. Brooks. Learning to coordinate behaviours. In *Proceedings of AAAI 1990*. Morgan Kauffman Publishers, San Mateo, California, 1990.

[54] C. Malcolm and T. Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly system. *Robotics and Autonomous Systems*, 6:123–144, 1990.

[55] M. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, MIT EECS PhD Thesis, 1994.

[56] D. McFarland. *Animal Behavior*. Longman, 1985.

[57] U. Mehlhaus. Concurrent simulation and control of robot tasks. In *27th Hawaii International Conference on System Sciences*, pages 553–562, Wailea, Hawaii, jan 1994.

[58] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning An Artificial Inteligence Approach, Vol I*. Morgan Kaufmann, Los Altos, CA, 1983.

[59] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning An Artificial Inteligence Approach, Vol II*. Morgan Kaufmann, Los Altos, CA, 1986.

[60] J.-T. Milde. Cyrano - Ein behavior-basierter künstlicher situierter Agent. *Arbeitsberichte Computerlinguistik 2-93. Universität Bielefeld, Fakultät für Linguistik und Literaturwissenschaft.*, 1993.

[61] J. Müller. *Verteilte künstliche Intelligenz. Methoden und Anwendungen*. BI Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993. HK620 V5K9I *BNR: 3053590.

[62] D. A. Norman. Twelve issues for cognitive science. *Cognitive Science*, 4:1–32, 1981.

[63] Nüsslein-Volhard, C. and Wieschaus, E. Mutations affecting segment number and polarity in drosophila. *Nature 287*, pages 795–801, 1980.

[64] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[65] C. Sammut and D. Hume. Observation and generalisation in a simulated robot world. In *Proceedings International Workshop on Machine Learning, Irvine*, pages 267–273, 1987.

[66] T. Sato and S. Hirai. A model enhanced intelligent and skillfull teleoperational robot system. In *Proceedings Robotics Research - The Fourth Int. Symposium*, pages 155–162, 1987.

[67] A. M. Segre. *Machine Learning of Robot Assembly Plans*. Kluwer Academic Publishers, 1988.

[68] M. Selfridge and A. Levas. Teaching robots by example. In *Proc. 13th Int. Symposium on Industrial Robots 7, Chicago*, pages 12–160–13–165, 1983.

[69] U. e. a. Siems. SimRobot - Ein Programm zur Simulation sensorbestückter Agenten in einer dreidimensionalen Umwelt , ZKW Bericht Nr. 1/94. Technical report, Zentrum für Kognitionswissenschaften, Universität Bremen, 1994.

[70] H. A. Simon and A. Newell. *Informationsverarbeitung in Computer und Mensch*, pages 112–145. Künstliche Intelligenz, Philosophische Probleme, Zimmerli, Walter und Wolf, Stefan (eds). Phillip Reclam jun. GmbH, 1994.

[71] S. P. Singh. Reinforcement learning with a hierarchy of abstract models. In *Proceedings AAAI-92*, pages 202–207. MIT Press, 1992.

[72] L. Steels. The PDL reference manual. Technical report, VUB AI LAb, Brussels-Belgium, 1992.

[73] S. Strippgen. *INSIGHT: Ein virtuelles Labor für Entwurf, Test und Analyse von behavior-basierten Agenten*. PhD thesis, Universität Bielefeld, 1996.

[74] R. Sutton. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings ML-90*, pages 216–224, 1990.

[75] R. Sutton and A. Barto. A temporal-difference model of classical conditioning. *Proceedings of the Ninth Conference of the Cognitive Science Society, Lawrence Erlbaum Associates*, 1987.

[76] R. S. Sutton. *Temporal Credit Assignment In Reinforcement Learning.* PhD thesis, Univerity of Massachusetts at Amherst, 1984.

[77] R. S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on ML, Morgan Kaufman*, June 1990.

[78] M. Tan. Csl: A cost-sensitive learning system for sensing and grasping objects. In *Proceedings IEEE Robotics and Automation, Sacramento*, pages 858–863, 1990.

[79] N. Tinbergen. *Instinktlehre : vergleichende Erforschung angeborenen Verhaltens.* Parey, Berlin, 1966.

[80] M. Torrance. Natural communication with robots. Master's thesis, Department of Electrical Engineering and Computer Science at MIT, 1994.

[81] R. Trieb and E. von Putkammer. The 3d7-Simulation Enviroment. A Tool for Autonomous Mobile Robot Development. In *MASCOT 1994, Durham, North Carolina*, 1994.

[82] A. Turing. *Computing machinery and intelligence*, chapter 1, pages 117–132. in Dotzler and Kittler, eds, Alan Turing - Intelligence Service. Brinkman und Bose, 1987.

[83] A. Turing. *Kann eine Maschine denken ?*, pages 39–78. Künstliche Intelligenz, Philosophische Probleme, Zimmerli, Walter und Wolf, Stefan (eds). Phillip Reclam jun. GmbH, 1994.

[84] T. Tyrell. *Computational Mechanisms for Action Selection.* PhD thesis, University of Edinburgh, 1993.

[85] P. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[86] C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, King's College, University of Cambridge, 1989.

[87] C. J. C. H. Watkins and P. Dayan. Technical note : Q-learning. *Machine Learning, special issue on reinforcement learning*, 8:278–292, May 1992.

[88] T. Wengerek. *Reinforcement-Lernen in der Robotik.* PhD thesis, Universität Bielefeld, 1996.

[89] T. Wengerek and H. Ritter. Reinforcemt learning and subtask. *Artificial Neural Networks 2, Elsevier Science Publishers B.V. (North Holland)*, 1992.

[90] T. Wengerek and H. Ritter. Application of q-learning in robot grasping tasks. *Artificial Neural Networks 3, Elsevier Science Publishers B.V. (North Holland)*, 1993.

[91] T. Zhao and M. Overmars. Forms library. a graphical user interface toolkit for x. Technical report, Department of Physics, University of Wisconsin-Miwaukee, 1995.

[92] Y. Zheng and L. Daneshmend. Learning error-recovery startegies in telerobotic systems. In *Proc. IEEE Robotics and Automation, Sacramento*, pages 252–259, 1991.