Entwurf adaptiver lernender Roboter

Mohamed Salah Hamdi

Fachbereich Informatik, Universität Hamburg

Entwurf adaptiver lernender Roboter

Entwurf adaptiver lernender Roboter

Mohamed Salah Hamdi

Dissertation

1999

Fachbereich Informatik, Universität Hamburg

Mohamed Salah Hamdi
FB Informatik, AB TIS
Universität Hamburg
D-22527 Hamburg

Vom Fachbereich Informatik der Universität Hamburg
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation

Prüfungsausschuß:
Prof. Dr.-Ing. Karl Kaiser (1. Gutachter)
Prof. Dr. Leonie Dreschler-Fischer (2. Gutachter)
Prof. Dr. Bernd Neumann (Vorsitzender)

Dekan des Fachbereichs: Prof. Dr. Horst Oberquelle
Tag der Disputation: 9.2.1999

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Arbeitsbereich Technische Informatik-Systeme (TIS) im Fachbereich Informatik der Universität Hamburg.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Karl Kaiser für die Anregung zu diesem Thema, für die wertvollen Ratschläge und für die Unterstützung und Förderung meiner Arbeit. Daß er mir die Arbeit in einem faszinierenden Forschungsgebiet in so großzügiger Weise ermöglichte, kann ich gar nicht hoch genug anrechnen.

Für das ebenfalls meiner Arbeit entgegengebrachte Interesse und für die Bereitschaft zur Übernahme des Koreferates danke ich Frau Prof. Dr. Leonie Dreschler-Fischer sehr herzlich.

Ich möchte auch ganz herzlich Herrn Dr. Bernd Kirsig sowie Herrn Dipl.-Ing. Dr. Werner Hansmann für die kritische Durchsicht der vorliegenden Arbeit danken.

Die allseitige Unterstützung durch die Mitarbeiter des Arbeitsbereichs TIS war eine wesentliche Grundlage für das Gelingen dieser Arbeit.

INHALTSVERZEICHNIS

Kapitel 1

EINLEITUNG

Autonome Agenten sind Systeme, die dynamische und unvorhersehbare Umgebungen bewohnen. Sie interpretieren Sensordaten, die Ereignisse in der Umwelt widerspiegeln, und führen Motorbefehle aus, die auf die Umwelt wirken und sie verändern. Was diese Systeme von anderen unterscheidet, ist ihre Fähigkeit, in komplexen Umgebungen effektiv zu agieren ohne ständig von Menschen geführt und gesteuert zu werden. Es gibt mehrere Arten von Umgebungen, die für Agenten interessant sind. Natürlich ist die physische reale Welt derart komplex, daß der Entwurf von Agenten, die in ihr erfolgreich arbeiten können, eine Herausforderung darstellt. Es gibt aber auch virtuelle Welten verschiedener Arten. Das Internet, zum Beispiel, wird heutzutage immer komplexer. Hier werden andere Formen von Agenten gebraucht, die nützliche Aufgaben in solcher Umgebung verrichten können. Fortschritte in den Gebieten der Computergraphik und der Simulationstechnologie haben ebenfalls dazu geführt, daß komplexe simulierte Welten und Agenten, die ein realistisches Verhalten aufweisen, konstruiert werden können.

Agenten können, abhängig von der Art der Umgebung, mehrere Formen annehmen. Agenten, die die physische Welt bewohnen, sind normalerweise Roboter. Agenten, die mit der realen Software (z.B. Betriebssystem eines PCs, Internet, usw.) interagieren, werden oft Softbots genannt. Software-Agenten, die Menschen beim Treffen von komplexen Entscheidungen helfen oder bei der Erledigung von anderen Wissensverarbeitungsaufgaben unterstützen, werden mit Expert-Assistenten bezeichnet. Diese Agenten werden oft mit anderer konventioneller Software (wie z.B. Datenbanksysteme) oder mit herkömmlichen Kontrollsystemen gekoppelt. Agenten in simulierten Umgebungen, wie computeranimierte Umgebungen oder Video-Spiele, werden synthetische Agenten ge-

nannt.

Agenten können auch viele verschiedene Rollen spielen. Roboter können für Erkundungsaufgaben und Räumarbeiten in gefährlichen oder schwer zugänglichen Umgebungen oder für Wartungsarbeiten in unserem alltäglichen Umfeld und nicht zuletzt in der Automatisierungstechnik eingesetzt werden. Softbots können als persönliche Assistenten, die auf dem eigenen Arbeitsplatzrechner des einzelnen Benutzers laufen, oder als öffentliche Informationsvermittler im Netzwerk dienen. Expert-Assistenten können in der medizinischen Überwachung, in der computerunterstützten Fertigung oder in der Verwaltung des Luftverkehrs eingesetzt werden. Synthetische Agenten, die eher Charakter-Eigenschaften wie Glaubwürdigkeit und Persönlichkeit als tiefe Intelligenz hervorheben, können in interaktiven Systemen für Unterhaltung, Kunst, Spiele, Bildung oder Verbraucher-Software eine bedeutende Rolle spielen.

Der Bedarf an solchen ,,intelligenten" und autonomen Systemen wurde in den letzten Jahren erkannt, und es wurden viele Anstrengungen unternommen, solche Systeme für eine Vielzahl von Aufgaben zu konstruieren. In diesen Arbeiten hat sich aber gezeigt, daß die Programmierung solcher ,,intelligenten" Systeme weitaus problematischer ist als die konventioneller Aufgabenstellungen. Konventionelle Steuerungssysteme sind meistens so programmiert, daß sie starr eine vordefinierte Aufgabe in einer vordefinierten Umgebung durchführen. Diese Starrheit ist dadurch verursacht, daß die Steuereinheiten mit Hilfe von vorgefaßten Modellen des Agenten und seiner Umwelt entworfen werden. Um diese Modelle handlich zu halten, werden oft viele Vereinfachungen gemacht und somit viele Details vernachlässigt. Das Ergebnis ist meistens eine unflexible und nicht zufriedenstellende Lösung. Um diese Nachteile zu vermeiden, müssen für autonome Systeme neue Programmier-Paradigmen entwickelt werden. Intelligente Systeme müssen in der Lage sein, auf ständig wechselnde Bedingungen in ihrer Umwelt zu reagieren und können daher im allgemeinen nicht durch einen Satz fester Instruktionen die von ihnen geforderten Leistungen erbringen. Ganz im Gegenteil, solche Systeme müssen Aktionen je nach dem aktuellen Zustand ihrer Umwelt auswählen und unter Einhaltung der zeitlichen Bedingungen ausführen können. Sie müssen außerdem ihre augenblicklichen Verhaltensweisen revidieren können, wenn unvorhergesehene Ereignisse eintreten.

Um beispielsweise einen mobilen Roboter zu programmieren, fällt es sehr schwer, ,,All-

tagswissen" in die Welt des Roboters mit seinen spezifischen Sensoren und Aktoren zu übersetzen. Die Diskrepanz zwischen Mensch und Maschine ist enorm. Menschen und biologische Systeme im allgemeinen lernen dauernd, wie sie auf die Situation, in der sie sich befinden (was auch immer sie sein mag), reagieren und wie sie ihre Umgebung wirkungsvoll beeinflussen. Menschen und Roboter unterscheiden sich auch bei der Interpretation von Sensordaten sehr. Während Menschen bei der Interpretation von dem, was sie hören und sehen, sehr kompetent sind, sind Computer (Roboter) hingegen auf diesem Gebiet viel weniger erfolgreich. Die Robotik-Forschung hat gezeigt, daß die meisten der Aufgaben, die für Menschen scheinbar einfach sind, wie z.B. in einem Raum umherwandern und Hindernisse vermeiden, hohe Ansprüche an heutige Robotersysteme stellen. Anforderungen an Robustheit und Anpassungsfähigkeit bei flexibler Wechselwirkung mit der Umgebung komplizieren solche Aufgaben so sehr, daß eine Spezifikation aller Steuerungsdetails kaum mehr möglich ist. Der Grund dafür sind unter anderem Probleme wie das Fehlen notwendiger Informationen zum Zeitpunkt des Entwurfs des Roboters, die Unvorhersehbarkeit der Dynamik der Umgebung, sowie das unvermeidbare ,,Rauschen der Sensoren und Aktoren" des Roboters.

Um diese Probleme zu lösen, müssen Systeme geschaffen werden, die fähig sind, sich selbständig an Veränderungen der Umwelt anzupassen und aus ihren Erfahrungen zu lernen. Solche Systeme würden offensichtlich für eine adäquate Leistung und hohe Überlebensfähigkeit sorgen und gleichzeitig die Arbeit des Designers erleichtern und ihn von der schwierigen Aufgabe des detaillierten Entwurfs und des sorgfältigen Tunings des Kontrollsystems erheblich entlasten. Systeme, die aufgrund ihrer bisherigen Geschichte zu Veränderungen fähig sind, die zu einer Verbesserung ihres zukünftigen Verhaltens führen, werden adaptive Systeme genannt. Bei dem Ziel, Adaptivität in einem Robotik-System zu realisieren, hat man als Inspirationsquelle die Anpassungsfähigkeit vor Augen, die uns in der eigenen Alltagserfahrung oder in der Natur begegnet. Von der Evolution kennen wir die sich über Jahrmillionen erstreckende Anpassung an immer neue Lebensumstände. Oder betrachten wir die Bewältigung einer neuartigen Aufgabe: Man probiert verschiedene Ansätze aus und überprüft ihre Brauchbarkeit; die weitere Suche nach einer besseren Lösung richtet sich nach dem bisherigen Erfolg oder Mißerfolg, sie wird dem bisherigen Verlauf der Lösungssuche angepaßt. Ein anderes Beispiel: Wenn man sich aus vorliegenden Daten eine Vorstellung über die zukünftige Entwicklung eines

technischen oder gesellschaftlichen Prozesses gebildet hat, wird man diese Vorstellung beim Bekanntwerden neuer Daten modifizieren und an das nunmehr vorliegende Datenmaterial anpassen. Und daß schließlich das Steuern eines Autos eine anpassungsintensive Tätigkeit ist, bedarf wohl keiner weiteren Erläuterung.

Menschen sind nicht die einzigen natürlichen Agenten, die ein interessantes, adaptives und autonomes Verhalten aufweisen. Sie sind auch vielleicht nicht besonders geeignet, um zum jetzigen Zeitpunkt und mit unserem jetzigen Wissensstand untersucht zu werden. Das Verhalten von Tieren ist dagegen einfacher. Tiere sind fähig, ihre begrenzten Verhaltensvorräte an die momentane Situation der realen Welt autonom anzupassen und stellen somit eine reichhaltige Quelle für potentielle Einsichten für Kontrollstrukturen dar.

Die Entwicklung von adaptiven Systemen, die sich durch ein hohes Maß an Flexibilität gegenüber Unbekanntem und Unvorhersehbarem auszeichnen, wurde in den letzten Jahren zu einem zentralen Forschungsschwerpunkt auf dem Gebiet der Robotik. Diese Dissertation konzentriert sich auf die Entwicklung einer adaptiven Steuerungsarchitektur für autonome mobile Roboter, also auf eine Architektur, die es einem mobilen Roboter erlaubt, während der Interaktion mit der Umgebung Wissen zu sammeln und anzuwenden, um sein Verhalten nachträglich anzupassen. Die Architektur beruht auf der Verknüpfung von Methoden des maschinellen Lernens mit verhaltensbasierten Ansätzen. Verhaltensbasierte Ansätze orientieren sich an Vorbildern in der Natur. Es wird dabei auf Ergebnisse der Verhaltensforschung und die dort gefundenen Modelle zurückgegriffen und versucht, Robotik-Systeme nach Art natürlicher sensomotorisch gesteuerter Systeme zu bauen, die mit eigenen Grundfähigkeiten ausgestattet in der Lage sind, ihre Funktionalität im Einsatz erfahrungsgestützt zu erweitern und zu verbessern.

Im ersten einführenden Teil dieser Arbeit werden zuerst allgemeine Definitionen und Begriffe sowie Wissensgebiete, die für autonome mobile Systeme wichtig sind, erläutert (Kap.2). Danach werden die bisherigen Ansätze zur Steuerung von mobilen Robotern verglichen und klassifiziert (Kap.3). Kapitel 4 gibt einen Überblick über die Methoden des maschinellen Lernens, die im zweiten Teil dieser Arbeit angewendet werden. Im zweiten Teil der Arbeit (Kap. 5 bis Kap. 8) wird schrittweise eine adaptive, verhaltensbasierte Steuerungsarchitekur für autonome mobile Roboter entwickelt. Es werden die zugrunde

liegenden Ideen und Algorithmen beschrieben und ihre Zweckmäßigkeit mit Hilfe von Simulationsergebnissen erprobt. Die Arbeit wird mit einem zusammenfassenden Kapitel (Kap. 9) abgeschlossen.

Kapitel 2

BEGRIFFE UND DEFINITIONEN

Der Begriff *Roboter* stammt aus der Übersetzung des 1920 von dem Dramatiker Karel Capek in der Tschechei geschriebenen Theaterstücks ,,Rossum's Universal Robots". Die Bezeichnung ,,Roboter" ist dabei vom slawischen ,,robota", was etwa ,,schwer arbeiten" bedeutet, abgeleitet. Nach der Namensgebung dauerte es 40 Jahre bis mit Beginn der industriellen Nutzung der Mikroelektronik die systematische Technologieentwicklung für Industrieroboter begann. Seit dieser Zeit setzte eine breite Entwicklung und Anwendung von Softwarewerkzeugen zur Unterstützung von Roboteranwendungen ein und es wurden viele technologische Fortschritte und Erkenntnisse erzielt. Heute stellt das Wissensgebiet Robotik ein interdisziplinäres Fachgebiet dar, in dem Beiträge aus Ingenieurwissenschaften, Naturwissenschaften und Arbeitswissenschaften zusammenfließen.

Die Methoden und Werkzeuge der modernen Informatik werden zur Einsatzplanung, zur Programmierung von Roboteranwendungen und zur Entwicklung effizienter Echtzeitsteuerungen angewandt. Die Modellierung kognitiver Fähigkeiten sowie ihre Integration in Robotersysteme verleiht Robotern zunehmend Intelligenz und Autonomie.

Die Entwicklung von autonomen mobilen Robotern bekam in den letzten Jahren die Aufmerksamkeit vieler Forscher. Mit fortschreitender Erforschung und Entwicklung kognitiver Systeme, automatisch planender und maschinell lernender Systeme kommt man diesem Ziel immer näher.

In diesem Kapitel wird ein Überblick über die verschiedenen Disziplinen gegeben, die sich mit der Robotik und Bildung von intelligenten Agenten beschäftigen. Einige wichtige Begriffe, die in den letzten Jahren entstanden sind, werden erläutert.

2.1 Was sind intelligente autonome Roboter

In vielen Forschungsgebieten gewann die Frage an Interesse, wie ein Roboter in einer realen, teilweise unbekannten, dynamischen und ungenauen Umwelt eine Aufgabe selbständig ausführen und auf unvorhergesehene Ereignisse reagieren kann. Die größte Herausforderung bei dieser Fragestellung liegt in der sehr engen Interaktion zwischen System und Umwelt und setzt somit tiefes Verständnis der Beziehung zwischen Roboter und Umwelt voraus. Zur Lösung dieser Probleme werden intelligente autonome Roboter oder Agenten untersucht. Im folgenden werden die Begriffe *Roboter*, *Autonomie* und *Intelligenz* näher betrachtet.

2.1.1 Roboter (Agent)

Ein Roboter ist ein physisches Objekt und unterliegt damit den Gesetzen der Physik. Was einen Roboter von anderen Objekten unterscheidet ist die Tatsache, daß ein Roboter sein Schicksal in Grenzen steuern kann. Ein Stein ist auf einfache Art und Weise den Gesetzen der Physik (zum Beispiel der Schwerkraft) unterworfen. Ein Stein kann nicht entscheiden, ob er nicht fällt oder ob er sich etwas nach vorne bewegt. Ein einfaches Tier (auch ein einzelliges) hingegen, kann sich in Richtung von Nahrung bewegen oder gefährliche Gebiete vermeiden. Die Unabhängigkeit von physischen Gesetzen ist natürlich relativ. Eine Person, die von einem Auto aufgefahren wird, kann recht wenig dagegen tun.

Eine Anforderung an diese relative Unabhängigkeit ist eine gewisse *Automatik*. Ein Roboter muß Möglichkeiten haben, Aspekte der Umgebung wahrzunehmen und die Umgebung zu beeinflussen (z. B. seine eigene Position zu ändern oder Objekte zu manipulieren). All dies muß mit Hilfe von automatischen Mechanismen stattfinden, Mechanismen also, die die Intervention von anderen nicht erfordern. Dies hat zur Folge, daß ein Roboter in der Lage sein muß zu entscheiden, was er tun soll und wann er das tut. Die Automatik ist eine Eigenschaft, die heute in vielen Maschinen zu finden ist, zum Beispiel in Systemen, die die Zentralheizung in einem Haus steuern oder in einem Flugzeug, das im automatischen Betrieb fliegt.

2.1.2 Autonomie

Damit ein Agent (Roboter) in der realen Welt funktionsfähig sein kann, muß er zusätzlich zur Automatik der Anforderung genügen, *autonom* zu sein. Die zentrale Idee in dem Konzept von Autonomie ist durch die Etymologie des Wortes *autos* (selbst) und *nomos* (Regel oder Gesetz) gegeben. Griechische Städte, deren Bürger ihre eigenen Gesetze gemacht haben und nicht nach den Gesetzen einer externen Regierungsmacht leben wollten, wurden als autonom bezeichnet. Es ist nützlich, Autonomie mit dem Konzept der Automatik zu vergleichen. Die Bedeutung des Wortes ,,Automatik" kommt von der Etymologie des Wortes *Kybernetik* (siehe unten), das von dem griechischen Wort für *Selbst-Steuerung* abgeleitet ist. In anderen Worten, automatische Systeme sind selbstregelnd, machen aber nicht die Gesetze, die von ihren Regelungsaktivitäten erfüllt werden müssen. Diese Gesetze werden ihnen zur Verfügung gestellt oder in sie eingebaut. Automatische Systeme steuern sich selbst entlang eines gegebenen Pfades. Während ihrer Arbeit korrigieren sie und gleichen die Auswirkungen von externen Störungen aus. Autonome Systeme andererseits sind Systeme, die für sich selbst die Gesetze und Strategien entwickeln, nach denen sie ihre Verhaltensweisen regulieren: Sie sind sowohl *selbstverwaltend* als auch *selbstregulierend*. Sie sind in der Lage, sowohl die Pfade zu bestimmen, denen sie folgen, als auch sich entlang dieser Pfade zu bewegen.

Damit ein System autonom sein kann, muß es zuerst automatisch sein. Dies bedeutet, daß das System in der Lage sein muß, in seiner Umgebung zu agieren, diese Umgebung wahrzunehmen und sie so zu beeinflussen, daß Aufgaben, die wichtig erscheinen, erledigt werden können. Autonomie aber geht über die Automatik hinaus, weil dabei zusätzlich angenommen wird, daß die Basis für Selbst-Steuerung aus der eigenen Fähigkeit des Systems entsteht, seine Verhaltensprinzipien zu bilden und anzupassen. Ferner, ist es so, daß der Prozeß des Aufbaus und der Anpassung der eigenen Kompetenz eine Sache ist, die während des Agierens des Systems in seiner Umgebung stattfindet. Das System hat nicht die Zeit, eine große Menge von Beispielen zu studieren oder tief darüber nachzudenken, wie es unvorhersehbare Situationen bewältigen kann. Stattdessen muß es kontinuierlich agieren und reagieren, um zu überleben.

Eine andere Art, Autonomie zu charakterisieren, wäre es, die Sicht des Betrachters zu berücksichtigen. Der Verhaltensforscher David McFarland [100] hebt hervor, daß ein

automatisches System ein System ist, für das man das Verhalten vollständig vorhersagen kann, sobald man seine interne Basis für die Entscheidungsfindung kennt. Ein autonomes System andererseits ist ein System, das sich selbst entscheiden kann. Selbst für den ursprünglichen Designer eines Systems ist es nicht klar, wie es in der Zukunft reagieren wird, weil es so aufgebaut war, daß Reaktionen sich entwickeln und ändern, um neue Situationen zu bewältigen.

Ein autonomer Agent ist somit ein selbstverwaltendes, selbstregulierendes System, das sich selbständig in seiner Umgebung zurechtfinden kann. Es gibt keine versteckten Drähte oder Funkstrecken, die ihn mit einem Bediener verbinden.

2.1.3 Intelligenz

Offenbar wird die Autonomie einer Einheit, einer Maschine oder eines Lebewesens durch dessen Intelligenz bestimmt. Man kann erwarten, daß ein hoher Intelligenzlevel zu einem hohen Autonomielevel führt. Was ist aber *Intelligenz*?

Dies ist eine Frage, die Philosophen hunderte von Jahren beschäftigt hat und für die es noch keine allgemeine Übereinkunft für eine Definition gibt. Bei dem Versuch, Intelligenz zu definieren, besteht die Gefahr, sich in philosophischen Fragen tief zu verwickeln und zu keinem Ergebnis zu kommen. Eine mehr informelle Definition von Intelligenz wäre, sie als etwas zu bezeichnen, was den Menschen als Menschen auszeichnet.

Intelligenz erweitert die Unabhängigkeit von Robotern von den physischen Kräften. Die Roboter bekommen zum Beispiel dadurch die Fähigkeit, ein Ereignis vorherzusagen bevor es stattfindet und aufgrund dieser Vorhersage zu agieren oder vor dem Agieren zu planen. Sie können dadurch auch die Fähigkeit erhalten, ein Ziel zu verfolgen, selbst wenn es keinen direkten sensorischen Kontakt mit diesem Ziel gibt. Es ist üblich in der Literatur über Intelligenz zwischen zwei verschiedenen Arten von Geschicklichkeit (skill) zu unterscheiden: Aktionsbasierte Geschicklichkeit und intellektbasierte Geschicklichkeit. Aktionsbasierte Geschicklichkeit zeigt sich bei sensomotorischen Aktivitäten wie zum Beispiel in einem Raum herumzulaufen, Auto zu fahren, oder Klavier zu spielen. Intellektbasierte Geschicklichkeit wird gebraucht bei der Implementierung von Computerprogrammen, beim Spielen von Schach oder bei der Formulierung von mathematischen

Beweisen. Diese Arten von Geschicklichkeit werden oft durch folgende Eigenschaften charakterisiert:

- *Wahrnehmung:* aktionsbasierte Geschicklichkeit basiert auf Informationen, die aus den physischen Gegebenheiten abgeleitet sind. Intellektbasierte Geschicklichkeit basiert auf Abstraktionen, ausgedrückt als Beschreibungen. Sie kann in Abwesenheit von physischen Gegebenheiten operieren.
- *Aktionsabhängigkeit:* aktionsbasierte Geschicklichkeit entfaltet sich in physischer Leistung. Obwohl sie grundsätzlich durch Sprache explizit gemacht werden kann, bleibt sie typischerweise ohne Erklärung (implizit in der Aktion). Intellektbasierte Geschicklichkeit wird aufgrund von verbaler Information oder aufgrund von Abstraktion während der Ausführung von Aktionen entwickelt. Sie ist bereits explizit oder kann leicht explizit gemacht werden.
- *Kontextabhängigkeit:* aktionsbasierte Geschicklichkeit hat eine Bedeutung nur innerhalb des Kontextes, in dem die dazugehörigen physischen Aktivitäten auftreten können. Intellektbasierte Geschicklichkeit ist kontextunabhängig, weil sie auf der Basis von Abstraktionen funktioniert. Die Verbindung zwischen Beschreibung und Geschicklichkeit muß explizit hergestellt werden.
- *Verkörperung:* bei aktionsbasierter Geschicklichkeit ist es der einzelne Körper, der in die Situation eindringt, und es sind seine Aktionen, die die notwendige Kompetenz entfalten. Intellektbasierte Geschicklichkeit basiert auf entkörperlichtem Wissen, das auf einfache Art und Weise gemeinsam benutzt werden kann (z.B. dadurch, daß man es niederschreibt). Es ist in vieler Hinsicht kulturelles Wissen, das durch Bücher und explizite Erziehung übermittelt wird.

2.2 Intelligente Robotersysteme: Ein interdisziplinäres Gebiet

Obwohl praktische Erfahrungen mit der Entwicklung von intelligenten autonomen Robotersystemen (oder besser Systemen mit einigen Elementen von Autonomie) erst seit einigen wenigen Jahren vorliegen, befinden sich diese Systeme jetzt in raschem Wachstum. Es gibt heute eine Anzahl von ehrgeizigen Entwicklungsprogrammen an Universitäten, in der Industrie und in vielen Forschungszentren. Diese Programme konzentrieren die

Bemühungen der Wissenschaftler und Ingenieure nicht nur auf das Problem der Autonomie und Intelligenz von Robotern, sondern auch auf die zukünftigen Stadien der Forschung und Entwicklung wie zum Beispiel die Koordination der verhältnismäßig autonomen Teilsysteme eines unbemannten Fertigungssystems, die Kooperation innerhalb mehrerer intelligenter Maschinen, die eine Aufgabe gemeinsam lösen sollen, usw. Autonomie wird heute immer mehr als zusätzliches Hilfsmittel für die Lösung von technischen Problemen in einer breiten Vielzahl von Bereichen eingesetzt.

Dieser schnelle Fortschritt wird durch eine direkte praxisbezogene Nachfrage angetrieben, die aus einer Vielzahl von Anwendungsgebieten kommt. Geräte für den Material- und Werkstücktransport in einer Fertigungsanlage, zum Beispiel, sind aufgrund der vorgegebenen festen Leitstrecken (z.B. Induktionsschleifen) in ihrer Kosteneffektivität eingeschränkt. Die meisten der flexiblen Fertigungssysteme sind von der autonomen Mobilität als Hilfsmittel für die Erhöhung der Produktivität abhängig. Mit autonomen mobilen Systemen wäre es möglich, Fertigungsanlagen mit sehr großer Flexibilität zu konzipieren. Es könnte jede Kombination von Maschinen nach einem virtuellen Fertigungskonzept geschaffen werden, um ein gewünschtes Produkt herzustellen. Auch viele andere Anwendungen, in denen unbemannte Arbeit notwendig ist (z.B. Bergbau, Überwachung von Unterwasser-Erdölleitungen, Arbeiten in kerntechnischen Anlagen oder Einsatz in der Raumfahrt), haben eine starke Nachfrage nach autonomen und teilautonomen Systemen verursacht. Solche Systeme könnten den Menschen in einer Vielzahl von unstrukturierten Situationen, in denen sowohl die Bewegung als auch die Verrichtung der Arbeit nicht vollständig im voraus vorgeschrieben werden kann, ersetzen oder ergänzen.

Der stärkste Antrieb jedoch für die Durchführung von wissenschaftlicher und technischer Arbeit in dem Bereich von intelligenten autonomen Systemen ist gegeben durch die strukturellen Veränderungen in der Wissenschaft und in der Technik (neue Forschungserkenntnisse, Fortschritte in der Computertechnik) und nicht zuletzt durch die Neugier und Bestrebung menschliche Intelligenz nachzuahmen. Es folgt die Aufzählung einiger Arbeitsgebiete, die sich mit der Entwicklung intelligenter Robotersysteme befaßt haben sowie einige historische Bemerkungen.

2.2.1 Kontrolltheorie

Gegenstand der Kontrolltheorie ist die gezielte Beeinflussung (Steuerung) sogenannter ,,dynamischer Systeme", die in vielfältiger Form bei der Modellierung praktischer Probleme aus Natur-, Technik-, Wirtschafts- und Gesellschaftswissenschaften entstehen. Betrachtet werden Fragen der Steuerbarkeit, Stabilisierbarkeit, Rekonstruierbarkeit und der optimalen Steuerung bei linearen und nichtlinearen Systemen. Kontrolltheorie wird auch oft als eine ,,konzeptorientierte" Disziplin verstanden, in der es darum geht, einen theoretischen Rahmen durch Abstrahierung des Konzepts der ,,Kontrolle" (Steuerung) zu konstruieren.

Der wichtigste Fortschritt in der Kontrolltheorie erfolgte durch die Entwicklung des Prinzips der *Rückkopplung* (feedback control), das in den vierziger Jahren zur Reife gekommen ist. Allgemein ist ein Rückkopplungssystem ein System, bei dem die Ursache gerade jene Wirkung erzeugt, die sie selbst zur Ursache hat (Prinzip der Schließung). In einem System, in dem eine Transformation stattfindet, gibt es Eingabe- und Ausgabedaten. Die Eingabedaten sind das Ergebnis der Wirkung der Umgebung auf das System und die Ausgabedaten stellen die Wirkung des Systems auf die Umgebung dar. In jedem Rückführungsschritt wird Information über das Ergebnis einer Transformation oder Aktion zur Eingabeseite des Systems als Eingabedaten zurückgeführt. Wenn diese neuen Daten die Transformation in der gleichen Richtung wie die vorhergehenden Resultate vereinfachen und beschleunigen, dann stellen sie positives Feedback dar (ihre Wirkung ist kumulativ). Wenn die neuen Daten Ergebnisse produzieren, die früheren Ergebnissen widersprechen, dann stellen sie negatives Feedback dar (ihre Wirkung stabilisiert das System). Bei kontinuierlichen Systemen erfolgt die Rückkopplung, wie schon der Name sagt, stetig, bei diskreten Systemen in diskreter zeitlicher Abfolge. Ersteres führt auf Differential-, letzteres auf Differenzengleichungen. Mathematisch bedeutet Rückkopplung auch Iteration bzw. Rekursion.

In den sechziger Jahren wurde die Kontrolltheorie durch die Benutzung von Zustandsraummethoden (state space methods) statt Transferfunktionen (transfer functions) geprägt. Dies hat zur raschen Entwicklung mehrerer Prinzipien und Methoden wie zum Beispiel optimaler Regler (optimal regulator), Kalman-Filter, Prinzip des internen Modells, sowie vieler Erweiterungen davon geführt. Optimierung wurde zu einem der meist

diskutierten Themen in der Kontrolltheorie. Das Maximumprinzip von Pontryagin [17] und die Dynamische Programmierung von Bellmann [14] wurden entwickelt.

Kontrolltheorie hat sich als funktionsfähiger und entwicklungsfähiger als einige andere allgemeine Ansätze erwiesen, hauptsächlich weil die Gemeinschaft der Spezialisten in Kontrolltheorie sich immer mit der etablierten mathematischen Apparatur von Differential- und Integralrechnung (später erweitert durch Methoden der linearen Algebra und andere Bereiche der Mathematik) und mit einem konkreten Kunden, der diese mathematische Apparatur verstanden (und/oder an sie geglaubt) hat, verbunden fühlte. Dennoch gab es immer eine gewisse Unzufriedenheit mit den Ergebnissen und der Orientierung der Kontrolltheorie.

Für einen langen Zeitraum hatten Spezialisten in dem Gebiet der Kontrolltheorie ein allgemeines Gefühl, daß es einen wichtigen Bedarf gibt, den Unterschied zwischen zwei Dingen zu formulieren: der Kontrolltheorie (konventionelle Kontrolltheorie) oder Theorie der automatischen Kontrolle auf der einen Seite, und einer zweiten, schwer bestimmbaren Instanz, die immer ringsherum ist und sich drastisch von der herkömmlichen Kontrolltheorie unterscheidet, auf der anderen Seite. Diese andere Instanz neigt dazu, unkonventionelle Methoden zu benutzen und aus dem Bereich der Integral- und Differentialrechnung zu einer Anzahl von anderen Bereichen, die von Automatentheorie bis Linguistik reichen, zu expandieren.

Es wurde also klar, daß ein Paradigma der konventionellen Kontrolltheorie existiert, welches Gegenstände angesichts eines bestimmten Teils ihres Verhaltens behandelt, das durch die Apparatur von Integral- und Differentialrechnung, die Theorie von Differentialgleichungen, lineare Algebra und Vektorrechnung dargestellt werden kann. Dies ist eine Möglichkeit für die Formulierung des Problems in der konventionellen Kontrolltheorie, in der das System als mathematische Struktur verstanden wird, die durch eine gegebene Menge von Zeitpunkten, eine Menge von Zuständen, eine Menge von Eingabewerten, eine nicht leere Menge von Eingabefunktionen (oder Befehlssequenz), eine Menge von Ausgabewerten und eine Menge von Ausgabefunktionen definiert ist. Es wird keine Kommunikation mit dem Benutzer als Teil der zukünftigen Kontrolloperation und keine Überlappung zwischen den Mengen angenommen. Das System kennt den Begriff „Ziel einer Operation" nicht und sieht ihn deshalb nicht als Teil der Struktur vor. Die Menge

der Definitionen und Axiome wird üblicherweise durch die Forderung von Stationarität, Linearität und Glätte ergänzt. Eigenschaften der Zustandsübergangsfunktion führen zu der bekannten formalen Darstellung, die bei der linearen Systemtheorie verwendet wird.

In den siebziger und achtziger Jahren hat ein neues Paradigma für die Lösung des Kontrollproblems angefangen sich zu etablieren, und es entwickelte sich, neben den konventionellen Kontrollsystemen, eine Menge neuer Kontrollsysteme: Systeme mit Erkennungskomponenten (recognition) in der Kontrollschleife [46], Systeme mit Lern- und Selbstorganisationselementen (self-organization) [144], und Systeme, in denen eine Darstellung in konventioneller mathematischer Form manchmal nicht möglich ist [54]. All diese Systeme brauchten eine Schleife (loop), die *intelligente* Mittel zur Verfügung stellen sollte, um eine Vielzahl komplizierter Probleme zu lösen. Die Kontrolle eines Prozesses wurde definiert als die Lenkung des Prozesses, um ein vorgegebenes Ziel zu erreichen [144]. Die Definition für die Kontroll-Regel (control law) in einem intelligenten System wurde somit um den Begriff ,,Ziel" erweitert und war unabhängig vom Modell des Systems und von der Kontrollmethode. Die Tendenz zur Benutzung und Integration von Methoden der künstlichen Intelligenz (siehe unten) wurde immer deutlicher und der Begriff ,,Intelligente Kontrolle" (intelligent control) hat immer mehr an Akzeptanz gewonnen.

2.2.2 Kybernetik

In den vierziger und fünfziger Jahren gab es eine andere Disziplin, die sich auch für intelligente Systeme und für das Verstehen menschlicher Intelligenz interessiert hat. Sie wurde unter dem Namen *Kybernetik* [174, 175] bekannt und entwickelte sich parallel zur Kontrolltheorie. Kybernetik ist das Studium der Mathematik von Maschinen. Die Betonung liegt dabei nicht auf den funktionellen Komponenten und wie sie verbunden sind und nicht auf dem Problem, was eine einzelne Maschine in einer bestimmten Situation tun kann, sondern auf allen möglichen Verhalten, die von einer einzelnen Maschine produziert werden können. Es gab eine starke Neigung, eine Maschine durch ihre Eingaben und Ausgaben zu charakterisieren. Soweit die innere Arbeitsweise einer Maschine unbeobachtbar war, wurde diese oft als *black box* behandelt. Die Analysewerkzeuge waren oft Differential- und Integralgleichungen.

Kybernetik entstand im Zusammenhang mit der Regelung von Maschinen und mit elektronischen Schaltungen und wurde, geprägt durch den Titel des Buches von Wiener: „Cybernetics or control and communication in the animal and the machine", oft als das Studium von Kontrolle und Kommunikation verstanden. In der Zeit, in der die Kybernetik ursprünglich entwickelt wurde, war das Rechnermodell analog. Die Ein- und Ausgaben für die Maschine, die analysiert werden soll, waren meistens kontinuierliche Funktionen mit vernünftigen Ableitungen. Die Mechanismen für die automatische Analyse und Modellierung waren oft Objekte, die man heute als analoge Komponenten bezeichnen würde.

Die meiste Arbeit in der Kybernetik zielte auf das Verstehen von Tieren und Intelligenz ab. Tiere wurden als Maschinen modelliert und es wurde gehofft, daß man aus diesen Modellen ableiten kann, wie die Tiere ihre Verhaltensweisen durch Lernen ändern und wie dies für den ganzen Organismus zu einer besseren Anpassung an die Umgebung führen kann. Der ursprüngliche Grundgedanke in der Kybernetik beschränkte sich auf die Beobachtung von Zuständen eines Systems. Der Nachteil dabei war, daß die beobachteten und definierten Zustände völlig abhängig von einem Beobachter waren, der als objektiv ausgelegt wurde und keine Wirkung auf das beobachtete System hatte. Es wurde aber ziemlich früh erkannt, daß, um das von einem Organismus produzierte Verhalten zu verstehen, der Organismus und seine Umgebung gemeinsam modelliert werden müssen [11]. Die Technik der Rückkopplung [12] wurde dazu benutzt, um Antworten auf Fragen wie die Stabilität des Systems bei Störungen in der Umgebung oder die Fähigkeit des Systems, bestimmte Parameter innerhalb vorgeschriebener Grenzen zu halten, zu finden.

Obwohl in der Kybernetik die meisten Modelle für Organismen, eher eine abstrakte Veranschaulichung von *Homeostasis* (automatische Tendenz eines Systems, einen Gleichgewichtszustand zu halten) waren, gab es einige Arbeiten, die sich auch mit physischen Robotern befaßt haben. In [164, 165, 166] werden Roboter beschrieben, die mit Hilfe von Prinzipien aus der Kybernetik gebaut wurden und zielorientiertes Verhalten, Homeostasis und Lernfähigkeiten demonstriert haben. Bei diesen Experimenten gab es aber zwei einschränkende Faktoren: erstens, die Recheneinheiten waren zu groß, um kleine, in sich abgeschlossene Roboter zu bauen und zweitens, Mechanismen zur abstrakten Beschreibung von Verhalten in einer modularisierten Art und Weise (eine Implementierung soll

die einfacheren Komponenten widerspiegeln) waren nicht vorhanden. Die Implementierung der Denkmodelle wurde somit im ersten Fall durch technologische Hindernisse und im zweiten Fall durch Mangel an bestimmten kritischen Modellkomponenten (Organisation in Untermodule) eingeschränkt.

Seit Mitte der sechziger Jahre wurde es klar, daß das Studium von Intelligenz auf der Basis von Grundsätzen der Kybernetik, um erfolgreich zu sein, bessere Abstraktionsebenen und Analysetools braucht. Arbib warnt zum Beispiel vor der starken Bindung an Modelle [4], auch an mathematischen Modelle, weil sie auch falsch sein könnten. Unglücklicherweise, sind bis heute die wenigen hauptsächlich benutzten Prinzipien des Gebiets der Kybernetik, wie z.B. das Gesetz der erforderlichen Mannigfaltigkeit (law of requisite variety) oder das Prinzip, daß das Ganze mehr als die Summe seiner Teile ist, typisch verschwommen und umstritten und weisen einen Mangel an Zusammenhang miteinander auf.

2.2.3 Künstliche Intelligenz

Die Künstliche Intelligenz (KI) [104, 180, 26, 32] ist heute eine anerkannte Disziplin. Das allgemeine Gebiet wird häufig dadurch charakterisiert, daß versucht wird, Computern die Fähigkeit zu verleihen, Aufgaben zu verrichten, die Intelligenz verlangen, wenn sie von Menschen ausgeführt werden. Winston [180] zum Beispiel charakterisiert die Ziele der KI als die Konstruktion von brauchbaren intelligenten Systemen sowie das Verstehen menschlicher Intelligenz .

Arbeiten in der KI basierten meistens auf der Annahme, daß Intelligenz eine Berechnungsart ist, die im Gehirn von Menschen stattfindet und ähnlich wie in der mathematischen Logik, die Manipulation von symbolischen Repräsentationen von ,,Fakten" und ,,Wissen" involviert. Es wurde angenommen, daß perzeptuelle Systeme in der Lage sind, symbolische Repräsentationen zu universalen Reasoning-Mechanismen zu liefern, die ihrerseits in der Lage sind, passende Aktionen zu veranlassen. Um bei der Berechnung potentielle kombinatorische Explosionen zu beschränken, wurden meistens kanonische objektive Repräsentationstechniken adoptiert [114, 91, 92].

Diese Auffassung von Intelligenz hat dazu geführt, daß KI-Forscher sich auf fortgeschrit-

tene intellektuelle Fähigkeiten konzentriert haben, wie z.B. die Benutzung von Sprache, die Planung von komplexen Sequenzen von Tasks oder die Anwendung von Expertenwissen für Aufgaben wie z.B. die Diagnose von Krankheiten. Mehrere Forschungsgruppen hatten bemerkenswerte Erfolge bei der Programmierung von Systemen, die solche Aufgaben bewältigen sollen. Diese Erfolge hatten die Überzeugung verstärkt, daß die Techniken der klassischen symbolischen KI erweiterbar sind, und daß intelligente Maschinen, die den *Turing-Test* bestehen, gebaut werden können. Der Turing-Test wurde von Alan Turing [161] im Rahmen seiner Bemühungen vorgeschlagen, eine akzeptable und konkrete Bedeutung für die Frage ,,können Maschinen denken?" zu finden. In diesem Test soll eine Person in englischer Sprache über Fernschreiber mit einer anderen Person und mit einem Computer kommunizieren. Die Kommunikation geschieht über zwei verschiedene Leitungen. Die Person weiß aber nicht welche Leitung zur anderen Person führt und welche zum Computer. Das Ziel ist es, zu erraten, ob es sich am anderen Ende um eine Person oder um einen Computer handelt. Wenn nach einem Dialog durch beide Leitungen die Person nicht sagen kann, welche Leitung mit der anderen Person und welche mit dem Computer verbunden ist, dann hat der Computer den Turing-Test bestanden und kann als intelligent bezeichnet werden. Dieser Test wurde mit der Zeit zu einem informellen Ziel der KI. Turing schlägt vor, daß die Maschine versuchen soll, eine Person zu simulieren, indem sie sich Zeit nimmt und bei arithmetischen Problemen Fehler macht.

Angetrieben von dieser Ansicht haben die KI-Forscher bei dem Versuch intelligente Robotik-Systeme zu konstruieren, einen Forschungsstil adoptiert, in dem Agenten hauptsächlich als Problemlöser konzipiert waren, die in symbolischen, abstrakten Domänen agierten. Die implizite Idee dabei ist, daß die Perzeptions- und Motorschnittstellen Mengen von Symbolen sind, die vom zentralen Intelligenzsystem manipuliert werden. Die Symbole sind zufällig. Ihre Interpretation und Semantik (was die Symbole repräsentieren) wird von den Beobachtern des Systems festgelegt. Intelligentes Agieren involviert somit den Empfang von Symbolen von symbolerzeugenden sensorischen Apparaten, sowie die anschließende Manipulation dieser Symbolen (z.B. mit Hilfe von Techniken der mathematischen Logik oder der algorithmischen Suche), um ein Ausgabesymbol oder eine Symbolstruktur zu erzeugen [65]. Diese Idee wurde unter einer Vielzahl von Namen bekannt, wie zum Beispiel *Repräsentationshypothese* (Representation Hy-

pothesis) [178], *physische Symbolsystemhypothese* (Physical Symbol System Hypothesis) [114], *Informationsverarbeitungsparadigma* (Information Processing Paradigm) [128] und *Wissensrepräsentationshypothese* (Knowledge Representation Hypothesis) [150]. Es gab natürlich viele technische Unterschiede zwischen den Forschungsprogrammen, die unter diesen verschiedenen Bezeichnungen durchgeführt wurden. Sie hatten aber alle die gleiche Sicht auf die Mechanismen, die für intelligentes Verhalten zugrundeliegen.

Für die Robotik wurden dadurch im wesentlichen zwei Bereiche identifiziert, in denen Techniken der KI angewendet werden können, nämlich das Lösen des Planungsproblems [67] und das Interpretieren von sensoriellen Informationen und Weltmodellierung [151].

Planen

Die Planer (Problemlöser) [155, 142, 113, 112, 41] manipulieren Symbole und bestimmen durch diese Manipulationen passende Aktionen. Planen ist der Vorgang der Planerzeugung. Ein Plan ist eine Darstellung, bei der eine oder mehrere Folgen von Aktionen (Zielen, Aufgaben) entworfen werden, um von einem Anfangszustand über eine Menge von Zwischenzuständen zu einem Endzustand zu gelangen. Diese Aktionsfolge ist unter Randbedingungen, die sich ändern können, zu generieren. Aktionen werden durch die Anwendung von Operatoren auf Zustände erzeugt. Durch die äußeren Randbedingungen vorgegeben können verschiedene Zwischenzustände auftreten und selbst der Zielzustand kann Änderungen unterworfen sein. Jede primitive Aktion wird oft durch Angabe ihrer erforderlichen Vorbedingungen und der erwarteten Folgen (Nachbedingungen) modelliert. Es wird dann eine Sequenz von Aktionen gesucht. Die erste Aktion soll im aktuellen Zustand der Umwelt anwendbar sein und die Folgen der letzten Aktion müssen das Ziel enthalten.

Ein Planer ist in der Regel so organisiert, daß er einen Suchraum definiert und einen Punkt in diesem Raum sucht, der das Ziel repräsentiert [160]. Die älteren Planer (vor 1975) definierten die Punkte in diesen Räumen als Zustände der statischen Anwendungswelt. Die neueren Planer (nach 1975) definieren Punkte in dem Suchraum als Teilpläne.

Seit den frühen siebziger Jahren waren Forscher auf dem Gebiet des Planens sehr intensiv damit beschäftigt, intelligente künstliche Agenten zu entwerfen. Und in der Tat, die meisten Innovationen beim Agentendesign scheinen von dieser Gemeinschaft initiiert

worden zu sein. Planen wurde hauptsächlich als automatische Programmierung verstanden, und es ging darum, eine Vorgehensweise zu bestimmen, die nach Initiierung zu einem gewünschten Ziel (goal) führen würde. Es wurde für lange Zeit behauptet, daß in irgend einer Form ein KI-Planungssystem die zentrale Komponente eines jeden künstlichen Agenten sein wird. Der vielleicht bestbekannte Planer war STRIPS [41]. Dieses System bekommt als Eingabe eine symbolische Beschreibung von der Welt und von einem gewünschten Zielzustand, sowie eine Menge von Aktionsbeschreibungen, die die mit verschiedenen Aktionen assoziierten Vor- und Nachbedingungen charakterisieren. Es versucht dann eine Sequenz von Aktionen zu finden, die zum Ziel führt. Dies geschieht mit Hilfe einer einfachen Analyse, die hauptsächlich aus dem Vergleich der Nachbedingungen von Aktionen mit dem gewünschten Ziel besteht. Das STRIPS-Planungssystem war zwar sehr einfach aber leider, selbst bei Problemen von moderater Komplexität, nicht sehr effektiv. Viele anschließende Arbeiten haben sich der Entwicklung von effektiveren Techniken gewidmet. Die bedeutendsten Innovationen waren das *hierarchische* und das *nicht-lineare* Planen [140, 141].

Weltmodellierung

Die obige Abstraktion (Repräsentationshypothese) hat zur Entstehung eines der größten Teilgebiete der KI, nämlich der *Wissensrepräsentation* [18], geführt. Die Wissensrepräsentationssysteme hatten das Ziel, Wissen über die Umwelt zu repräsentieren und Weltmodelle durch Transformation der Sensordaten in Symbole zu konstruieren.

Bilderkennung [92] wurde oft als adäquates Mittel für die Bildung von Weltmodellen angesehen. In [135] wurde ein Bildverarbeitungsprogramm demonstriert, das Bilder von einfachen Blöcken mit vorher gespeicherten Modelle vergleichen kann. Dieses Programm war der Vorläufer der modernen Bildverarbeitungsprogramme, und es dauerte Jahre, bis andere Programme seine Leistung erbringen konnten. Die Eingaben für das Programm waren Graustufenbilder der Umgebung. Das Programm extrahierte daraus eine Linienzeichnung, die nach einigen Transformationen mit vorher gespeicherten Modellen verglichen wurde. Die Ergebnisse sahen zur damaligen Zeit sehr überzeugend aus. Dies hat die Vermutung verstärkt, daß das Programm die richtige und natürliche Art darstellt, Bilder zu verarbeiten und Modelle der Realität zu bilden, und daß das Linie-

nerkennungsproblem lösbar ist.

Inspiriert durch diese und andere Arbeiten [179], haben sich Forscher auf diesem Gebiet jahrelang mit der Bearbeitung von Szenenbeschreibungsproblemen beschäftigt. Die implizite Absicht dabei war, daß die Bildverarbeitungssysteme in der Lage sein würden, die benötigten Weltmodelle bereitzustellen, wenn man die nötigen Reasoning-Systeme der KI hat. Diese Weltmodelle wurden oft als dreidimensionale Rekonstruktionen der statischen externen Umgebung aufgefaßt. Für dynamische Umgebungen wurde angenommen, daß es ausreicht, wenn diese Rekonstruktionen oft und schnell gemacht werden.

Mit der Zeit wurde es immer deutlicher, wie schwierig es ist, vollständige Weltmodelle zu bilden und zu manipulieren. Die älteren Programme hatten nur eine sehr eingeschränkte und vereinfachte Menge von Bildern bearbeitet (die Beleuchtung, die Farbe der Blöcke und der Hintergrund wurden so sorgfältig gewählt, daß die Bilder der Blöcke vollständige Linienzeichnungen ohne Störungen produzieren konnten). Im allgemeinen ist es aber sehr schwierig, verläßliche Linienzeichnungen aus realistischen Bildern zu extrahieren.

Mobile Roboter

Im Laufe der Jahre gab es auch einige Arbeiten, die sich mit mobilen Robotern beschäftigt haben. Shakey [116], der vom SRI (Stanford Research Institute) in den späten sechziger Jahren entwickelt wurde, ist vielleicht das bekannteste Beispiel. Ausgestattet mit einer begrenzten Fähigkeit für die Wahrnehmung und Modellierung seiner Umgebung, konnte Shakey Aufgaben erledigen, die Planung, Wegsuche und Umordnung von einfachen Objekten erforderten. Shakey hat in speziell für ihn vorbereiteten Räumen agiert. Er navigierte von einem Raum zu einem anderen und versuchte ein Ziel, das durch Fernschreiber eingegeben wurde, zu erreichen. Abhängig vom Ziel und von den Umständen, sollte er Hindernisse (große, gemalte Blöcke) umfahren, sie aus seinem Weg wegräumen oder sie zu bestimmten Positionen schieben. Der Hauptsensor war eine schwarz-weiß Onboard-Fernsehkamera. Ein Offboard-Computer analysierte das Bild und fügte die Beschreibungen von dem, was gesehen wurde, in ein existierendes Weltmodell ein, das auf Prädikatenlogik erster Stufe basierte. Ein Planungsprogramm, STRIPS genannt (siehe oben), verarbeitete diese symbolischen Beschreibungen der Umwelt, um eine Sequenz von Aktionen für Shakey zu generieren. Diese Pläne wurden durch eine

Serie von Verfeinerungen zu Aufrufen von atomaren Aktionen übersetzt. Die atomaren Aktionen waren ziemlich fest rückgekoppelt mit atomaren Sensoroperationen, die die anderen Sensoren von Shakey, wie z.B. Stoßstange und Wegmesser, nutzten.

Shakey wurde zu der damaligen Zeit als ein großer Erfolg betrachtet, mit dem man ein integriertes System, das Mobilität, Perzeption, Repräsentation, Planung, Ausführung und Wiederanlauf im Fehlerfall umfaßte, demonstrieren konnte. Es gab aber auch andere signifikante Entwicklungen wie z.B. CART [108] in Stanford und Hilare [51] in Toulouse.

All diese Systeme hatten Offboard-Computer und konnten damit die bestverfügbare Rechenleistung nutzen. Sie hatten größtenteils in statischen Umgebungen, die speziell für sie konstruiert wurden, agiert und mit Hilfe von Sensoren versucht, zwei- oder dreidimensionale Weltmodelle für ihre Umgebungen zu erzeugen. Danach wurden Planer aktiviert, die die Weltmodelle benutzten, um Pläne von Aktionen zu erzeugen und die angegebenen Ziele zu erreichen. Trotz der vielen Vereinfachungen waren diese Roboter äußerst langsam. Die meiste Rechenzeit wurde von dem perzeptuellen Teil des Systems und von der Komponente zur Bildung des Weltmodells verbraucht. Verhältnismäßig wenige Berechnungen wurden beim Planen und Agieren benutzt.

Ein wichtiger Effekt dieser Arbeiten war es, einen Rahmen geschafft zu haben, in dem andere Forscher operieren konnten, ohne daß sie ihre Ideen an realen Robotern getestet haben, und sogar ohne daß sie auf reale Roboterdaten zugegriffen haben. Es wurde dabei immer implizit angenommen, daß, wenn für den einfacheren Fall des Operierens in einer statischen Umgebung eine Lösung gefunden werden könnte, dann auch der schwierigere Fall des Operierens in einer dynamischen Umgebung lösbar wäre.

2.2.4 Biologienahe Ansätze

In den letzten Jahren wurde die klassische Auffassung von Intelligenz oft bestritten. Mehrere Forscher meinen, daß die KI-Forschung nur auf festen Grundlagen stehen kann, wenn sie mit der Biologie [34, 95] stark integriert wird. Einer der Ansprüche dieses neuen Forschungsstils ist, daß wir ein großes Stück in Richtung des Verstehens der biologischen Grundlagen intelligenter Aktivität zurückgelegt haben, wenn wir den Basismechanismus studieren und verstehen können, mit dem wir und andere Tiere die Grundantriebe (z.B.

Ernährung oder Fortpflanzung) befriedigen. Es stellt sich heraus, daß viele der Annahmen der klassischen KI überprüft werden müssen, wenn ein strenger biologischer Ansatz angewendet wird.

Im folgenden werden zuerst einige Teilgebiete der Biologie kurz erläutert und anschließend die daraus inspirierten neuen Forschungsfelder für intelligente Agenten und die dadurch entstandenen Begriffe aufgeführt.

Neurowissenschaft (Neuroscience)

In den sechziger Jahren und wiederum Mitte der achtziger Jahre gab es ein weitverbreitetes Interesse an sogenannten ,,künstlichen neuronalen Netzen" [115, 105, 139], (siehe auch Kapitel 4). Diese künstlichen Netze wurden hauptsächlich durch physiologische Beobachtungen des Gehirns und anderer Teile des Nervensystems motiviert. Der Haupteinfluß kam von der Beobachtung, daß das Nervensystem aus einer großen Anzahl von sehr einfachen ,,Bearbeitungseinheiten" (*Neuronen*) besteht, die sehr komplex verschaltet sind und alle parallel arbeiten. Weil die Gesamtleistung des Netzes oft aufgrund der Interaktion zwischen den Neuronen entsteht (es gibt keine klare Arbeitsaufteilung, in der bestimmte, einzelne Neuronen besondere Aufgaben verrichten), werden künstliche neuronale Netze als Beispiele für parallele, verteilte Datenverarbeitungssysteme betrachtet.

Obwohl es noch nicht bekannt ist, wie ,,Berechnungen" im Gehirn genau stattfinden, war es mit Hilfe neuronaler Netze möglich, einige Erkenntnisse über Aspekte intelligenten Verhaltens sowie Anhaltspunkte über Sensor- und Motorsysteme zu gewinnen. [170] zum Beispiel, gibt einen Einblick in die Weise, in der sensoneurologische Koppelungen mit der Umwelt stattfinden. Durch die Wahl der richtigen Sensoren brauchen Tiere oft nur sehr wenige neurologische ,,Berechnungen", um die nötigen Informationen für die Erledigung einer bestimmten Aufgabe zu extrahieren. [53] und [33] beschreiben die Arbeitsweise von einfachen Tieren, basierend auf dem Verständnis ihrer neurologischen Schaltungen.

Verhaltensforschung (Ethology)

Tiere haben Nervensysteme, die bestimmte *Verhaltensweisen* verursachen. Einige von diesen Verhaltensweisen würden wir oft als ,,intelligent" bezeichnen. Das Studium des Verhaltens von Tieren ist unter dem Namen *Verhaltensforschung* (Ethology) bekannt. Die Verhaltensforschung versucht die Ursächlichkeit, die Entwicklung, das Überleben und die Evolution von Verhaltensmustern in Tieren zu erklären [87, 98]. Der Erfolg oder Mißerfolg eines bestimmten Nervensystems ist davon abhängig, ob es vernünftiges Verhalten erzeugt oder nicht. ,,Weglaufen vor Feinden" oder ,,Bewegung Richtung Nahrung" sind im allgemeinen vernünftige Verhaltensweisen. ,,Bewegung Richtung Feind" oder ,,Weglaufen vor Nahrung" sind dagegen unvernünftig.

Es gibt noch keine vollständig entwickelte Theorie mit der folgende Frage genau beantwortet werden kann: ,,Wie wird die Entscheidung getroffen, welches Verhaltensmuster (z.B. Trinken oder Essen) in einem Tier aktiviert sein muß?" Eine große Anzahl von Versuchen macht aber die Existenz von komplexen internen und externen Feedback-Loops deutlich, die bei der Ermittlung von passenden Verhaltensmustern benutzt werden. In [99], zum Beispiel, werden solche Versuche präsentiert und die Herausforderungen für die Theorien diskutiert.

Ökologie (Ecology)

Das Gebiet der *Ökologie* beschäftigt sich unter anderem mit der Interaktion von Organismen mit ihrer sozialen und physischen Umgebung (siehe z.B. [82]). Für ein Tier zum Beispiel, ist die Entscheidung, ob ein bestimmtes Verhalten vernünftig ist oder nicht, nicht nur abhängig von den unmittelbaren Umständen des Tieres, sondern auch von der *ökologischen Nische* des Tieres, d.h. von der Interaktion des Tieres mit seinem Standort und mit anderen Lebewesen wie Feinde, Beute oder Mitglieder der gleichen Art. Viele Verhaltensweisen sind in hohem Grade an bestimmte Umgebungen und ökologische Nischen gebunden. Der Versuch, das Tier getrennt von seiner Umwelt zu analysieren, führt oft nur zur Verwirrung. Das Tier und sein Verhalten sind an seine natürliche Umgebung so fein angepaßt, daß nur sehr wenig von dem ursprünglich interessanten Verhalten übrig bleibt, wenn das Tier von der Umgebung getrennt wird.

Evolution

Laut der biologischen *Evolution* [47, 132], sind für jedes Tier das Nervensystem, die Verhaltensweisen und die ökologische Nische *evolutionären Prozessen* wie der natürlichen Selektion ("survival-of-the-fittest") unterworfen. Die Evolution ist ein kontinuierlicher Prozeß, der dauernd etablierte Muster geringfügig ändert und an die aktuelle Umgebung anpaßt. Fast immer spielt die evolutionäre Geschichte eines Tieres eine sehr wichtige Rolle bei der Erklärung des Verhaltens des Tieres und seiner Physiologie.

Neue Forschungsgebiete

Für die neuen Forschungsfelder, die auf den oben erwähnten Teilgebieten der Biologie basieren, gibt es noch keine einheitliche Terminologie. In den letzen Jahren wurden unter anderem folgende englische Bezeichnungen benutzt: *Behavior-Based Robots* [19, 124, 152], *Situated Robotics* [1, 76, 23, 25], *Animat (artificial animals) Research* [176, 16, 181, 101], *Evolutionary Robotics* [31, 154], *Adaptive Behavior* [88, 44], *Artificial Life* [84, 85].

Die gegenwärtige Forschung in diesen Gebieten sieht keinen Bedarf an symbolischen Repräsentationen und an detailliertem Planen, die für die klassische KI typisch sind. Die zwei Prinzipien, die oft angenommen werden, sind: Erstens Biologie ernst zu nehmen, und zweitens sich auf *situierte* Aktivität von *ganzen* autonomen Agenten zu konzentrieren.

Es werden oft Computermodelle benutzt, um Prozesse zu studieren, die mit relativ einfachen, lokal interagierenden Einheiten beginnen und komplexe individuelle oder gemeinschaftliche Verhalten generieren. Der Begriff *Emergence* stellt ein Schlüsselkonzept dar. Beim Studium komplexer Systeme wird die Idee von Emergence benutzt, um die Entstehung von Mustern, Strukturen oder Eigenschaften zu bezeichnen, die durch die Betrachtung der vorher existierenden Komponenten und ihrer Interaktion alleine nicht als adäquat beschreibbar scheinen. Komplexes high-level Verhalten wird häufig als Resultat der Kombination einfacher low-level Berechnungsmechanismen interpretiert.

Für einen Agenten (Roboter) wird oft genau die gleiche Definition wie für ein lebendes System benutzt [153]. Ein Roboter ist demnach ein System, das sich aktiv selbsterhalten

kann. Der Antrieb in Richtung auf *Selbsterhaltung* (self-maintenance), den man in der Biologie in vielen verschiedenen Ebenen findet, wird daher auch für Robotik-Agenten definiert:

- *Die genetische Ebene:* dies ist die Ebene, die die Überlebensfähigkeit der Spezies aufrechterhält. Die Mechanismen des Kopierens (copying), der Mutation (mutation) und der Rekombination (recombination) zusammen mit der Selektion (selection), die auf die gentragenden Organismen wirken, sind die Hauptmechanismen mit denen ein kohärenter Genpool sich selbst aufrechterhalten und an veränderliche Verhältnisse anpassen kann. Für künstliche Robotik-Agenten entsprechen die Konstruktionspläne, die Entwurfprinzipien und die initiale Struktur eines Agenten einer Art genetischer Ebene. Mehrere Forscher haben angefangen, diese Ebene zu realisieren und Experimente in der genetischen Evolution durchzuführen [43, 31].
- *Die strukturelle Ebene:* dies ist die Ebene der Komponenten und Prozesse, die die einzelnen Agenten bilden: Zellen, Ansammlungen von Zellen, Organe, usw. Jede dieser Komponenten hat ihre eigenen Verteidigungsmechanismen, Erneuerungsmechanismen und adaptiven Prozesse. Im Fall des Gehirns zum Beispiel gibt es Neuronen, Netzwerke von Neuronen, neuronale Gruppen, Regionen mit bestimmten Funktionen, usw. In künstlichen Systemen involvieren diese Komponenten interne Größen, Elektronik- und Berechnungseinheiten, Systeme zur Einstellung der Sensoren und Aktoren, usw.
- *Die Individuum-Ebene:* dies ist die Ebene des individuellen Agenten, der sich selbst aufrechterhalten soll, indem er sich in der gegebenen Umgebung passend verhält. In vielen biologischen Systemen (z.B. Bakterien- oder Ameisenkolonien) haben Individuen keinen oder nur geringen Eigennutz. Es ist aber klar, daß mit steigender Komplexität des Systems das Individuum schrittweise wichtiger wird, und daß Konflikte zwischen den Ebenen auftreten können. Hohe Individualität scheint eine enge Verbindung mit der Entwicklung der Intelligenz zu haben. In künstlichen Systemen entspricht die individuelle Ebene der Ebene des Robotik-Agenten als ganzes, der innerhalb seiner ökologischen Nische überleben soll.
- *Die Gruppenebene:* dies ist die Ebene, in der Gruppen von Individuen gemeinsam eine kohärente Einheit bilden und sich als Gruppe aufrechterhalten. Dies kann

Verteidigungsmechanismen, soziale Unterscheidung entsprechend den Bedürfnissen der Gruppe, usw. beinhalten. In künstlichen Systemen ist die Gruppenebene wichtig, wenn es Gruppen von Robotik-Agenten gibt, die kooperieren müssen, um Aufgaben gemeinsam zu lösen [93].

Kapitel 3

STEUERUNGSARCHITEKTUREN FÜR AUTONOME MOBILE ROBOTER

Die Entwicklung autonomer Systeme zielt darauf ab, Systeme zu realisieren, welche in der Lage sind, selbständig eine bestimmte Klasse von Aufgaben in einer realen, teilweise unbekannten Umwelt durchzuführen. Im Bereich intelligenter autonomer Roboter zeigt sich, daß diese Selbständigkeit nur durch eine sehr enge Wechselwirkung zwischen System und Umwelt erreicht werden kann. Hierin liegen die prinzipiellen Schwierigkeiten für eine konkrete Realisierung autonomer Systeme begründet.

Eine reale Welt ist durch kontinuierliche Zustandsänderungen charakterisiert und stellt somit einen unendlichen Zustandsraum dar. Der gesamte Merkmalsraum kann nicht exakt durch diskrete Messungen erfaßt werden. Sämtliche Daten, welche ein System über seine Umgebung aus Sensorbeobachtungen gewinnen kann, sind mehr oder weniger unscharf, d.h. mit einer bestimmten Ungenauigkeit behaftet. Die von Sensoren extrahierten Daten sind generell mit Rauschen und Messfehlern belegt und lassen sich oft nicht direkt auf die gewünschten physikalischen Größen (z.B. Entfernung oder Volumen) abbilden. Messungen verschiedener Sensoren können sich überlappen oder sich gar widersprechen. Systeme, die in der realen Welt agieren, sollten daher Methoden zur Behebung und zum Umgang mit diesen Ungenauigkeiten bereitstellen.

Neben der Behandlung ungenauer Information über die Umwelt lassen sich weitere Anforderungen aufstellen, welche bei der Implementierung von Steuerungsstrukturen für autonome Systeme berücksichtigt werden müssen:

- Modularität: Das System soll inkrementell aus kleinen Komponenten, die leicht zu

implementieren und leicht zu verstehen sind, gebildet werden können.

- Aufmerksamkeit: Zu keinem Zeitpunkt darf sich das System dessen was geschieht nicht bewußt sein. Es muß immer in der Lage sein, auf unvorhersehbare Sensordaten zu reagieren.
- Robustheit: Ausfälle von Systemkomponenten bzw. unvorhergesehene Veränderungen in der Umwelt dürfen höchstens zu einer angemessenen Leistungsminderung des Gesamtsystems führen. Selbst im Falle schwerwiegender Systemfehler sollte es noch möglich sein, das System in einen definierten Endzustand zu bringen.

In diesem Kapitel werden existierende Methoden zur Implementierung solcher Systeme verglichen und grob klassifiziert. Siehe auch [56].

3.1 Was ist eine Architektur

Ein Robotersystem besteht aus einer Vielzahl von funktionalen Einzelkomponenten, die zur Ausführung einer bestimmten Aufgabe kooperierend zusammenwirken und einen Roboter in seiner Funktionalität charakterisieren. Grob können diese Komponenten in drei Gruppen eingeteilt werden: Mechanik, Sensorik und Steuerung (siehe Abbildung 3.1).

Durch die mechanische Struktur des Roboters ist seine Fähigkeit bestimmt, in seiner Umgebung mit den Effektoren definierte Positionen und Orientierungen einzunehmen und Bewegungstrajektorien abzufahren. Die Beweglichkeit des Robotersystems ergibt sich durch die Auslegung seiner mechanischen Subsysteme. Sensoren dienen der Erfassung der inneren Zustände des Robotersystems, der aktuellen Wechselwirkung der Effektoren mit der Umgebung und der äußeren Zustände im Einsatzbereich des Robotersystems. Die Funktion von Sensoren basiert auf der Umwandlung eines am Eingang anliegenden physikalischen Phänomens (z.B. Druck, Kraft, Berührung) in ein quantitatives elektrisches Maß. Die Steuerung (intelligente Verarbeitung) verbindet die Mechanik mit der Sensorik und spielt die wichtigste Rolle bei der Gestaltung des Verhaltens des Systems. Die Wechselwirkung mit der Umwelt läuft wie folgt ab (siehe Abbildung 3.2): Zugang zur realen Welt erfolgt über Sensoren. Die intelligente Verarbeitung setzt die erhaltenen Daten gemäß einer bestimmten Steuerungsstrategie um und löst die Aktionen des

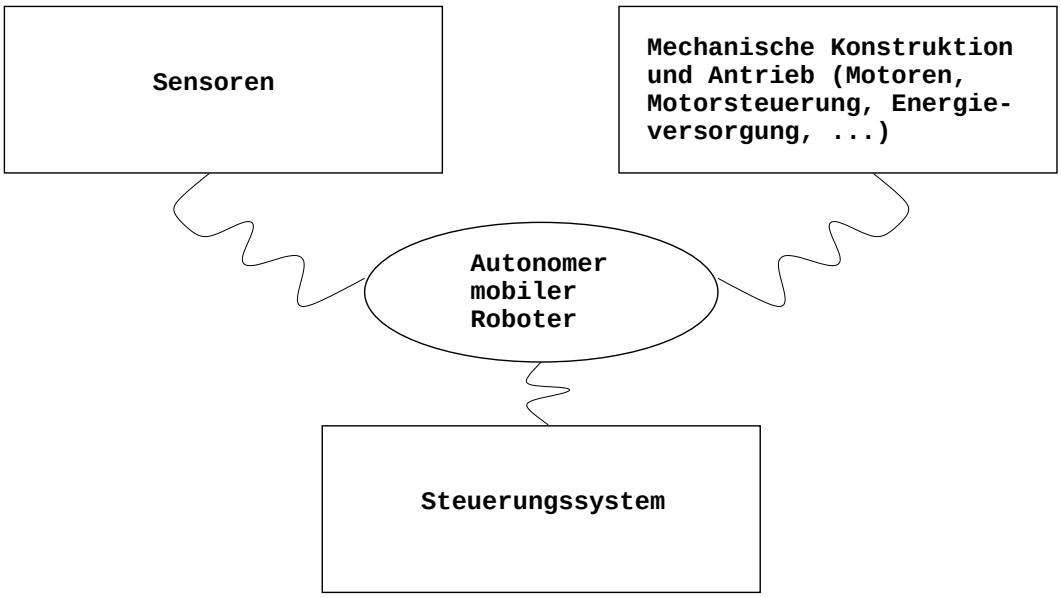


Abbildung 3.1: *Komponenten eines Robotersystems.*

Roboters aus. Die Aktionen verändern die reale Welt und alles fängt wieder von vorne an.

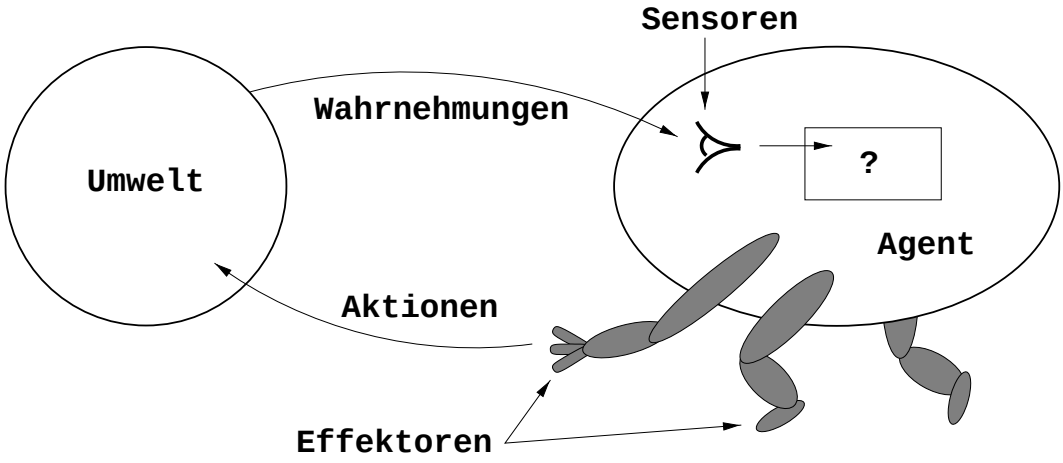


Abbildung 3.2: *Wechselwirkung mit der Umwelt.*

Analog zu Rechnerarchitekturen spricht man bei Robotern von Roboterarchitekturen oder Steuerungsarchitekturen. Sie sind durch ihr Operationsprinzip, ihre interne Organisation sowie die zu verarbeitenden Daten charakterisiert. Die Roboterarchitektur definiert ein logisches Konzept zur funktionellen Organisation eines Roboters, zur Or-

ganisation der Rechenvorgänge und zur Organisation von logischem Verhalten sowie Strukturen zur Interpretation von Umweltdaten.

Während der letzten Jahre wurden viele verschiedene Architekturen für autonome mobile Roboter in der Literatur vorgeschlagen. All diese Architekturen definieren grundsätzlich Systeme, die aus zusammengebundenen Komponenten bestehen. Abhängig von der Architektur können die Komponenten Funktionsmodule, Verhaltensmodule, geometrische Datenbanken, usw. sein. Auch die Kommunikation zwischen diesen Komponenten ist variabel. Beides, Systeme mit niedriger Bandbreite und Systeme mit hoher Bandbreite existieren: Blackboard, Punkt-zu-Punkt-Kommunikation, zentralisierte Kommunikation, shared memory, usw. In jedem System werden außerdem Werkzeuge für die Auflösung von Konflikten zwischen Tasks definiert: Balckboard, Priorisierung, usw.

Abhängig von der Topologie (Zerlegungsrichtung), können zwei Basisklassen innerhalb diese Architekturen unterschieden werden: *Funktionsbasierte* Architekturen (z.B. [102, 117, 52, 77, 80, 121, 2]) und *verhaltensbasierte* Architekturen (z.B. [19, 24, 25, 76, 162, 5, 119]).

Der funktionsbasierte Ansatz stellt die klassische Art für die Bildung von Steuerungssystemen für Roboter dar. Bei diesem Ansatz wird das Steuerungsproblem in eine Reihe von Funktionseinheiten zerlegt, die bestimmte Tasks wie Weltmodellierung, Perzeption, Planung und Ausführung bearbeiten. Das Gesamtverhalten des Systems entsteht indem zu jedem Zeitpunkt jede dieser Komponenten ausgewertet wird. Der verhaltensbasierte Ansatz gilt als Alternative zum klassischen Ansatz. Er benutzt verhaltenserzeugende Module als grundlegende Einheiten bei der Zerlegung des Steuerungsproblems. Das Gesamtverhalten des Systems entsteht durch die Interaktion zwischen diesen Modulen.

Funktionsbasierte Architekturen werden auch *deliberative*, *top-down* oder *wissensgesteuerte* (knowledge-driven) Architekturen genannt. Für verhaltensbasierte Architekturen gibt es entsprechend die Bezeichnungen *reaktive*, *bottom-up* oder *datengesteuerte* (data-driven) Architekturen (siehe z.B. [49, 9, 8]). Beide Ansätze haben ihre Vor- und Nachteile. Im folgenden werden diese Ansätze etwas näher betrachtet.

3.2 Funktionsorientierte Architekturen

Der funktionsbasierte Ansatz ist ein Paradigma, das seit den ersten Tagen der Robotik angewendet wird und heute immer noch ein aktives Forschungsthema ist. Er basiert auf den Ideen der Weltmodellierung und Planung und zerlegt das Kontrollproblem in einer Sequenz von Funktionseinheiten. Autonomie soll in diesem Ansatz durch eine hoch entwickelte Integration von Perzeption (perception), Kognition (cognition) und Aktion (action) entstehen. Perzeption involviert die Interpretation von Sensordaten. Aktion umfaßt die Fähigkeit, durch die Umwelt zu navigieren und nützliche Aufgaben zu erledigen. Kognition verbindet diese beiden Prozesse. Abbildung 3.3 zeigt einen funktionsbasierten Entwurf für einen autonomen Roboter [131].

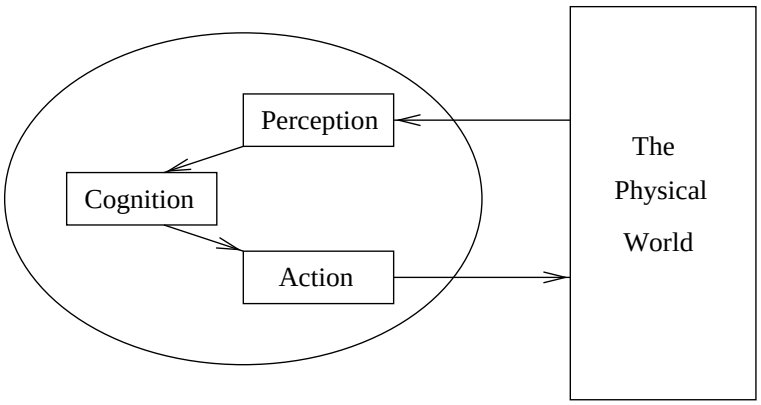


Abbildung 3.3: *Ein Entwurf für einen autonomen Roboter.*

Der Ablauf in einer solchen Architektur ist üblicherweise wie folgt: Zuerst werden die Daten aus allen Sensoren gesammelt. Dann wird versucht, Rauschen und Konflikte aufzulösen, so daß ein konsistentes Weltmodell konstruiert werden kann. Das Weltmodell muß die geometrischen Details aller Objekte in der Roboterwelt sowie ihre Positionen und Orientierungen enthalten. Danach benutzt der Roboter sein Weltmodell, um für das (meistens durch den Programmierer des Roboters) gegebene Ziel eine Sequenz von Aktionen zu planen, die zu diesem Ziel führt. Schließlich wird der formulierte Plan ausgeführt, indem passende Befehle zu den Aktoren geschickt werden. Bei einem fortgeschrittenen Planer kann das erzeugte Roboterprogramm auch sensorische Tests enthalten.

Funktionsbasierte Systeme können als integrierte Systeme mit mehreren Funktionsmo-

dulen angesehen werden. Für jede ,,Funktion" des intelligenten Systems gibt es eine eigenständige Systemkomponente, welche die alleinige Zuständigkeit für ihren Aufgabenbereich erhält. Systemkomponenten sind z.B. die mechanischen Bauteile und Effektoren, das Sensorsystem, die Weltmodellierung, die Planung, die Ausführung, usw. Die Funktionsfähigkeit des Gesamtsystems orientiert sich am Entwicklungsstand der einzelnen Komponenten. In der Regel zieht der Ausfall eines Systemmoduls den Ausfall des gesamten Systems nach sich. Abhängig von der Art der Interaktion zwischen den Komponenten unterscheidet man hierarchische und verteilte Strukturen.

In einer hierarchischen funktionsorientierten Architektur sind die Funktionsmodule hierarchisch und das Gesamtsystem ist in mehreren Schichten eingeteilt, die aus den Hierarchieebenen der einzelnen Funktionsmodule bestehen. Eine direkte Verbindung ist nur zwischen zwei benachbarten Schichten möglich. Abbildung 3.4 [130] zeigt ein Beispiel einer hierarchischen funktionsorientierten Struktur mit drei hierarchischen Modulen: Sensorsystem, Weltmodellierung und Planung. In der Wegplanungsebene (global path planning) erzeugt der Planungsmodul einen Bewegungspfad, der eine grobe Beschreibung des realen Weges darstellt. Der Navigator benutzt die Ergebnisse des Planers sowie andere Information und versucht die Auflösung des Bewegungspfads zu verbessern und die Fehler zu reduzieren. Der Pilot transformiert die vom Navigator beschriebenen Trajektorien in eine detailliertere Sequenz von Aktionen. Die Ausführungskomponente (execution controller) führt die Aktionen so genau wie möglich aus. Zielorientierte Aufgabenbeschreibungen befinden sich an oberster Stelle in der Hierarchie; diese werden auf unterster Ebene auf Signale abgebildet, welche direkt mit dem eigentlichen System interagieren. Jede Ebene des Planungsmoduls zerlegt komplexe Aufgaben in eine Sequenz von einfacheren Teilaufgaben für die darunterliegende Ebene. Die Art der Zerlegung wird von den vorliegenden Sensordaten auf dieser Ebene beeinflußt. Der Informationsgehalt der extrahierten Sensorwerte nimmt mit steigender Hierarchieebene zu. Die Weltmodellierungshierarchie verbindet die Komponenten des Sensormoduls und des Planungsmoduls und enthält das Weltmodell (z.B. geometrische Karten). Die momentanen Sensordaten werden dazu benutzt, das Weltmodell zu aktualisieren.

In einer verteilten funktionsorientierten Architektur bilden die Komponenten eine Menge spezialisierter Teilsysteme, welche über einen zentralen Kommunikationsmechanismus miteinander kooperieren. In der Regel umfassen diese Teilsysteme Komponenten zur

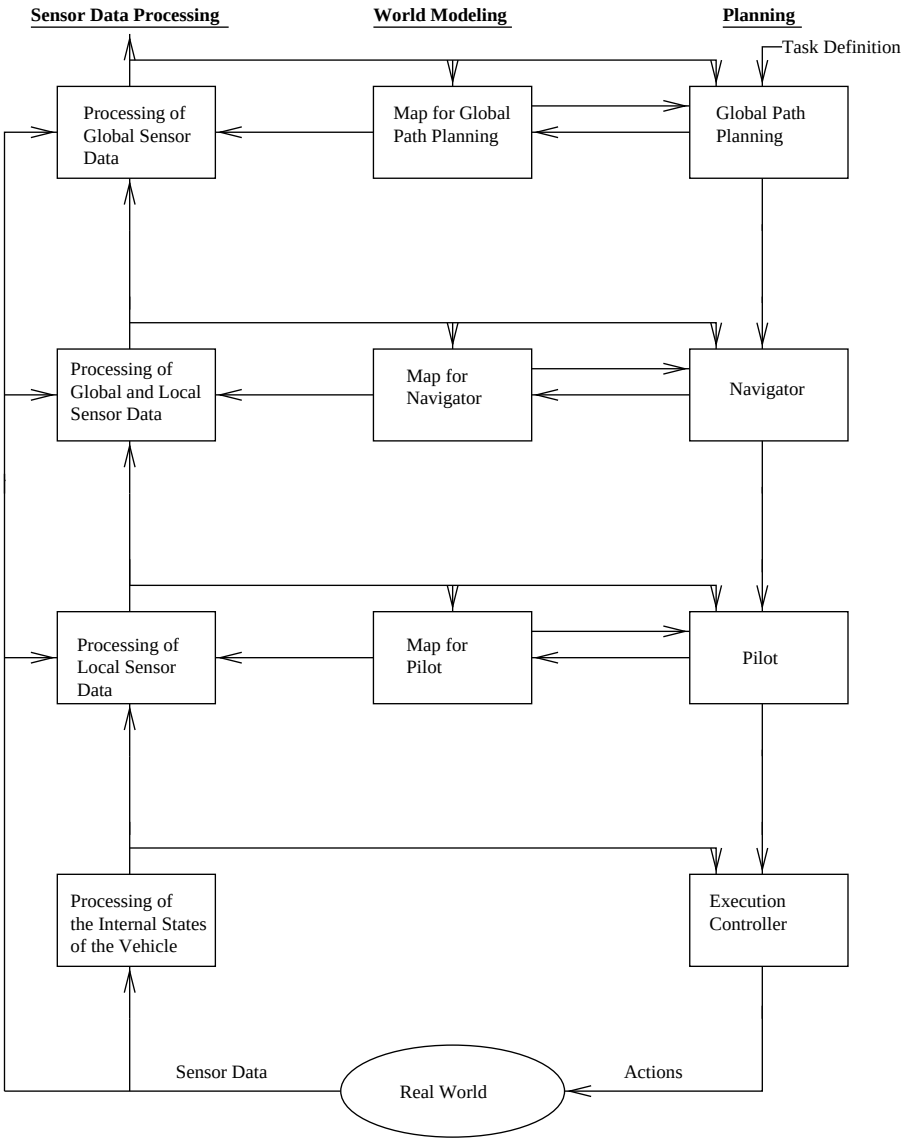


Abbildung 3.4: *Ein hierarchisches Navigationssystem.*

Umwelterfassung, Wissensspeicherung und Ausführung usw. Jedes dieser Teilsysteme ist ein Experte auf seinem Fachgebiet und arbeitet unabhängig von anderen Komponenten. Insbesondere gibt es keine expliziten Hierarchien innerhalb des Gesamtsystems, so daß ein hoher Grad an Parallelität erreicht werden kann. Abbildung 3.5 zeigt ein Beispiel für eine verteilte funktionsorientierte Architektur (Kontrollsystem des mobilen Roboters NavLab der CMU [52]). Das Gesamtsystem besteht aus einer Anzahl unabhängiger Systemmodule, welche über ein Blackboard-System miteinander kommunizieren. Der

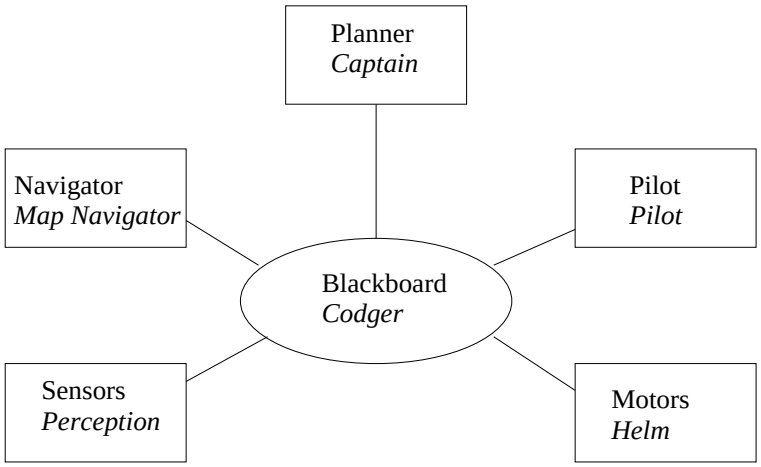


Abbildung 3.5: *Steuerungsarchitektur des Navlab-Systems*

Planer (Captain) ist die Zentrale Komponente innerhalb des Gesamtsystems. Er interpretiert Fahrbefehle und sendet entsprechende Kommandos an den Navigator (Map Navigator). Dieser durchsucht die Datenbasis nach einem optimalen Fahrweg und stellt daraus eine Folge von Wegsegmenten zusammen. Diese Information wird an den Piloten (Pilot) weitergegeben. Der Pilot hat nun die Aufgabe, das Fahren entlang einzelner Wegsegmente mit Hilfe der Sensoren (Perception) bzw. des Antriebssystems (Helm) zu steuern und zu überwachen. Sämtliche Kommunikation und Koordination zwischen diesen Modulen erfolgt durch ein zentrales Blackboard-System namens CODGER (Communication database with geometric reasoning). Insbesondere werden durch Codger parallele Aktivitäten im Bereich der Ausführung und Kommunikation unterstützt. Die Koordination bzw. Auswertung der Information des Multisensorsystems (sensor fusion) erfolgt ebenfalls blackboardgesteuert.

Der funktionsbasierte Ansatz hat einen starken Anreiz hauptsächlich wegen der Garantien und Optimierungen, die er ermöglicht. Es gibt Planungsstrategien, die in einer endlichen Zeit eine Sequenz von Bewegungen, die garantiert zum Ziel führen, berechnen können oder zeigen können, daß die gegebene Aufgabe unlösbar ist. Außerdem können erfolgreiche Pläne optimiert werden, bevor der Roboter sich bewegt.

Der funktionsbasierte Ansatz hat allerdings einige Nachteile, besonders wenn die Umwelt veränderlich und unvorhersehbar ist. Für die meisten Anwendungen, die von praktischer Nützlichkeit sind, scheinen diese Systeme zu komplex zu werden. Sie erfordern größere

Mengen von Speicher sowie intensive Berechnung.

Um in der Lage zu sein, globale Information zu benutzen und sich mit verläßlichen Plänen zu befassen, muß die Darstellung der Umwelt (Weltmodellierung) vollständig und sehr genau sein. Bei dem Versuch, ein solches Modell zu konstruieren, treten aber viele Probleme auf: Erstens, die reale Welt ist voller Details, und Methoden für ihre Repräsentation brauchen viel Speicher. Zweitens, Sensordaten sind unvermeidlich verrauscht und Sensoren sind systematischen Fehlern unterworfen. Dies führt dazu, daß Sensorfusion (die Kombination von Daten aus mehreren Sensoren zu einer Datenstruktur) benutzt wird, und dies ist in der Regel sehr rechenintensiv. Die meisten Systeme versuchen daher, den Speicher und die Manipulation des Weltmodells praktikabel zu machen, indem sie die Umwelt vereinfachen. Dies führt unglücklicherweise aber zur Einschränkung der Autonomie des Roboters.

Die Arbeitsweise in einem funktionsbasierten Systems ist meistens sequentiell [74]. Der Roboter macht zuerst eine Momentaufnahme von der Umwelt, dann bearbeitet er die erworbene Information, und dann agiert er. Wenn die Umwelt sich zwischen Momentaufnahme und Aktion ändert, dann kann der Plan ungültig werden. Der Versuch, die Aktionen eines solchen Programms intelligenter zu machen, kann zu unzufriedenstellenden Ergebnissen führen. Je mehr Zeit das Programm der Auflösung von Konflikten zwischen den Sensordaten, der Verfeinerung des Weltmodells und der Optimierung des Plans widmet, desto länger wird die Verzögerung zwischen Sensoren und Aktoren. Diese Verzögerung erhöht die Wahrscheinlichkeit, daß eine signifikante Änderung in der Umwelt stattfindet, und daß der Plan somit ungültig wird. Perzeption und Aktion sind so ineinander verflochten, daß sie in Verbindung miteinander studiert werden müssen. Es ist eine Verschwendung von Leistung, anzuhalten und die ganze Umwelt zu katalogisieren, bevor irgendeine Aktion ausgeführt wird: Nur die Aspekte der Umwelt, die für die Aufgabe relevant sind, brauchen identifiziert zu werden.

3.3 Verhaltensbasierte Architekturen

Durch die Erkenntnis, daß eine zentrale symbolische Repräsentation die Quelle von vielen Problemen bei funktionsbasierten Architekturen ist, haben einige Forscher versucht,

effizientere Alternativlösungen zu finden. Mitte der achtziger Jahre entstand der reaktive verhaltensbasierte Ansatz. Der reaktive Ansatz sollte besonders gut funktionieren in Umgebungen, die teilweise unbekannt sind und in denen die Sensordaten verrauscht und unverläßlich sind und in denen das Operieren in Realzeit erfolgen soll und unerwartete Ereignisse berücksichtigen muß. Während klassische funktionsbasierte Systeme auf der Existenz und Manipulation eines globalen Weltmodels basieren, vertreten reaktive verhaltensbasierte Systeme die andere extreme Ansicht, daß ein solches internes Modell unnötig ist (In der Literatur über reaktive Systeme findet man oft folgende Aussage: ,,Die reale Welt dient als ihr eigenes Modell"). Stattdessen wird jede Aktion als direkte Reaktion auf einen wahrgenommenen Stimulus interpretiert. Obwohl dies auf den ersten Blick einfach erscheint, ist die erreichte Leistung beachtenswert und wächst, wenn mehrere reaktive Komponenten miteinander verbunden werden und innerhalb des gleichen Systems benutzt werden. Die Interaktionen zwischen solchen reaktiven Komponenten können zu Ergebnissen führen, die einem Betrachter als durchdachte Tätigkeit oder sogar als vernünftiges, logisches Denken erscheinen (siehe [28]). Der Erfolg des reaktiven Ansatzes hat gezeigt, daß komplexes, quasi-intelligent scheinendes Verhalten erzeugt werden kann, ohne von Speicher und Modell der Umgebung Gebrauch zu machen. Ein oft zitiertes Beispiel eines biologischen reaktiven Systems ist die Ameise (siehe [147]). Ihre Bewegungen scheinen komplex zu sein, aber die reale Komplexität ist die der Umgebung. Die Scheinkomplexität des Verhaltens der Ameise ist das Resultat einer kleinen Menge von Antworten zu einer veränderlichen Umgebung.

Die Basiskomponente eines reaktiven Systems ist ein Prozeß, der Perzeption und Aktion direkt verbindet. Dieser Prozeß wird oft *Verhaltensmuster* (task-achieving behavior) genannt. Ein Verhaltensmuster definiert die Reaktion des Systems auf bestimmte Umweltsituationen. Ein Verhaltensmuster kann z.B. für die Vermeidung von Kollisionen zwischen einem Roboter und Objekten in seiner Umgebung oder für die Haltung des Gleichgewichts des Roboters zuständig sein. Ein Verhaltensmuster kann mit Hilfe vieler verschiedener Datenverarbeitungstechniken wie neuronale Netze, fuzzy Logik, klassische Kontrollalgorithmen oder jede Kombination davon implementiert werden. Das Gesamtsystem wird durch die Menge der Verhaltensmuster charakterisiert. Der aktuelle Zustand des Gesamtsystems ist durch die Menge der momentan aktiven Verhaltensmuster definiert. Oft bilden die Verhaltensmuster ein Netzwerk. Zur Koordination des Gesamt-

systems verfügt jedes Muster über die Möglichkeit, andere Muster zu aktivieren bzw. den Einfluß aktiver Muster auf das Gesamtverhalten abzuschwächen.

Einen wesentlichen Beitrag zu reaktiven Systemen leistete Brooks [19, 24, 25], der die folgenden Prinzipien für seine Arbeiten als grundlegend ansieht:

- Intelligentes Verhalten kann ohne zentrale symbolische Repräsentation und das darauf operierende Reasoning generiert werden.
- Die Intelligenz muß in der realen Welt *situiert* sein (und nicht in unverkörperten Systemen wie Theorembeweiser): Roboter sollen nicht mit Hilfe abstrakter Beschreibungen ihrer Umwelt arbeiten, sondern das ,,Hier" und ,,Jetzt" ihrer Umwelt soll ihr Verhalten bestimmen.
- Intelligenz ist eine *emergente* Eigenschaft (emergent property), die aus der Interaktion mit der Umwelt und ,,in den Augen des Betrachters" entsteht.

Brooks schlägt eine verhaltensorientierte Zerlegung der Roboterfunktionen vor und bildet eine Architektur (Subsumption Architecture) aus Kontrollschichten, die Ebenen von Verhaltensmustern, Kompetenzebenen (levels of competence) genannt, entsprechen (siehe Abbildung 3.6). Jede Kompetenzebene wird, entsprechend ihrer Verhaltensmuster,

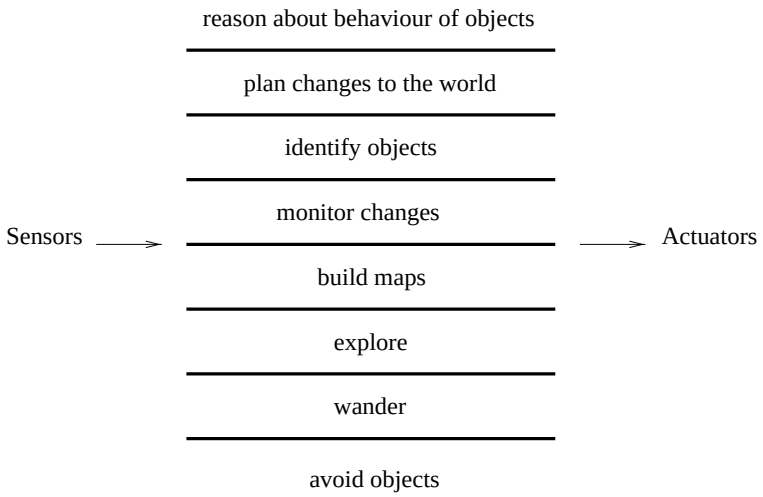


Abbildung 3.6: *Zerlegung des Steuerungssystems eines mobilen Roboters auf der Basis von Verhaltensmustern (task-achieving behaviors).*

durch ein Netz von mit Timern und Registern erweiterten endlichen Automaten realisiert. Die Automaten (siehe Abbildung 3.7 [21]) Kommunizieren durch Austausch von

Nachrichten. Die Ankunft einer Nachricht oder eines Timer-Signals verursachen Zustandsübergänge in den entsprechenden Automaten. Einige Automaten sind direkt mit

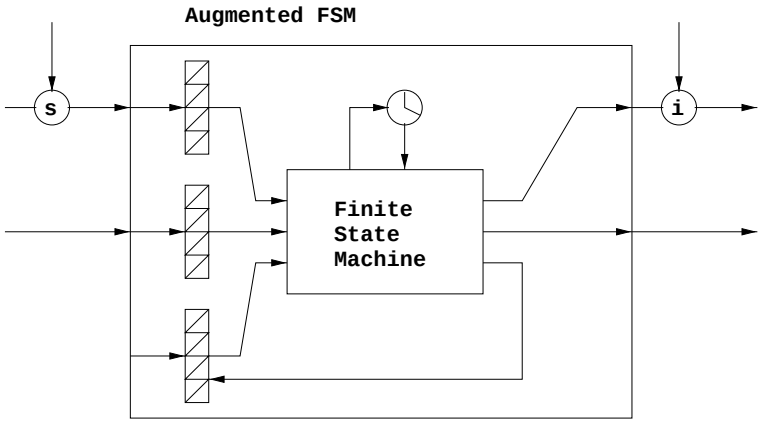


Abbildung 3.7: *Ein erweiterter endlicher Automat der „Subsumption Architecture" bestehend aus Registern, Timern, einem kombinatorischen Netzwerk und einem gewöhnlichen endlichen Automaten.*

der Roboterhardware verbunden. Sensoren schreiben ihre Werte in die Register und Ausgangsleitungen leiten Befehle zu den Aktoren. Innerhalb einer Ebene gibt es keine zentrale Kontrolle. d.h. jedes Modul arbeitet zunächst unabhängig von den anderen. Zur Koordination, auch zwischen den Ebenen, besteht nun die Möglichkeit, die Ein- bzw. Ausgänge bestimmter Module temporär inaktiv zu schalten. Die Interaktion zwischen den Verhaltensmustern sichert die Reaktivität. Ein Verhaltensmuster aus einer höheren Ebene wie z.B. „Navigieren" wird beispielsweise sofort abgebrochen, wenn ein Hindernis den Bewegungspfad sperrt. Es wird zum Verhaltensmuster für Hindernisvermeidung gewechselt. Abbildung 3.8 zeigt die einzelnen Kontrollschichten des Steuerungssystems des mobilen Roboters Mobot des MIT-Labors [20] und wie sie aus Netzen von erweiterten Automaten realisiert wurden. Die Entwicklung des Gesamtsystems kann stufenweise durch das Hinzufügen neuer Kompetenzen erfolgen. Eine höhere Ebene enthält als Untermenge sämtliche hierarchieniederen Ebenen und definiert somit zusätzliche Beschränkungen für das Gesamtverhalten des Systems. Niedere Ebenen bestehen aus reaktiven, instinktiven Verhaltensmustern wie z.B. die Vermeidung von Hindernissen oder das Wandern. Höhere Ebenen bestehen aus Verhaltensmustern wie z.B. die Identifizierung von Objekten oder das Ziehen von Schlußfolgerungen über die Bewegung von

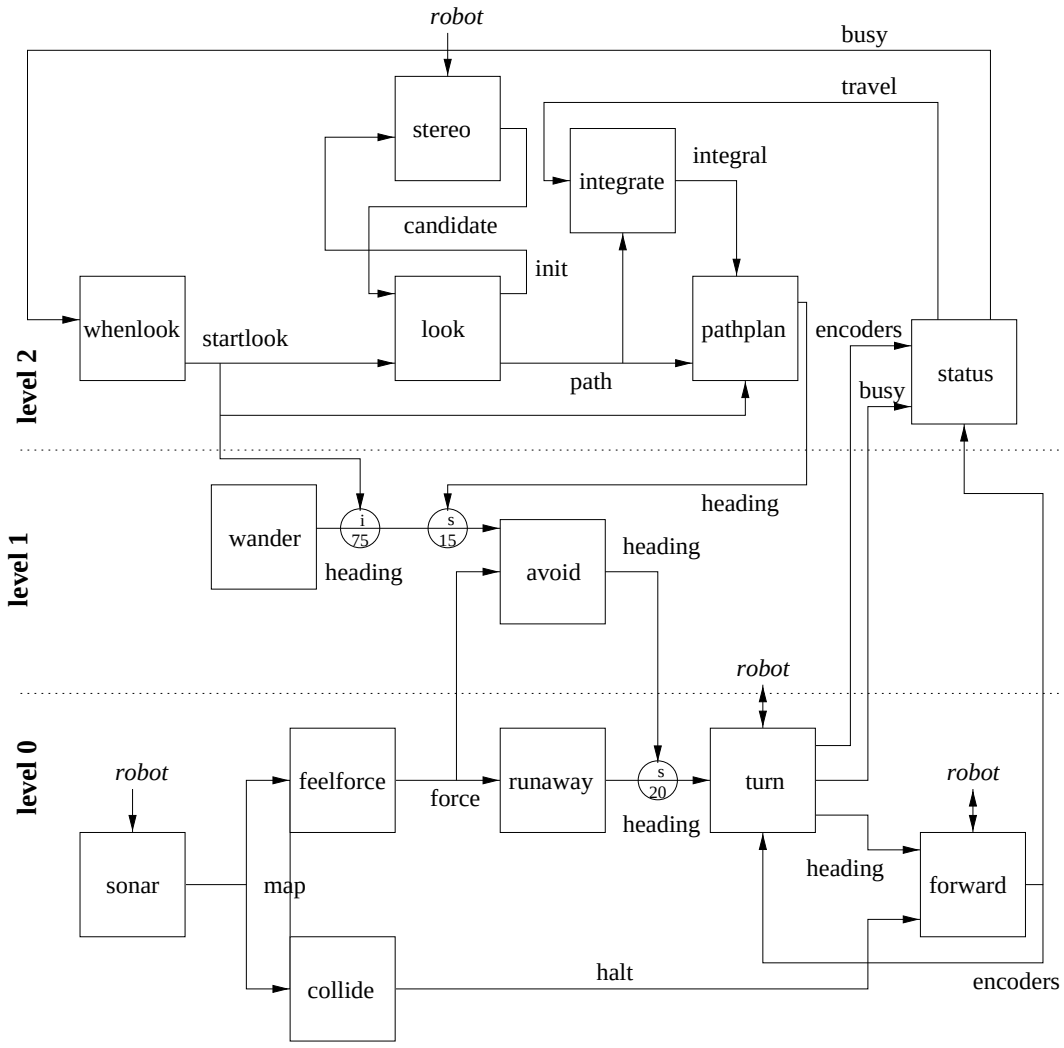


Abbildung 3.8: *Steuerungsstruktur des Mobot-Systems.*

Objekten.

Die Konsequenzen dieser prinzipiellen Überlegungen für das Design konkreter Kontrollarchitekturen von Robotern werden in [19, 22] besprochen. Wichtige Punkte sind dabei:

- Zerlegung einer Aufgabe in eine Reihe paralleler Module (Verhaltensmuster), die ihrerseits vollkommen unabhängig arbeiten, also mit sensorischen sowie motorischen Fähigkeiten ausgestattet sind.
- Jedes dieser Verhaltensmuster wird intensiv getestet und muß seine robuste Anwendbarkeit bewiesen haben. Danach wird es nicht mehr modifiziert, sondern es

werden bei Bedarf weitere Verhaltensmuster neu hinzugefügt.

- Es gibt keine zentrale Kontrollinstanz und keinen globalen Plan, sondern nur ein Aktivierungsprinzip, das dafür sorgt, daß das jeweils kompetenteste Modul die Aktivität der anderen unterdrückt und das aktuelle Verhalten prägt, wobei aber unter veränderten Umweltbedingungen wieder andere Verhaltensweisen die Kontrolle übernehmen können.

Vorteile der ,,Subsumption Architecture" sind vor allem die Einfachheit und die Schnelligkeit. Die Modellierung der realen Welt, die eine sehr schwierige Aufgabe ist, entfällt. Die Programme für die Steuerung von solchen Robotern sind somit einfach genug, daß sie leicht in Hardware implementiert werden können. Die Robustheit angesichts unsicherer sensorischer Information in einer prinzipiell unvorhersagbaren und sich laufend ändernden Welt wird möglich. Insbesondere bewirkt der Ausfall eines Verhaltensmusters in der Regel nur eine Leistungsminderung, das Gesamtsystem arbeitet entsprechend den Fähigkeiten der verbliebenen Verhaltensmuster weiter.

Auf der anderen Seite, ist es nicht klar, ob die ,,Subsumption Architecture" in der Lage ist, komplexe Planungsprobleme zu lösen. Roboter, die auf dieser Grundlage gebaut wurden, scheinen, im Vergleich zu klassischen Problemlösern, die in der Lage sind eine breite Vielfalt von Tasks behandeln zu können, einen Mangel an Flexibilität zu haben. Das Kontrollsystem ist festverdrahtet und somit sehr schwer änderbar. Der Roboter ist nicht programmierbar und beschränkt sich auf die Verhaltensmuster, die der Designer implementiert hat [117]. Das Problem der Arbitrierung (Wahl der passenden Verhaltensmuster) stellt sich hauptsächlich bei verhaltensbasierten Architekturen. Jedes Verhaltensmuster hat nur eine lokale Sicht der Umgebung. Um gute globale Resultate zu erzielen, müssen die Verhaltensmuster kooperieren und ihre Aktionen koordinieren. Das Problem der Arbitrierung zwischen Verhaltensmustern kann allerdings besser gelöst werden in Systemen, die einiges zentrales Wissen besitzen. Für ein System mit einem Planer unterliegen die Tasks oft einer partiellen Ordnung und die Arbitrierung ist somit nur in speziellen Fällen nötig.

Neben Brooks gab es auch viele andere Forscher, die die Schwierigkeiten der klassischen symbolischen KI erkannt haben und sich für reaktive Systeme interessierten. Agre und Chapman [28], zum Beispiel, haben die Beobachtung gemacht, daß die meiste alltäg-

liche Aktivität, in dem Sinne, daß sie nur wenig oder gar kein neues abstraktes Denken braucht, ,,Routine" ist. Die meisten Aufgaben können, einmal gelernt, routinemäßig mit wenig Abweichung erledigt werden. Routinemäßige Entscheidungen können in eine low-level Struktur (z.B. digitale Schaltung) codiert werden. Um neue Arten von Problemen bearbeiten zu können, braucht diese Struktur nur periodisch aktualisiert zu werden. Diese Idee wurde durch das simulierte Computerspiel PENGI [1] illustriert. Payton [119] hat eine Kollektion von Verhaltensweisen, ,,reflexive Verhalten" (reflexive behaviors) genannt, beschrieben. Diese Verhaltensweisen reagieren direkt auf sensorische Informationen und führen zu einem intelligent scheinenden Gesamtverhalten. Arkin [6] entwickelte motorische und perzeptuelle Schemata. Ein Schema kann als ein Rechenagent angesehen werden. Für jedes aktivierte motorische Schema werden perzeptuelle Schemata kreiert. Diese Schemata werden dazu benutzt, um Ereignisse zu erkennen, die ein neues, mit den Anderen konkurrierendes, motorisches Schema auslösen. Um einen hohen Grad an Parallelität zu erreichen, kommunizieren die Schemata durch ein Blackboard-System. Kaelbling [76] hat eine reaktive Architektur entwickelt, die eine Erweiterung der Arbeit von Brooks darstellt. Der Schwerpunkt liegt dabei auf eingebetteten Systemen für Realzeitsteuerung. Es werden Verhaltensmodule benutzt, aber es gibt keine Verbindungen zwischen ihnen. Die Verwaltung der Verhaltensmodule geschieht mit Hilfe von Vermittlern (mediators), die die Ausgänge der Verhaltensmodule, abhängig von internen Bedingungen, ausschalten können. Firby [42] benutzt Module, RAPs (Reactive Action Packages) genannt, die Tasks für den Roboter verkapseln. Sie laufen bis das Ziel erreicht ist oder bis sie scheitern. Ein RAP-Interpreter verwaltet die Aktivierung und Konfliktauflösung.

Das verhaltensbasierte Paradigma hat insgesamt viele wünschenswerte Eigenschaften: Keine Engpässe wegen zentraler Repräsentation der Umwelt, Minimierung von Zeitverzögerungen zwischen Sensorik und Motorik, Fehlertoleranz und niedrige Implementierungskosten. Das Design der einzelnen Schichten sowie der Schnittstellen zwischen den Schichten scheint aber ohne feste Grundsätze zu sein. Auch das Lösen von Aufgaben, die viel Planung erfordern, ist schwierig. Man muß zwar vorsichtig sein, wenn man behauptet, daß eine Aufgabe nur durch die Benutzung von internen Weltmodellen gelöst werden kann, aber es existieren grundsätzliche Einschränkungen für die Fähigkeiten von Systemen, die keine Informationen über ihre Umgebung enthalten. Selbst solche Aufga-

ben, die nur durch Reflexaktionen erledigt werden können, können durch die Benutzung von Weltmodellen manchmal effizienter und effektiver gelöst werden.

3.4 Hybride Architekturen

Da sowohl reaktive als auch deliberative Ansätze Stärken und Schwächen aufweisen, sind verschiedentlich hybride Systeme vorgeschlagen worden, die die Robustheit reaktiver Systeme mit den höheren kognitiven Leistungen deliberativer Systeme, etwa der Planungs- oder Kommunikationsfähigkeit, verbinden (siehe z.B. [50, 106, 7, 48, 40, 146, 110]).

Durch die Koppelung eines symbolisch arbeitenden, deliberativen Systems und eines situierten, verhaltensorientierten, reaktiven Systems wird gehofft, daß beide Teilsysteme die Vorteile des jeweils anderen nutzen und einen Großteil der eigenen Nachteile vermeiden. Durch Hybridizität soll das reaktive System an Deliberativität und das deliberative System eine Schnittstelle zu einer idealen Welten ohne Unsicherheiten gewinnen. Das resultierende System soll flexibel und robust sein und die Fähigkeit haben, sein Ziel zu erreichen und seine Aufgabe zu erledigen. Die deliberative Komponente ist in einer solchen hybriden Architektur meistens für die Verfolgung von mehr oder weniger längerfristigen Zielen verantwortlich und realisiert die notwendige Sequentialisierung von Steuerungsparametern komplexer Handlungsfolgen. Das verhaltensbasierte System ist für die situativ richtige und zeitadäquate Umsetzung der eingehenden Sensordaten in Aktuatorbefehle verantwortlich. Die einzelnen Routinen der Module bilden die für sie relevanten Sensordaten unmittelbar auf Aktuatorkommandos ab.

Ein Beispiel für eine hybride Architektur ist die InteRRaP-Architektur [110]. Die InteRRaP-Architektur (siehe Abbildung 3.9) enthält drei Kontrollschichten: Die verhaltensbasierte Schicht (Behavior-based Layer, BBL), die Schicht der lokalen Planung (Local Planning Layer, LPL) und die Schicht der kooperativen Planung (Cooperative Planning Layer, CPL). Der reaktive Teil der Architektur ist durch die BBL-Schicht, die eine Menge von Situation-Aktion-Regeln enthält implementiert. Diese Regeln beschreiben die reaktiven Fähigkeiten des Agenten und ermöglichen die Implementierung von schneller Situationserkennung, um auf zeitkritische Situationen reagieren zu können. Während die mittlere LPL-Schicht lokales zielorientiertes Verhalten implementiert, gibt

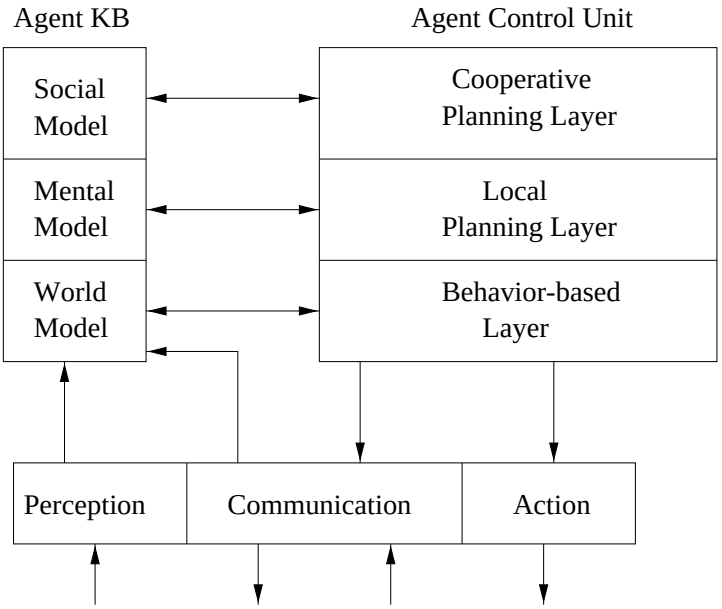


Abbildung 3.9: *Die hybride Architektur InteRRaP.*

die oberste CPL-Schicht den Agenten die Möglichkeit zu planen und mit anderen Agenten zu kooperieren, um Multi-Agent-Pläne zu erledigen und Konflikte zu lösen. Die LPL-Schicht und CPL-Schicht ermöglichen mehr Deliberativität. All diese Schichten arbeiten asynchron und benutzen verschiedene Modelle (Weltmodell, mentales Modell, soziales Modell) aus der Wissensbasis des Agenten. Die InteRRaP-Architektur wurde durch die Konstruktion einer Anwendung evaluiert, die einen in einer automatisierten Ladeumgebung arbeitenden Gabelstapler-Roboter simuliert.

Eine weitere Architektur, die die Idee von reaktiven Systemen mit herkömmlichen KI-Planern kombiniert, ist TCA (Task Control Architecture) [146]. TCA ist ein verteiltes System mit zentralisierter Kontrolle, das zur Steuerung von autonomen Robotern über längere Zeiträume, in unstrukturierten Umgebungen wie der Oberfläche des Mars, entworfen wurde. TCA behandelt besonders Fragen, die im Kontext von mehrfachen Zielen (goals) und begrenzten Betriebsmitteln entstehen. Die Architektur stellt Mechanismen für hierarchisches Task-Management bereit und erlaubt Aktion auf der Basis von unvollständigen Plänen. Weil Roboter durch die Bewegung in der Umwelt neue Informationen gewinnen, erlaubt TCA den frühzeitigen Abbruch von Plänen, wenn Ziele mit höherer Priorität entstehen. Einige Situationen erfordern hochreaktives Verhalten. TCA

realisiert schnelle Antwortzeiten, wann immer möglich, durch die Parallellisierung von Planung und Ausführung. Bei dem Entwurf von Gehbewegungen in unebenem Gelände zum Beispiel, plant TCA einen Schritt, leitet ihn ein, und beginnt dann die Planung des nächsten Schrittes, bevor die Beinbewegung abgeschlossen ist.

Ein anderes Programm für die Kombination von heuristischem Problemlösen mit Reaktivität heißt THEO-Agent [106]. THEO-Agent enthält zwei Subsysteme, ein reaktives Subsystem und einen allgemeinen Problemlöser, THEO [107] genannt. Wenn das reaktive Subsystem keine Vorgehensweise vorschlagen kann, erzeugt der Problemlöser einen Plan für den Roboter. Während der Planausführung benutzt der Roboter eine Lernkomponente, um neue reaktive Module zu erzeugen. Der Roboter gewinnt somit mit der Zeit (Erfahrung) zunehmend an Reaktivität.

PRS (Procedural Reasoning System) [50] ist ein symbolisches Roboterplanungssystem, das Planung und Ausführung verschachtelt. In PRS stellen Ziele (goals) nicht, wie in klassischen Planungssystemen üblich, Weltzustände, sondern Roboterverhaltensweisen dar. PRS enthält Prozeduren, die Ziele in Teilziele (subgoals) oder Iterationen davon umwandeln. Eine Prozedur wird entweder durch das Vorliegen eines Ziels oder durch das Eintreffen einiger Sensordaten aufgerufen. Der Roboter sollte somit in der Lage sein, sich zielorientiert zu verhalten und schnell reagieren zu können, wenn die Umwelt sich ändert oder wenn der Plan fehlschlägt. Ziele und Prozeduren werden symbolisch repräsentiert. Ein zentraler Problemlöser benutzt einen Stack, um die Prozeduraufrufe zu verwalten.

Weitere Beispiele für hybride Architekturen stellen die Arbeiten von Arkin [7] und Gat [48] dar. Arkin vertritt die Ansicht, daß in Architekturen für Autonome mobile Roboter beide, deliberative (hierarchische) und reaktive (Schema-basierte) Steuerungssysteme benutzt werden sollten. In [7] wird gezeigt, wie durch die Einbindung eines herkömmlichen Planers, der mit einem flexiblen und modularen reaktiven Steuerungssystems arbeitet, spezielle Robotik-Konfigurationen konstruiert werden können, die Verhaltenswissen, Perzeptualwissen und a priori vorhandenes Wissen über die Umwelt integrieren. In [48] schlägt Gat ein System (Atlantis) aus drei Schichten vor, das aus einem Lisp-Deliberator, einem Sequenzer zur Behandlung von Fehlern des reaktiven Systems und einem reaktiven Kontroller besteht. Das System wurde auf Prototypen des Roboters „Mars Rover" getestet.

Kapitel 4

LERNANSÄTZE ZUR STEUERUNG VON ROBOTERN

Intelligenz von Maschinen war immer an ihrer Lernfähigkeit gebunden. Eines der Kriterien, das oft mit Intelligenz in Beziehung gebracht wird, ist, daß Maschinen nicht als intelligent bezeichnet werden können, bis sie in der Lage sind, anstatt nur das zu tun, was man ihnen beigebracht hat, zu lernen, wie sie neue Aufgaben erledigen und wie sie sich an neuen Situationen anpassen können. Es ist klar, daß die Fähigkeit der Anpassung an neuen Umgebungen, sowie die Fähigkeit zur Lösung neuer Probleme wichtige Eigenschaften von intelligenten Systemen sind. Die Frage ist, ob Computerprogramme solche Fähigkeiten haben können. Maschinelles Lernen ist eines der ältesten Forschungsziele der künstlichen Intelligenz, welches sich mit der Beantwortung dieser Frage beschäftigt. Bereits seit dem Gelingen der ersten praktischen Rechnereinsätze gehört das maschinelle Lernen zu den Fähigkeiten, deren Verfügbarkeit auf einem Rechner als Ziel formuliert wurde. Die Idee dabei war, daß Programmierer von Routinearbeiten entlastet und Programme schneller erstellt werden sollten. Für Alan Turing war die Lernfähigkeit eines Rechners die wichtigste Intelligenzleistung [161]. Er Empfahl, einen Rechner ,,zu erziehen", so daß er seine Leistungen verbessert, da man unmöglich alles einprogrammieren könne.

In diesem Kapitel werden zuerst das Lernen im allgemeinen, sowie das Lernen im Zusammenhang mit Robotern erläutert. Danach wird ein Überblick über das Lernen mit neuronalen Netzen, sowie Reinforcement Lernen gegeben. Diese Methoden haben in den letzten Jahren in dem Gebiet der Robotik verstärkt Anwendung gefunden. Sie werden im restlichen Teil dieser Arbeit erweitert, kombiniert und bei der Entwicklung der Steue-

rungsarchitektur dazu benutzt, um Adaptivität und Selbstverbesserung zu ermöglichen. Mehr Details und ausführlichere Beschreibungen können in der gegeben Literatur gefunden werden.

4.1 Was ist Lernen?

Die erste Frage, die meist gestellt wird, ist, wie wir Lernen definieren können, so daß wir einem Rechner eine - eingeschränkte - Lernfähigkeit zusprechen können. Die bekannte Definition von Simon [148] lautet:

> Lernen bezeichnet Veränderungen in einem System, die adaptiv sind in dem Sinn, daß sie dem System erlauben, dieselbe Aufgabe oder Aufgaben derselben Klasse das nächste Mal effizienter oder effektiver zu bewältigen.

So wie oben definiert, ist Lernen ein sehr weiter Begriff und umfaßt eine große Anzahl von Phänomenen. Auf der einen Seite des Spektrums steht *die Adaptierung von Verhalten* (skill refinement). Einfach durch Übung bekommt man viele Aufgaben besser in den Griff. Je mehr man zum Beispiel Fahrrad fährt oder Tennis spielt, desto besser wird man. Auf der anderen Seite des Spektrums liegt *die Aufnahme von Wissen* (knowledge acquisition). Wissen wird häufig aus der Erfahrung gewonnen.

Die Aufnahme von Wissen selbst umfaßt viele Aktivitäten. Die einfache Speicherung von berechneter Information (auswendig lernen) ist die grundlegendste Lernart. Viele Computerprogramme, wie zum Beispiel Datenbanksysteme, können in diesem Sinn als lernend bezeichnet werden, obwohl die meisten Leute diese einfache Speicherung nicht als Lernen bezeichnen würden. Es existieren aber einige Programme, die in der Lage sind, ihre Leistung mit Hilfe von Techniken des Auswendiglernens wesentlich zu verbessern (siehe z.B. [143]).

Eine andere Lernart ist das Lernen durch Unterweisung. Zugrunde liegt unvollständiges Grund- und Detailwissen, das für das Ausführungselement nicht ausreicht. Dem Lernelement werden abstrakte Informationen in Form von Anweisungen oder Regeln übergeben. Die Transformation dieses abstrakten Wissens in eine konkrete, für das Ausführungselement brauchbare Form, kann manchmal eine Operationalisierung erfordern. Das System

muß dabei die abstrakte Eingabeinformation interpretieren und in seine bereits vorhandene Wissensstruktur eingliedern.

Menschen lernen auch durch Experimentieren. Nachdem ein komplexes Problem gelöst wurde, erinnern wir uns an die Struktur des Problems und an die Methode, die wir benutzt haben, um es zu lösen. Das nächste Mal, wenn wir mit dem Problem konfrontiert werden, können wir es effizienter lösen. Ferner können wir aus unserer Erfahrung generalisieren, um ähnliche Probleme auf eine einfachere Art und Weise zu lösen. Im Gegensatz zum Lernen durch Unterweisung, wird beim Lernen durch Experimentieren das Lernelement nicht direkt mit neuen Informationen versorgt, die vorher nicht verfügbar waren. Dies bedeutet, daß das Lernprogramm sich an seine Erfahrungen erinnert und aus ihnen generalisiert. Das Programm erweitert aber nicht die transitive Hülle seines Wissens (alles was man durch Generalisierung erreichen kann), wie es beim Lernen durch Unterweisung durch den Empfang von Anweisungen von außen der Fall ist. Dennoch, Effizienssteigerung spielt bei der Lösung von größeren Problemen eine ganz entscheidende Rolle. Lernen kann in der Praxis bedeuten, daß Probleme, für die es noch keine Lösungen gibt, sich doch als lösbar erweisen. Außerdem kann ein Programm, das durch Experimentieren lernt, in der Zukunft qualitativ bessere Lösungen erzeugen.

Eine andere Form des Lernens, die Anweisungen von außen involviert, ist das Lernen aus Beispielen (auch induktives Lernen). Menschen sind oft in der Lage zu lernen, Sachen zu klassifizieren, ohne daß explizite Regeln zur Verfügung stehen müssen. Erwachsene, zum Beispiel, können Katzen von Hunden unterscheiden. Kleine Kinder dagegen sind oft nicht in der Lage, dies zu tun. Irgendwann im Laufe der Zeit aber, wenn sie mehrere Beispiele von beiden Arten gesehen haben, entwickeln sie Methoden für die Unterscheidung. Lernen aus Beispielen erfordert oft einen Lehrer (teacher), der uns dabei hilft, Dinge zu klassifizieren, indem er uns korrigiert, wenn wir Fehler machen. Manchmal kann ein Programm jedoch einige Dinge entdecken, ohne die Hilfe des Lehrers in Anspruch zu nehmen.

Während in den sechziger und siebziger Jahren erste einfache maschinelle Lernverfahren unter kognitionswissenschaftlichen, erkenntnis- und rekursionstheoretischen Gesichtspunkten entwickelt und analysiert wurden, fand insbesondere im letzten Jahrzehnt, bedingt durch die Entwicklungen auf dem Gebiet der ,,wissensbasierten Systeme", eine

stärkere Orientierung an der Wissensakquisitionsproblematik statt. Mittlerweile findet das maschinelle Lernen nicht nur im wissenschaftlichen Kontext Interesse, selbst im Rahmen realer Anwendungen wurden einige Lernverfahren bereits erfolgreich eingesetzt (siehe z.B. [136]).

Maschinelles Lernen ist nicht zwangsläufig an die Entwicklung von wissensbasierten Systemen und die Teilautomation des Wissenserwerbsprozesses gekoppelt. Häufig können maschinelle Lernverfahren als Unterstützung traditioneller statistischer Analysemethoden, zur Inspektion und Zusammenfassung von Daten, zur Aufdeckung von bisher unbekannten Zusammenhängen in Daten (data mining [69]), zur Entwicklung von Werkzeugen zur Entscheidungsunterstützung oder zur Unterstützung des Softwarekonstruktionsprozesses eingesetzt werden.

Die Lernergebnisse können hierbei von der Justierung von Parametern, der inhaltlichen Zusammenfassung von Daten, deren Gruppierung und Klassifizierung, von der Aufdeckung statisch signifikanter Regelhaftigkeiten, über „Faustregeln" für Nicht-Experten und als Anregungen für einen Wissensingenieur, bis hin zur Anpassung und Spezialisierung allgemeineren Wissens an konkrete Umgebungsbedingungen reichen.

Eine Reihe maschineller Lernverfahren steht mittlerweile zur Verfügung, die für unterschiedliche Problemstellungen konstruiert wurden und über unterschiedliche Eigenschaften verfügen. Ein allgemeiner Überblick über Lernen sowie eine Klassifizierung von Lernverfahren können z.B. in [109, 131, 38] gefunden werden. Eine Zusammenfassung von Kriterien zur Beschreibung und Auswahl maschineller Lernverfahren findet man in [71].

4.2 Lernen im Zusammenhang mit Robotern

Lernen im Zusammenhang mit Robotern kann auf zahlreiche Lernziele auf unterschiedlichen Abstraktionsebenen innerhalb eines Robotersystems bezogen werden. In jedem Fall liegen dabei Vorstellungen zugrunde, die eine Anpassung, eine Verfeinerung, eine Optimierung oder sogar eine Erweiterung der Systemfähigkeiten durch das System selbst vorsehen. Lernen im Sinne eines lernenden Steuerungssystems wird nach Fu [45] und Saridis [144] wie folgt definiert:

> Ein System wird als lernend bezeichnet, wenn es in der Lage ist, unbekannte Eigenschaften eines Prozesses oder seiner Umgebung durch schrittweises Handeln und Beobachten zu erfassen. Die dadurch gewonnene Erfahrung wird benutzt, um Vorhersagen, Klassifikationen und Entscheidungen durchzuführen, damit ein vorgegebenes optimales Systemverhalten erreicht werden kann. Ein lernendes System wird als lernendes Steuerungssystem bezeichnet, wenn die durch schrittweises Handeln und Beobachten erfaßte Information benutzt wird, um einen Prozeß mit unbekannten Eigenschaften zu steuern. Steuern durch Lernen kann off-line durch Training oder on-line nach den Prinzipien sich selbst organisierender, adaptiver Systeme erfolgen.

Die Anwendung von Methoden des maschinellen Lernens hat insbesondere dort in der Robotik Bedeutung, wo die Modellbildung von Prozessen entweder zu komplex, fehlerhaft oder nicht ausreichend ist, um ein korrektes ausführbares Roboterprogramm mit entsprechenden Kontrollstrukturen zu erzeugen. Wieweit die Anwendung von Lernverfahren in der Robotik effiziente Lösungen ermöglicht, ob sie klassischen analytischen Methoden überlegen ist oder sie ergänzt, wird an Beispielen aus der realen Welt gemessen.

Bei der Betrachtung maschineller Lernstrukturen wird oft von hypothetischen Modellen der Lernstruktur bei höheren Lebewesen ausgegangen. Eine große Anzahl der frühen Arbeiten über Lerntheorien gehen von Verhaltensbeobachtungen von Tieren in synthetischen Situationen aus. Gegenstand der Beobachtungen sind dabei Beute- und Jagdverhalten, Nahrungsaufnahme, Flucht oder Schutz- und Sexualverhalten. Die synthetische Situation wird meist in einem unerwünschten Anfangszustand, wie z.B. Hunger, vorgegeben, das Ziel ist die Nahrungsaufnahme. Das Erreichen des Ziels wird durch Hindernisse erschwert.

Die Lösung des Problems wird zur Grundlage der Lerntheorie gemacht. Skinners bekannte Verhaltensstudien an Ratten [149] sind ein Beispiel hierzu. Lernen wird demnach als jede situations- und umgebungsbezogene Verhaltensänderung bezeichnet, die als Folge einer individuellen Informationsverarbeitung oder Problemlösung eintritt. In der Regel ist damit eine verbesserte Anpassung des Verhaltens an die bestehende Umwelteigenschaft oder deren Änderungen verbunden. Durch Modifikation der synthetischen Situati-

on in Tierversuchen konnte die These erhärtet werden, daß mit der Verhaltensänderung auch eine Korrektur einer Gedächtnisstruktur verbunden sein muß. Lernen kann also in der Ausbildung oder Korrektur von individuellem Gedächtnisbesitz definiert werden [78]. Zur Durchführung der Verhaltenskorrektur wird eine rezeptorische Informationsaufnahme aus der realen Umgebung erforderlich (sensorische Erfassung der Umwelt), sowie deren Umsetzung in höhere Wahrnehmungsformen (kognitive Leistung). Nach der Bewertung der Wahrnehmungen müssen umgebungsangepaßte motorische Verhaltenseinheiten aktiviert werden. Perzeptions-, Aktions- und Entscheidungsfähigkeiten sind damit elementare Voraussetzungen für die Korrektur des Verhaltens durch Lernen. Entscheidungsfähigkeit bedeutet dabei, die Auswahl zwischen verschiedenen Verhaltensalternativen. Führt das gewählte Verhalten nicht zum Ziel, muß die Gedächtnisstruktur modifiziert werden.

Ein wichtiger Aspekt dieses Lernmodells ist seine Bezogenheit auf die reale Umwelt. Aus der Umgebung werden Reizeinflüsse aufgenommen und nach deren internen Verarbeitung (Auswertung des Informationsgehalts) zur Verhaltensänderung, also einer Interaktion zwischen Organismus und Umwelt herangezogen. Die Ausbildung oder Korrektur von Gedächtnisbesitz erfolgt über einer komplizierten Speicherstruktur, die über eine große Anzahl von Kanälen, z.B. durch visuelle, verbale oder akustische Eindrücke, oder durch von Haut-, Gelenk- und Muskelrezeptoren gelieferten Eindrücken stimuliert wird. Die Lernleistungsfähigkeit setzt eine Flexibilität der umgebungsgemäßen Wahrnehmung, der umgebungsangepaßten Steuerung und der Anregung von Gedächtnisbesitz voraus. Die zu lösenden Problemtypen hängen jeweils von den aktuellen Bedürfniszuständen, wie z.B. Hunger oder Beuteverhalten ab. Bei jungen Tieren werden schon sehr früh Verhaltensstrukturen beobachtet, die angeboren oder triebgesteuert sind und erst noch durch Verhaltenskorrektur optimiert werden. In zahlreichen Versuchen hat diese Korrektur die Form des Lernens durch Versuch und Irrtum (trial and error). Wird eine falsche Verhaltensweise angewandt, führt die sensorische Rückmeldung des Resultats zu einer Registrierung einer fehlerhaften korrekturbedürftigen Gedächtnisstruktur, die auch die Bewertung der Verhaltensentscheidung beeinflußt. Wird dagegen eine richtige Reaktion oder Verhaltensweise aktiviert, führt die Rückmeldung zu einer Stabilisierung der relevanten Entscheidungs- und Gedächtnisstruktur (Verstärkungsprinzip). Die Beziehung zwischen Situationsmerkmalen, angepaßten Verhaltensweisen und ihrem Nutzen

stellt die wesentliche Leistung des Gedächtnisses in der Lernsituation dar. Die Wiedererkennung von Situationsmerkmalen mit der damit verbundenen Aktivierung einer angepaßten Verhaltensweise beinhaltet demnach einen kognitiven Aspekt. Der zielgerichtete Handlungsaspekt beinhaltet die Bewertung der Merkmale und die Gewichtung der Güte und des Nutzens der Entscheidung.

Bei höher organisierten Lebewesen und insbesondere beim Menschen können das situationsbedingte Verhalten und seine umgebungsbedingte Korrektur nicht immer auf einen einzigen Lernprozeß reduziert werden. Meistens liegt ein komplexes Verhaltens- und Problemlösungsspektrum vor, dessen einzelne Elemente sehr selten elementar und unverbunden einen Handlungsablauf steuern (siehe [129]). Die einzelnen verhaltensbestimmenden Elemente sind vielmehr Bestandteile von Handlungskonzepten, Strategien und Verarbeitungsprozeduren, die zur Bewältigung eines Problems kooperieren. In vielen Lernverfahren sind meist Kombinationen von solchen Problemlösungsprozessen wie Versuch/Irrtum-Strategien, Datenorganisation und Bedingungsvariation von Problemvariablen modelliert.

Die Ansätze zur Modellierung der oben skizzierten Lernsituation betonen besonders die Gedächtnisleistung zur Herstellung von Beziehungen zwischen Motivationsgrundlage, Situationsmerkmalen, angepaßten Verhaltensweisen, sowie ihrem effektiven Nutzen. Die aus den Disziplinen Verhaltenspsychologie und Neurophysiologie entwickelten Modelle approximieren jeweils Teilaspekte der betrachteten Lernsituation und des Lernapparats. Im folgenden werden einige Lernprozesse in ihrer reduzierten, modellhaften Form des maschinellen Lernens, nämlich Lernen mit künstlichen neuronalen Netzen und Reinforcement Lernen, etwas näher erläutert.

4.3 Robotik-Umgebungen

Bei der Formulierung von Lernkonzepten wird oft zuerst versucht, eine formale Charakterisierung der Aufgabe eines Roboters (Agenten) sowie seiner Umgebung vorzunehmen. Jeder Roboter wird oft als die Implementierung einer Menge von Abbildungen von aktuellen und vorherigen Sensorsignalen und Aktionen auf Aktionssignale beschrieben (siehe

z.B. [177, 133]). Im diskreten Fall, kann dies wie folgt formal beschrieben werden:

$$a_t = F(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t) \tag{4.1}$$

Dabei bezeichnet a_τ die Aktion des Roboters im Zeitpunkt τ. s_τ bezeichnet den Vektor der Sensorsignale, die vom Roboter zum Zeitpunkt τ empfangen werden. F ist eine Funktion, die im allgemeinen nicht deterministisch ist, d.h, F kann statt eine einzige Aktion eine ganze Menge von möglichen Aktionen liefern.

Da F eine beliebige Funktion sein kann, sind durch Gleichung (4.1) der Komplexität des Roboters keine Schranken gesetzt. Eine attraktivere dennoch keinesfalls wenig allgemeine Formulierung von Gleichung (4.1) ist durch die folgende Kombination von Gleichungen gegeben:

$$a_t = f(S_t) \tag{4.2}$$

$$S_t = g(S_{t-1}, a_{t-1}, s_t) \tag{4.3}$$

Gleichung (4.2) besagt, daß die Aktion des Roboters eine Funktion seines Zustands ist. Gleichung (4.3) bedeutet, daß der Zustand des Roboters eine Funktion des vorherigen Zustands, der vorherigen Aktion und der Aktuellen Sensordaten ist. Wenn die Funktion g einfach als die Hintereinanderreihung ihrer Argumente definiert wird und $f(S_t)$ durch einen Aufruf von F ersetzt wird, dann sind Gleichungen (4.2) und (4.3) mit Gleichung (4.1) identisch. Es ist jedoch hilfreicher, sich die nächste Aktion des Roboters als eine Funktion seiner vorherigen Aktion, der jetzigen Sensordaten und seines internen Zustandes vorzustellen.

Die Umgebung, in der sich der Roboter befindet, ist im allgemeinen dynamisch, d.h. unterliegt zeitlichen Änderungen. Auch hier besteht eine häufig gemachte Annahme und Idealisierung darin, daß die relevante Information von einer endlichen Menge von Variablen abhängt, die Funktionen der Zeit sind (Zustand). Die Umgebung interpretiert die Sequenz von Aktionen und generiert eine Sequenz von Sensorvektoren. Sie kann, ähnlich wie beim Roboter, wie folgt beschrieben werden:

$$s_t = f'(E_t) \tag{4.4}$$

$$E_t = g'(E_{t-1}, a_{t-1}) \tag{4.5}$$

E_t bezeichnet dabei den Zustand der Umgebung zum Zeitpunkt t. Genau so wie F, sind beide Funktionen f' und g' nicht deterministisch. Dies bedeutet, daß durch eine bestimmte Aktion in einem bestimmten Zustand verschiedene Zustände resultieren können, und daß im gleichen Zustand verschiedene Sensorvektoren erzeugt werden können. Die Gleichungen (4.2), (4.3), (4.4) und (4.5) definieren gemeinsam ein Protokoll, mit dem der Roboter mit seiner Umgebung interagiert. Der Roboter agiert aufgrund der Sensordaten, die er bekommt. Die Umwelt reagiert auf die Roboteraktionen. Dieser allgemeine Rahmen beschreibt eine Menge von Robotern, die in der Lage sind, ein breites Spektrum von Aufgaben in einem breiten Spektrum von Umgebungen zu verrichten. Die Funktionen f und g beschreiben, neben den Aktionen, die vom Roboter ausgeführt werden, implizit auch die Aufgabe (Task), die vom Roboter erledigt werden muß. Da die Gleichungen (4.2) und (4.3) die Komplexität des Roboters nicht einschränken, setzen sie auch keine Schranken für die Komplexität der gestellten Aufgabe. Oft ist es aber so, daß die Erledigung einer Aufgabe viel weniger Information verlangt als durch Gleichung (4.1) spezifiziert ist. Einige sehr einfache Aufgaben sind von keiner Art von Sensor- und Aktionsinformation abhängig. Viele andere Aufgaben, sogar solche, die Wissen über vorherige Aktionen verlangen, können erledigt werden, auch wenn g nur eine Funktion des Zustands und der aktuellen Sensordaten ist (d.h. F ist nur eine Funktion der Sensordaten). Der Roboter kann in diesem Fall z.B. spezielle Hardware benutzen, um die Aktionsinformation in Sensorinformation zu transformieren. Dies würde die explizite Abhängigkeit der Funktion g von a_{t-1} überflüssig machen.

Da a_t benutzt werden kann, um das Verhalten eines Agenten, der eine Aufgabe löst, zu beschreiben, kann a_t auch benutzt werden, um das gewünschte Verhalten eines Agenten, der *lernt*, die Aufgabe zu lösen, auszudrücken. Wichtig dabei ist das Lernen von Abbildungen: Abbildungen zwischen Sensordaten und Zuständen, Abbildungen zwischen Zuständen und Sensordaten, Abbildungen zwischen Zuständen und Aktionen, usw. Im folgenden wird dies am Beispiel des Lernens mit neuronalen Netzen und Reinforcement Lernen gezeigt.

4.4 Lernen mit neuronalen Netzen

Eine gute Möglichkeit, die im vorherigen Abschnitt erwähnten Abbildungen zu codieren und zu lernen, stellt ein neuronales Netz dar. In diesem Abschnitt wird eine kurze Einführung in das Gebiet der neuronalen Netze gegeben, um die wesentlichen Grundgedanken dieser Methode zu erläutern. Hierbei wird weniger auf theoretische Betrachtungen wie Leistungsfähigkeit bzw. Beschränktheit bestimmter Netze oder auf mathematische Beschreibungsmethoden eingegangen. Einführende Literatur zu diesen Fragestellungen stellen [4, 105, 167, 134, 90] dar.

Das gegenwärtige Interesse an künstlichen neuronalen Netzen liegt großenteils an ihrer Fähigkeit, natürliche Intelligenz nachzuahmen. Obwohl diese Methoden begrenzt sind und Mängel aufweisen, haben sie in manchen Anwendungen eindrucksvolle Leistungen erbracht. Gerade diese Mischung aus Erfolg und Mißerfolg weckt die verlockende Hoffnung, daß die Forschung letzten Endes zur Entwicklung von künstlichen Systemen führen wird, die in der Lage sind, einen großen Teil der Aufgaben zu verrichten, die jetzt noch menschliche Intelligenz erfordern. Dies erklärt auch das schnelle Wachstum der Anzahl der Forschungseinrichtungen auf diesem Gebiet. Diese Untersuchungen haben dazu geführt, daß neuronale Netze in einem breiten Spektrum von Anwendungen benutzt wurden. Diese beinhalten Musterklassifikation, Funktionsapproximation, Optimierung, Vorhersage, automatische Kontrolle und nicht zuletzt Robotik. Die Untersuchungen haben außerdem zur Entstehung einer großen Anzahl von Netzwerkparadigmen mit unterschiedlichen Namen geführt. Obwohl man, aufgrund dieser Vielfalt an Ansätzen, erwarten würde, daß das Gebiet höchst zerstückelt ist, und daß es lediglich aus einer Menge von unverwandten Methoden und Zielen besteht, ist dies nicht der Fall. Alle künstlichen neuronalen Netze erfüllen im Grunde die gleiche Funktion: sie bekommen eine Menge von Eingaben (Eingabevektor) und erzeugen eine entsprechende Menge von Ausgaben (Ausgabevektor). Alle Anwendungen von neuronalen Netzen sind ebenfalls Spezialfälle von solchen Vektorabbildungen. Wie in Abbildung 4.1 zu sehen ist, bekommen Vektorabbildungen eine Menge von Eingaben und erzeugen, gemäß einer in der Struktur codierten Abbildungsrelation, eine Menge von Ausgaben. Abbildung 4.2, zum Beispiel, zeigt ein System, das einen Eingabevektor mit drei Komponenten (Größe, Gewicht und Alter) auf einen Ausgabevektor mit zwei Komponenten (Lebenserwartung

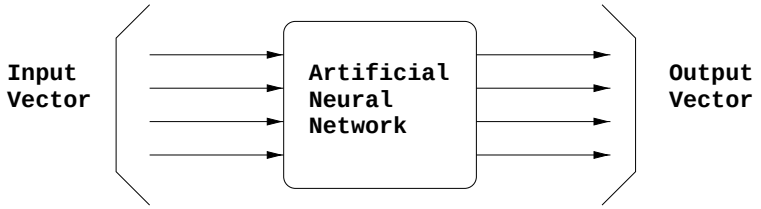


Abbildung 4.1: *Die allgemeine Betrachtung eines neuronalen Netzes als eine Vektorabbildung.*

und Versicherungsprämie) abbildet.

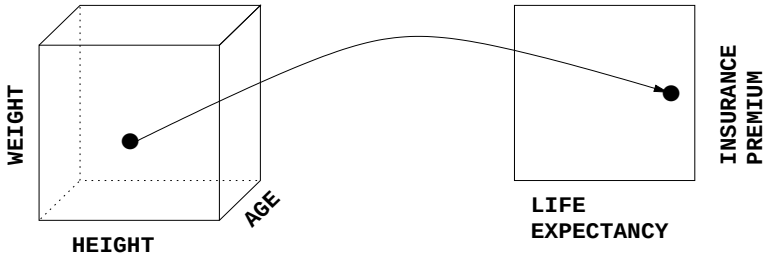


Abbildung 4.2: *Eine Vektorabbildung nimmt einen Vektor (oder Punkt) und konvertiert ihn in einen anderen Vektor (oder Punkt).*

Die wesentlichen Grundprinzipien eines künstlichen Neurons sind vom biologischen Vorbild abgeleitet. Jedes Verarbeitungselement erhält eine Menge von Eingaben und liefert dazu einen Ausgabewert. Dabei können die Eingaben entweder von anderen Neuronen oder von externen Eingabemedien stammen. Die Eingaben können über die Verbindungen verstärkt (exzitatorisch) oder abgeschwächt (inhibitorisch) werden. Die Berechnung der Ausgabe weicht bei den meisten künstlichen neuronalen Netzen stark von der natürlichen Vorgabe ab.

Ein sehr einfaches, schon 1943 vorgestelltes Neuronen-Modell ist das McCulloch-Pitts-Neuron [97] (siehe Abbildung 4.3). Hierbei werden die Eingänge x_i mit Gewichten ω_i multipliziert und aufsummiert. Das Neuron ist genau dann aktiv, wenn die so erhaltene Summe über einem vorher spezifizierten Schwellwert θ liegt. Das Ausgabesignal y eines Neurons ist daher gegeben durch:

$$y = \phi(\sum_i \omega_i x_i - \theta) \tag{4.6}$$

wobei gilt $\phi(x) = 1$ für $x \geq 0$ und $\phi(x) = 0$ für $x < 0$. y, ω_i und θ sind diskret ($y \in \{0,1\}, \omega_i \in \{-1,1\}, \theta \in \{0,1,2,3,\ldots\}$). McCulloch und Pitts demonstrierten, daß sich jede beliebige logische Funktion durch geeignete Kombination derartiger Elemente aufbauen läßt. Ihr Beweis beruht auf der Beobachtung, daß sich als Spezialfälle von Gleichung (4.6) insbesondere UND-Gatter und Inverter realisieren lassen, auf die dann jede andere logische Funktion zurückgeführt werden kann. Das Modell von McCulloch und Pitts ließ erstmals ahnen, wie Neuronen logische Operationen ausführen könnten. Ihre Idee vom Neuron als logischem Schwellwertelement war ein fundamentaler Beitrag und hat sich bis heute, wenn auch in oft abgewandelter Form, in vielen Modellen erhalten.

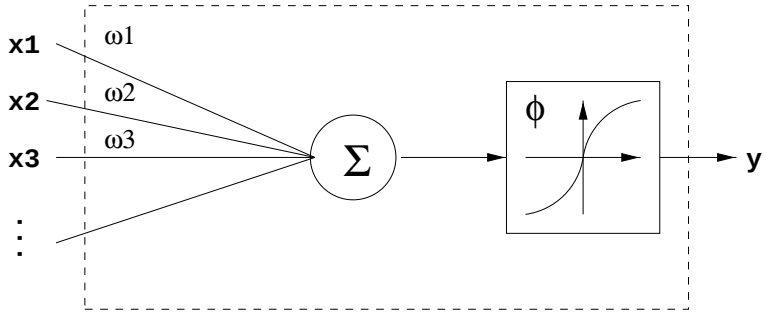


Abbildung 4.3: *McCulloch-Pitts-Neuron. x_i: Eingänge (Stimuli), ω_i: Gewichte, Σ: Summation, ϕ: Aktivierungsfunktion, y: Erregungszustand (Aktivierungswert).*

In neueren Publikationen sind die Gewichte und die Aktivierungswerte meistens reelle Zahlen. Für die Aktivierungsfunktionen werden oft differenzierbare Funktionen benutzt. Ein neuronales Netz besteht aus einer Anzahl von Neuronen, die mit Hilfe von gewichteten Verbindungen kommunizieren. Zusätzlich zu den Eigenschaften der Basiseinheit (Neuron), haben auch die Topologie und der Datenfluß einen großen Einfluß auf die Funktionalität eines neuronalen Netzes. Man unterscheidet dabei oft zwischen Feed-Forward-Netzen und Recurrent-Netzen. Bei Feed-Forward-Netzen ist der Datenfluß zwischen Eingabeneuronen und Ausgabeneuronen ausschließlich in Vorwärtsrichtung. Die Datenverarbeitung kann sich auf mehrere Units (Neuronen) erstrecken. Es sind aber keine Feedback-Verbindungen vorhanden, d.h., es existieren keine Verbindungen von Outputs von Units zu Inputs von Units, die in der gleichen oder vorherigen Schicht liegen. Klassische Beispiele für Feed-Forward-Netze sind das Perzeptron [104] und Adaline

(adaptive linear element) [173]. Recurrent-Netze enthalten Feedback-Verbindungen. Im Gegensatz zu Feed-Forward-Netzen, sind hier die dynamischen Eigenschaften des Netzes von großer Wichtigkeit. In einigen Fällen werden die Aktivierungswerte der Units einem Anpassungsprozeß (relaxation process) unterzogen bis das Netz einen stabilen Zustand erreicht, in dem die Aktivierungswerte sich nicht mehr ändern. In anderen Anwendungen, sind die Änderungen der Aktivierungswerte der Ausgabe-Units signifikant. Das dynamische Verhalten definiert in diesem Fall die Ausgabe des Netzes [120]. Recurrent-Netze wurden z.B. in den Arbeiten von Anderson [3] und Kohonen [81] untersucht. Beispiele für verschiedene Topologien stellen Multi-Layer-Netze (mehrschichtige Netze) und Hopfield-Netze dar. Bei Multi-Layer-Netzen sind, wie schon der Name sagt, mehrere Schichten von Neuronen miteinander verbunden, wobei es keine Verbindungen zu Neuronen in der gleichen Schicht gibt. Diese Netze werden zur Klassifikation eingesetzt und können zur Darstellung von stetigen, nichtlinearen Funktionen verwendet werden. Gerade diese Eigenschaft ist in der Robotik von großem Nutzen, denn sowohl die Kinematik und Dynamik als auch die Sensorverarbeitung können mit nichtlinearen Funktionen beschrieben werden. Hopfield-Netze [70] haben eine andere Netzstruktur. Sie werden als Assoziativspeicher und zum Lösen kombinatorischer Optimierungsaufgaben eingesetzt. Topologisch unterscheiden sie sich von Multi-Layer-Netzen dadurch, daß alle Neuronen miteinander verbunden sein können.

Künstliche neuronale Netze sind in der Lage aus der Erfahrung zu lernen. Diese Eigenschaft hat, vielleicht mehr als jede andere, das gegenwärtige Interesse an diesen Methoden hervorgebracht. Um die Netze an die gegebenen Aufgaben anzupassen, werden verschiedene Lernstrategien eingesetzt, die die Gewichte im Netz so verändern, daß das gewünschte Ein- und Ausgabeverhalten erzielt wird. Lernen basiert in den meisten Ansätzen auf der einheitlichen Anwendung einer Lernregel auf ein Netz von formalen Neuronen. Jedes Neuron j befindet sich in einem Zustand y_j, der durch folgende generische Gleichung beschrieben wird:

$$y_j(t+\delta t) \quad = \quad \phi[\sum_{i \in I_j} \omega_{ij}(t) y_i(t)]. \tag{4.7}$$

Dabei ist I_j die Menge von Inputs zu Neuron j (ein Input kann aus der Umgebung kommen oder der Output eines anderen Neurons sein), ω_{ij} das Gewicht der Verbindung zwischen Neuron i und Neuron j, t der Zeitpunkt, und ϕ nimmt meistens die Form einer

deterministischen oder einer stochastischen Schwellwertfunktion an.

Da ein Lernvorgang das Netzwerk in ganz bestimmter Weise adaptieren soll (bei einem assoziativen Netzwerk zum Beispiel dahingehend, daß gewünschte Assoziationen reproduziert werden), muß er in eine Richtung *gelenkt* werden. Mit anderen Worten, Lernen soll nicht völlig chaotisch und zufällig sein (wie es etwa rein stochastische Suchverfahren (die ,,random walk Methoden") sind), sondern soll sich in Richtung einer (vorher definierten) Performanzänderung bewegen. Beim Lernen unterscheidet man im allgemeinen zwei Formen von Leitung: (i) direkt durch einen *Lehrer* oder (ii) durch Festlegung, wie sich Aktivierung heranbilden kann. Im ersten Fall definiert sich die einzuschlagende Richtung anhand der Abweichung der momentanen Aktivierungen von den gewünschten. Da diese Abweichung für die Modellkomponente von außen definiert werden muß, wird diese Art von Lernen *überwachtes Lernen (supervised learning)* genannt. Der zweite Fall ist wesentlich, wenn die genaue Antwort des Modells nicht von vorneherein bekannt ist, wohl aber ihre prinzipielle Ausprägung. Da das Lernen in gewissem Sinne ,,sich selbst überlassen" (d.h. nicht direkt von außen beeinflußt) wird, wird dies als *unüberwachtes Lernen (unsupervised learning)* bezeichnet.

4.4.1 Überwachtes Lernen

Die Lernregeln, die bei dieser Lernart benutzt werden, bestehen aus dem Vergleich der ,,Antwort zu einem gegebenen Eingabemuster" mit der ,,gewünschten Antwort" und der anschließenden Änderung der Gewichte in Richtung des abnehmenden Fehlers. Zwei solche Regeln, nämlich, die LMS-Regel (Least Mean Square) und der Backpropagation-Algorithmus, wurden zur Steuerung von Robotern angewendet.

Die *LMS-Lernregel* [173] wird durch folgende Gleichung ausgedrückt:

$$w_{ij}(t+1) \quad = \quad w_{ij}(t) + c(y_j^*(t) - y_j(t))y_i(t) \tag{4.8}$$

wobei beide, die gewünschte Antwort $y_j^*(t)$ und die tatsächliche Antwort $y_j(t)$ von Neuron j, reelle Zahlen sind und ϕ in Gleichung (4.7) die Identitätsfunktion ist. Um ein unbeschränktes Wachsen der Gewichte zu vermeiden, werden verschiedene Normalisierungsprozeduren benutzt. Häufig werden Prozeduren verwendet, die auf der euklidischen Metrik basieren.

Die wiederholte Präsentation von Paaren (X_l, z_l) ($l = 1, \ldots, k$ und $X_1, \ldots, X_k$ linear unabhängig) zu einem zweischichtigen Netz, das mit dieser Regel (auch *Medaline* genannt) ausgestattet ist, führt zur Konvergenz der Gewichte Richtung einer Konfiguration bei der die Antwort auf ein Stimulus X_l die gewünschte reelle Zahl z_l ist. Diese Regel stellt somit eine iterative Prozedur zur Verfügung, mit der die Lösung eines linearen Gleichungssystems gefunden werden kann. Wenn die Muster $X_1, \ldots, X_k$ nicht linear unabhängig sind, kann die Regel leicht modifiziert werden (Parameter c in Gleichung (4.8) durch eine Variable ersetzen, die mit der Zeit Richtung 0 geht), so daß die Konvergenz der Gewichte den mittleren quadratischen Fehler zwischen den tatsächlichen Outputs und den gewünschten Outputs minimiert (deshalb auch der Name LMS: Least Mean Square). Mit dieser leichten Modifikation wird somit durch die LMS-Regel eine lineare Regression iterativ berechnet.

Eine Erweiterung dieser Regel zu dem Fall, in dem die gewünschte Antwort nur für eine Teilmenge der Neuronen (solche, deren Outputs den Output des Netzes bilden) spezifiziert ist, stellt der *Backpropagation-Algorithmus* dar [171, 137]. Wie schon der Name sagt, werden beim Backpropagation-Algorithmus Fehlersignale von den Ausgabeneuronen zurück zu den Eingabeneuronen und durch alle Zwischenschichten propagiert, so daß passende Korrekturen auf allen Verbindungsgewichten angewendet werden können:

$$w_{ij}(t+1) \quad = \quad w_{ij}(t) - c\frac{\partial E}{\partial w_{ij}}. \tag{4.9}$$

Dabei bezeichnet E den mittleren quadratischen Fehler zwischen den tatsächlichen und gewünschten Antworten zu allen Eingabemustern.

Bei genauer Betrachtung von Gleichung (4.9), stellt man fest, daß die LMS-Regel tatsächlich eine spezielle Instanz dieser generischen Regel ist. Da für die LMS-Regel die Funktion ϕ in Gleichung (4.7) die Identität ist, ist $\partial y_j / \partial w_{ij} = y_i$ und somit gilt:

$$\frac{\partial E}{\partial w_{ij}} \quad = \quad \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} \quad = \quad -(y_j^* - y_j)y_i. \tag{4.10}$$

Setzt man dieses Ergebnis in Gleichung (4.9) ein, so ergibt sich Gleichung (4.8).

Der Backpropagation-Algorithmus ist eine der am häufigsten verwendete Lernstrategien. Er kann als eine Gradientenabstiegsmethode betrachtet werden, die versucht, den Fehler E zu minimieren. Ein prinzipieller Nachteil dieser Lernstrategie ist das Problem *lokaler*

Minima. E ist in der Regel eine höchst komplexe Funktion sämtlicher Gewichte w_{ij} und kann daher zahlreiche lokale Minima besitzen. Je nach den gewählten Anfangswerten der w_{ij} führt der Gradientenabstieg stets in das nächstliegende Minimum, unabhängig davon, wie weit dies über dem absoluten Minimum liegt. Dadurch kann der Lernvorgang vorzeitig „steckenbleiben", ohne daß das Netz seine Aufgabe löst. Ob ein gutes Minimum erreicht wird, hängt dabei in meist unvorhersehbarer Weise von den Startwerten für die Gewichte und von der (in aller Regel unbekannten) Form des „Fehlergebirges" E ab. Eine weitere Schwierigkeit bieten Wertegebiete, für die die Höhe des Fehlergebirges nur geringfügig mit den w_{ij} variiert. Dort ist der Gradient sehr klein und damit auch die Adaptationsschritte. Trotz dieser Schwierigkeiten hat sich das Backpropagation-Verfahren gerade für Anwendungen aus der Robotik als sehr brauchbar erwiesen. Viele Arbeiten haben sich mit der Erweiterung und Beschleunigung dieses Algorithmus beschäftigt (siehe z.B. [36]).

4.4.2 Unüberwachters Lernen

Die Lernstrategien bei diesem Ansatz basieren meistens auf Variationen der klassischen *Hebbschen Hypothese* [66]. Danach ist die durch eine Verbindung (Synapse) bewirkte Verschaltung zwischen zwei Neuronen *plastisch* und ändert sich proportional zur korrelierten Aktivität vor und hinter der Synapse. Diese Hebbsche Hypothese hat in verschiedenen mathematischen Fassungen bis heute als Bestandteil vieler lernfähiger Netzwerkmodelle überlebt, obwohl ihre experimentelle Verifikation immer noch umstritten ist. Eine ihrer einfachsten mathematischen Fassungen in dem Kontext des generischen Neuronenmodells von Gleichung (4.7) ist gegeben durch die Lernregel:

$$w_{ij}(t+1) \;=\; w_{ij}(t) + cy_i(t)y_j(t). \tag{4.11}$$

Hierbei ist $c > 0$ ein Parameter, der die Größe eines einzelnen „Lernschritts" bemißt. Auch hier gelten die gleichen Normalisierungsbetrachtungen wie bei der LMS-Regel.

Die Hebbsche Lernregel wurde in Modellen des sogenannten *Competitive Learning* [138] integriert. Die Grundarchitektur dieses Verfahrens besteht aus mehreren Ebenen, die mit Feedforward-Verbindungen mit *positiven* Gewichten (exitatory) in eine Richtung

vernetzt sind. Die unterste Ebene ist die Eingabeschicht, an die Muster von außen angelegt werden können. Die Units (Neuronen) der anderen Schichten sind in Gruppen (Clusters) aufgeteilt, die lateral mit für die *Kompetition* notwendigen negativen Gewichten (inhibitory) verbunden sind. Das Netzwerk wird mit Eingabemustern angeregt und dann im Rahmen der Architektur und der vorgegebenen Lernschritte sich selbst überlassen. Muster, die so in einem der Clusters entstehen, können danach gegebenenfalls interpretiert werden, oder für die Eingabe an andere Modelle verwendet werden. Dadurch wird das Modell von einer externen Steuerung befreit, und stattdessen rein von der Architektur und den Eingabemustern (der Umgebung) geleitet.

Die Interpretation dieses selbstorganisierenden Prozesses ist im allgemeinen der einer *Kategorisierung*. Eine Unit in einem Cluster steht dabei für eine Kategorie von Mustern, die auf ein Element der Kategorie mit starker Aktivierung reagiert. Welche Kategorie es ist, kann natürlich a-priori nicht gesagt werden, da ja keine externe Kontrolle über die Auswahl der Units gegeben ist. Es kann nur gesagt werden, daß sich die Units in einem Cluster, sofern sich die Muster im Trainingsset in Gruppen (Kategorien) aufteilen lassen, mit hoher Wahrscheinlichkeit auf jeweils eine Kategorie spezialisieren (*sparse pattern learning theorem*, siehe z.B. [55]). Eine sichere Aussage kann nicht gegeben werden, da dies von den zufälligen Anfangsverteilungen der Gewichte abhängt. Deshalb werden in einer Schicht meist mehrere Cluster zur Verfügung gestellt. Dadurch erhält das Modell zusätzlich die Möglichkeit, Kategorien entlang verschiedener Dimensionen zu ,,entdecken", da Muster sich oft nach verschiedenen Kriterien gruppieren lassen. Diese Kriterien sind die Ähnlichkeiten unter den Mustern, das Maß dafür ist die Überlappung von aktiven Units.

Die Funktionsweise ist wie folgt: Wird ein Eingabemuster angelegt, so breiten sich zunächst die Aktivierungen aus und die Output-Werte der Units aller höherliegenden Schichten werden bestimmt. Danach folgen zusätzliche Update-Zyklen in diesen Schichten. Da Units innerhalb eines Clusters negativ verbunden sind, setzt Kompetition in jedem Cluster und somit der *,,the rich get richer" Effekt* ein. Dies wird solange fortgesetzt, bis der Gewinner des Clusters praktisch Maximalaktivierung (1) und alle anderen Units Minimalaktivierung (0) erreicht haben. Zur Vereinfachung kann das Ergebnis allerdings vorweggenommen werden: Dazu wird in jedem Cluster der Gewinner ermittelt und sofort auf 1 gesetzt, während alle anderen Units eine Nullaktivierung erhalten. Dieser

Prozeß wird auch *,,winner take all"* genannt.

Ein dem Competitive Learning eng artverwandtes Modell sind die sogenannten *topological feature maps* (auch *self-organizing feature maps* genannt) [81]. Auch dieses Modell geht von einer Architektur aus, in der eine Eingabeschicht voll mit einer Schicht von kompetitiven Units verbunden ist. Ein wesentlicher Unterschied besteht darin, daß diese kompetitiven Units eine topologische Struktur besitzen, daß also für jede Unit auch *Nachbarn* definiert sind.

In der Schicht von Units findet wieder eine Kompetition, also die Auswahl eines Gewinners statt. Eine Unit k gewinnt falls sie folgende Bedingung erfüllt:

$$\sum_i w_{ik} y_i \quad \geq \quad \sum_i w_{ij} y_i, \forall j. \tag{4.12}$$

Die Lernregel des Competitive Learning muß insofern abgewandelt werden, als daß nicht der Gewinner alleine sondern auch seine Nachbarn - mit einem Abschwächungsfaktor - mitlernen:

$$w_{ij}(t+1) \quad = \quad w_{ij}(t) + ch_k(j)(y_i(t) - w_{ij}(t)), \tag{4.13}$$

wobei $h_k(.)$ eine vom Abstand zwischen j und k abhängige Funktion ist, die ihr Maximum bei $j = k$ hat und die für große Abstände gegen 0 strebt. Dies hat nun eine entscheidende Auswirkung: Sofern die Inputs gemäß ihren Ähnlichkeiten eine topologische Struktur in der gleichen Dimensionalität wie die Struktur der kompetitiven Ebene haben, so wird diese Topologie vom Netzwerk sozusagen ,,wiederentdeckt". Das heißt, sie spiegelt sich insofern wider, als daß benachbarte Units auf in der Topologie benachbarte Input-Muster ansprechen. Das Ergebnis wird daher *Karte (map)* genannt. Dabei ist es wesentlich, zwischen der Dimension der Input-Vektoren und der Dimension der Karte zu unterscheiden, die keineswegs gleich sein muß. Sehr oft wird ein hochdimensionaler Input-Raum auf eine ein- oder zweidimensionale Karte abgebildet, die demnach nur eine Teilstruktur des Input-Raums nachempfinden kann.

4.5 Reinforcement Lernen

Überwachtes Lernen stellt eine allgemeine Methode dar, mit der ein parametrisierter Abbildungsapproximator, wie zum Beispiel ein neuronales Netzwerk, trainiert werden kann.

Beim überwachten Lernen sind aber Muster von Input-Output-Paaren der Abbildung, die gelernt werden muß, erforderlich. Anders ausgedrückt, es wird eine Menge von Fragen mit den entsprechenden Antworten benötigt. Methoden des überwachten Lernens, wie zum Beispiel Backpropagation, basieren auf der Annahme, daß für die Ausgabeknoten des Systems Fehlersignale zur Verfügung stehen. Das System wird häufig mit einer festen Menge von Beispielen trainiert, die von vorneherein bekannt ist. Unglücklicherweise gibt es viele Situationen, in denen die richtigen Antworten unbekannt sind. Aus diesem Grund gibt es ein großes Interesse für das Gebiet des *Reinforcement Lernens* (RL). Beim RL wird dem System ein Ziel (goal) gegeben. Das System lernt durch *,,Versuch und Irrtum"* (trial and error), wie das Ziel erreicht werden soll. Methoden des Reinforcement Lernens sind besonders geeignet, wenn das System on-line lernen muß, oder wenn kein ,,Lehrer" vorhanden ist, der Fehlersignale oder richtige Antworten liefert.

Bei RL-Problemen ist das Feedback einfach ein Skalarwert, der eventuell nur nach zeitlicher Verzögerung (delayed reinforcement) zur Verfügung steht. Dieses Bewertungssignal (reinforcement signal) reflektiert den Erfolg oder Mißerfolg des Gesamtsystems, nachdem eine gewisse Sequenz von Aktionen ausgeführt wurde. Das Bewertungsignal kann somit keine ,,Belohnung" oder ,,Bestrafung" zu einer speziellen Aktion der Sequenz (das *temporal credit assignment* Problem) oder zu einem speziellen Knoten oder Systemelement (das *structural credit assignment* Problem) zuordnen.

RL versucht somit, eine kompliziertere Aufgabe als überwachtes Lernen zu lösen. Um ein Lernen ohne Lehrer möglich zu machen, überwacht ein Agent Änderungen im Bewertungssignal und entscheidet sich für die *beste* Aktion. Die beste Aktion ist diejenige, die die Summe der erwarteten Bewertungssignale maximiert.

RL wird häufig auch als *Lernen mit einem Kritiker* bezeichnet und nimmt damit in gewisser Hinsicht eine Mittelstellung zwischen überwachtem Lernen einerseits und unüberwachtem Lernen andererseits ein. Zu den bekanntesten RL-Methoden zählen AHC (Adaptive Heuristic Critic) [157] und Q-Learning [168]. Im folgenden werden zuerst Markoffsche Entscheidungsprozesse erläutert. Diese gelten als Basis für theoretische Betrachtungen im Zusammenhang mit RL-Problemen. Danach werden AHC und Q-Learning kurz beschrieben. Mehr Details über Reinforcement Lernen können in [13, 168, 158] gefunden werden.

4.5.1 Markoffsche Umgebungen und Entscheidungsprozesse

Im Rahmen von RL trifft ein Agent Entscheidungen auf der Basis des Zustandes seiner Umgebung. Mit Zustand ist jede Information gemeint, die für den Agenten zur Verfügung steht. Es ist klar, daß der Zustand die unmittelbaren sensorischen Daten umfaßt. Zustandsdarstellungen können aber wesentlich mehr enthalten. Sie können das Ergebnis einer intensiven Verarbeitung originaler sensorischer Daten sein. Sie können aber auch komplexe Strukturen sein, die mit der Zeit aus Sequenzen sensorischer Information gewonnen werden.

Wenn die Zustandsdarstellung die vorhergehenden sensorischen Daten kompakt zusammenfaßt und gleichzeitig dafür sorgt, daß alle relevanten Informationen erhalten bleiben, dann besitzt sie die *Markoff-Eigenschaft*. Um dies formal zu definieren betrachte man die Antwort einer allgemeinen Umgebung zum Zeitpunkt $t+1$ auf die Aktion, die zum Zeitpunkt t ausgeführt wurde. Im allgemeinsten Fall kann diese Antwort abhängig sein von allem was vorher geschehen ist. Die Dynamik der Umgebung kann in diesem Fall durch die Angabe der Wahrscheinlichkeiten:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, \ r_1, s_0, a_0\} \tag{4.14}$$

für alle s', r, und alle möglichen Werte der vergangenen Ereignisse: $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ definiert werden. Dabei bezeichnen s_t, a_t und r_t jeweils den Zustand der Umgebung, die Aktion und das Bewertungssignal (reward) zum Zeitpunkt t. Wenn die Zustandsdarstellung die *Markoff-Eigenschaft* besitzt, dann ist die Antwort der Umgebung zum Zeitpunkt $t+1$ nur vom Zustand und von der Aktion zum Zeitpunkt t abhängig. In diesem Fall kann die Dynamik der Umgebung durch die Angabe der Wahrscheinlichkeiten:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{4.15}$$

für alle s', r, s_t und a_t spezifiziert werden. Anders ausgedrückt, eine Zustandsdarstellung besitzt die Markoff-Eigenschaft genau dann, wenn (4.15) und (4.14) für alle s', r und Historien $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ gleich sind.

Wenn eine Umgebung die Markoff-Eigenschaft besitzt, dann ermöglicht ihre Einschritt-Dynamik (4.15) die Vorhersage des nächsten Zustandes und des nächsten Bewertungssignals auch wenn nur der momentane Zustand und die momentane Aktion bekannt

sind. Es folgt auch, daß die Markoff-Eigenschaft die bestmögliche Basis für die Auswahl von Aktionen darstellt. Das heißt, die beste Strategie für die Auswahl von Aktionen im Markoffschen Fall ist genau so gut wie die beste Strategie in dem Fall, in dem die vollständige Historie bekannt ist.

Da die Markoff-Eigenschaft selten genau erfüllt ist, wird in RL versucht, die Zustandsdarstellung so zu wählen, daß diese Eigenschaft mindestens näherungsweise gilt. Die Markoff-Eigenschaft ist in RL wichtig, weil angenommen wird, daß Entscheidungen und Werte nur Funktionen des momentanen Zustandes sind. Damit diese Entscheidungen und Werte effektiv und informativ sein können, muß die Zustandsdarstellung informativ sein. Ein RL-Task, bei dem die Markoff-Eigenschaft gilt, wird *Markoff-Entscheidungsprozeß (Markov decision process (MDP))* genannt. Ein endlicher MDP ist ein MDP mit endlicher Zustandsmenge und endlicher Aktionsmenge. Die aktuelle RL-Theorie beschränkt sich meistens auf endliche MDPs. Die Methoden und Ideen sind aber allgemeiner anwendbar.

Markoff-Entscheidungsprozesse werden allgemein verwendet, um dynamische Systeme zu modellieren. In einem solchen Modell interagiert ein *lernender Agent* mit einer *Umgebung*, die durch Diskretisierung des Zustandsraumes und des Zeitparameters charakterisiert ist. In jedem Zeitschritt nimmt der Agent den Zustand der Umgebung $s_t \in \mathcal{S}$ wahr und wählt auf dieser Basis eine Aktion $a_t \in \mathcal{A}$ aus. Als Antwort auf jede Aktion a_t, erzeugt die Umgebung einen Zeitschritt später einen numerischen Reward (Bewertungssignal) r_{t+1} und einen nächsten Zustand s_{t+1}. Wenn $\mathcal{S}$ und $\mathcal{A}$ endlich sind, dann wird die Dynamik der Umgebung durch die Angabe der Zustandsübergangswahrscheinlichkeiten:

$$\mathcal{P}^a_{ss'} \quad = \quad Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{4.16}$$

und der Erwartungswerte der Rewards:

$$\mathcal{R}^a_{ss'} \quad = \quad E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{4.17}$$

für alle $s, s' \in \mathcal{S}$ und $a \in \mathcal{A}$ modelliert. Die beiden Größen, $\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$, ermöglichen eine vollständige Spezifikation der wichtigsten Aspekte der Dynamik der Umgebung.

Die Aufgabe des Agenten besteht darin, eine *optimale Entscheidungsstrategie*, d.h. eine Abbildung, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, von Zuständen auf Wahrscheinlichkeiten für die Auswahl vorhandener Aktionen, zu bestimmen, die den Erwartungswert der diskontierten

(discounted) Summe der zukünftigen Rewards:

$$V^{\pi}(s) = E\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} |\ s_t = s, \pi\} \tag{4.18}$$

$$= E\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k\ r_{t+k+2} | s_t = s, \pi\}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}\ [\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')] \tag{4.19}$$

maximiert. Dabei ist $\pi(s,a)$ die Wahrscheinlichkeit mit der die Entscheidungsstrategie π die Aktion $a \in \mathcal{A}$ in Zustand s auswählt und $\gamma \in [0,1]$ ein sogenannter *Diskontierungsfaktor* (discount factor), der die Gewichtung der Rewards derart erlaubt, daß Rewards in der Zukunft geringer zur Summe beitragen als solche, die unmittelbar bevorstehen. Der Diskontierungsfaktor ist darüber hinaus für die Konvergenz der Summe in Gleichung (4.18) notwendig. Die größe $V^{\pi}(s)$ heißt *Wert* von Zustand s unter Entscheidungsstrategie π und V^{π} heißt *Wertefunktion* für die Strategie π. Eine *optimale Wertefunktion* liefert den Wert eines Zustandes unter einer optimalen Entscheidungsstrategie:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{4.20}$$

$$= \max_{a \in \mathcal{A}} E\{r_{t+1} + \gamma V^*(s_{t+1})\ | s_t = s, a_t = a\}$$

$$= \max_{a \in \mathcal{A}} \sum_{s'} \mathcal{P}^a_{ss'}\ [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{4.21}$$

Jede Entscheidungsstrategie, die den Wert jedes Zustandes maximiert (siehe Gleichung (4.20)) ist per Definition eine optimale Entscheidungsstrategie. Wenn V^* bekannt ist, dann kann eine optimale Entscheidungsstrategie leicht bestimmt werden. Man braucht nur in jedem Zustand s die Aktion a zu wählen, die die Summe auf der rechten Seite von Gleichung (4.21) maximiert.

Oft ist es ist günstiger Werte nicht mit Zuständen, sondern mit (Zustand,Aktion)-Paaren zu assoziieren. Der Wert für die Auswahl von Aktion a in Zustand s, $Q^{\pi}(s,a)$ (Q-wert), ist der Erwartungswert der diskontierten Summe der zukünftigen Rewards, wenn mit Aktion a in Zustand s angefangen wird und künftig Aktionen gemäß π gewählt werden:

$$Q^{\pi}(s,a) = E\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} |\ s_t = s, a_t = a, \pi\} \tag{4.22}$$

$$= \sum_{s'} \mathcal{P}^a_{ss'}\ [\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s', a') \; Q^\pi(s', a') \right] \tag{4.23}$$

Q^π heißt *Aktion-Wertefunktion* für Entscheidungsstrategie π. Die optimale Aktion-Wertefunktion ist gegeben durch:

$$\begin{aligned} Q^*(s, a) &= \max_\pi Q^\pi(s, a) && (4.24) \\ &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \; | s_t = s, a_t = a\} \\ &= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} \; Q^*(s', a') \right] && (4.25) \end{aligned}$$

Die Gleichungen (4.19), (4.21), (4.23) und (4.25) sind die bekannten *Bellman-Gleichungen* [15]. Sie drücken die Beziehung zwischen dem Wert eines Zustandes und den Werten seiner Nachfolgezustände aus. In RL sind die Größen $\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$, die den MDP definieren, im voraus nicht bekannt. Ein RL-Algorithmus muß eine optimale Entscheidungsstrategie durch Interaktion mit der Umgebung finden. Die meisten RL-Algorithmen für die Lösung von MDPs sind iterativ und erzeugen eine Sequenz von Schätzungen für die optimale (Aktion-)Wertefunktion oder für die die optimale Entscheidungsstrategie oder für beide. Dabei werden wiederholt alte Schätzungen mit den Ergebnissen eines neuen Versuchs kombiniert und neue Schätzungen erzeugt. AHC und Q-Learning sind Beispiele für solche RL-Algorithmen. Im folgenden werden diese kurz erläutert.

4.5.2 Adaptive Heuristic Critic (AHC)

AHC ist ein allgemeiner Algorithmus, der die Wahrscheinlichkeit von Aktionen, die zu einem hohen Reward führen, erhöht und die Wahrscheinlichkeit von Aktionen, die zu einem niedrigen Reward führen, reduziert. Der Agent wird bei diesem Algorithmus in zwei Module eingeteilt: Das erste ist ein Policy-Modul, das Empfehlungen für Aktionen aufgrund der aktuellen sensorischen Information (aktueller Zustand) generiert (lernt eine Schätzung für eine optimale Entscheidungsstrategie); das zweite ist ein Critic-Modul, das Vorhersagen für die Werte von Zuständen generiert (lernt eine Schätzung für die optimale Wertefunktion) (siehe Abbildung 4.4).

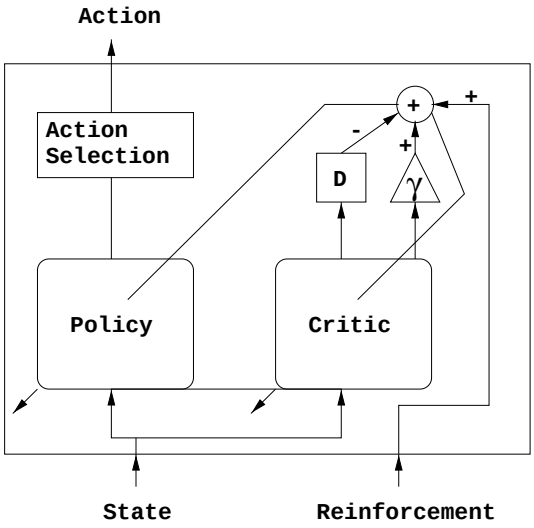


Abbildung 4.4: *AHC [157] hat zwei Komponenten: Das Policy-Modul generiert basierend auf den aktuellen Sensordaten Schätzungen für die auszuführenden Aktionen; das Critic-Modul generiert Vorhersagen für die Werte der Zustände. Das Policy-Modul erzeugt für jede Aktion einen Wert, der die Wahrscheinlichkeit mit der die Aktion ausgewählt wird, bestimmt (Eine Aktion wird dann gemäß diesen Wahrscheinlichkeiten durch einen Mechanismus zur Aktionsauswahl selektiert). Der aktuelle Reward (reinforcement) wird mit dem durch das Critic-Modul gelieferten und diskontierten (mit γ multiplizierten) Wert addiert. Das Ergebnis wird dann mit dem durch das Critic-Modul im letzten Zeitschritt (die Box „D" verzögert ihren Input um einen Zeitschritt) gelieferten Wert verglichen. Die Differenz wird dazu benutzt, das Critic-Modul anzupassen und das Policy-Modul zu modifizieren.*

AHC wird mit Hilfe einer sukzessiven Approximationsprozedur trainiert. In jedem Versuch wird die Entscheidungsstrategie angewendet (Aktionen werden gemäß dieser Entscheidungsstrategie gewählt) und für jeden besuchten Zustand die Critic-Modul-Vorhersage neuberechnet. Nach jeder Aktion wird die Critik-Modul-Schätzung für den Wert des vorigen Zustands modifiziert, um den aktuellen Reward und die Critik-Modul-Schätzung für den Wert des aktuellen Zustands zu reflektieren:

$$Critic(s_t) \leftarrow r(s_t, a_t) + \gamma Critic(s_{t+1}). \tag{4.26}$$

Dabei ist $Critic(s_t)$ die aktuelle Critik-Modul-Schätzung für den Wert von s_t und $r(s_t, a_t)$ der Reward für die Ausführung von Aktion a_t in Zustand s_t. „$A \leftarrow B$" bedeutet, daß A so modifiziert wird, daß der Absolutbetrag der Differenz zwischen B und A, $|B - A|$, reduziert wird (wie diese Anpassung genau stattfindet, ist vom Mechanismus für die Repräsentation von A abhängig: Tabelle, neuronales Netz, usw.). Diese Anpassungsprozedur stellt eine Form des *temporal difference learning* [156] dar. Nach einer gewissen Anzahl von Versuchen und unter bestimmten Bedingungen [35], konvergieren die Schätzungen zu den korrekten Werten, d.h., die linke Seite von Gleichung (4.26) konvergiert zum Erwartungswert der rechten Seite.

Wenn das Critic-Modul trainiert wurde, dann kann das Policy-Modul verbessert werden. Dies geschieht durch Auswahl und Ausführung von zufälligen Aktionen, die der alten Entscheidungsstrategie widersprechen, und Vergleich der Werte der besuchten Zustände. Wenn der Agent eine Aktion a ausführt, die der alten Entscheidungsstrategie widerspricht und von Zustand j nach Zustand i führt, und der erhaltene Reward plus dem Wert von Zustand i besser als erwartet ist (d.h, $r(j, a) + \gamma V(i) > V(j)$), dann ist die Aktion a besser als die der alten Entscheidungsstrategie. In diesem Fall kann eine neue Entscheidungsstrategie erzeugt werden, die diese verbesserte Aktionsauswahl berücksichtigt. Wenn die Umgebung stochastisch (nichtdeterministisch) ist, dann muß die neue Aktion mehrmals ausprobiert werden (theoretisch unendlich lang, in der Praxis reichen meistens wenige Tests), damit sichergestellt wird, daß sie die Entscheidungsstrategie im Mittel verbessert. Ein neues Critic-Modul kann jetzt trainiert werden, um die korrekten Zustandswerte für diese neue Entscheidungsstrategie vorherzusagen. Es ist natürlich klar, daß nur Zustände, die nach Zustand j führen, betroffen sind. Wenn das neue Critic-Modul bereitsteht, dann können weitere Verbesserungen an der Entscheidungsstrategie gelernt werden. Wiederholt man diesen Prozeß bis keine Änderungen mehr möglich sind, dann erhält man am Ende eine optimale Entscheidungsstrategie.

Anstatt das Policy-Modul und das Critic-Modul iterativ anzupassen, können diese beiden Prozesse simultan erfolgen, indem zufällige Aktionen, die der alten Entscheidungsstrategie widersprechen, ausgeführt werden. In diesem Fall ist die Entscheidungsstrategie stochastisch, und es wird mit jeder Aktion in jedem Zustand ein *Gewicht* assoziiert, das die Wahrscheinlichkeit mit der die Aktion ausgewählt wird bestimmt. Das Gewicht wächst oder schrumpft abhängig davon, ob die Aktion einen höheren oder einen niedri-

geren Zustandswert als andere Aktionen generiert. Aus diesem Grund wird es für eine Aktion wahrscheinlicher, daß sie in einem Zustand in der Zukunft ausgewählt wird, wenn ihre Ausführung zu einem Wert für den Zustand führt, der besser als erwartet ist. Wenn, im Gegensatz, ihre Ausführung zu einem Wert führt, der schlechter als erwartet ist, dann wird es für sie weniger wahrscheinlich, daß sie in der Zukunft ausgewählt wird. Die nächste Aktion wird auf der Basis der Gewichte der Aktionen, die im aktuellen Zustand verfügbar sind, durch den Aktionsselektor (siehe Abbildung 4.4) stochastisch ausgewählt. Im Gegensatz zum obigen iterativen Ansatz, wurde beim simultanen Fall die Konvergenz zur optimalen Entscheidungsstrategie nicht bewiesen.

4.5.3 Q-Learning

Der AHC-Algorithmus ist in der Lage, viele schwierige RL-Aufgaben zu Lernen. In seiner simultanen Standardform wurde seine Konvergenz zu einer optimalen Entscheidungsstrategie allerdings nicht bewiesen. *Q-Learning* [168] ist ein RL-Algorithmus, für den die Konvergenz unter bestimmten Bedingungen bewiesen werden konnte [168, 169].

Q-Learning ist dem AHC-Algorithmus sehr ähnlich. In gewisser Hinsicht kombiniert dieses Verfahren die Funktionen der beiden AHC-Module in einem einzigen Modul. Anstatt Werte für Zustände vorherzusagen, generiert das neue Modul Vorhersagen für *Q-Werte* (Werte für (Aktion,Zustand)-Paare). In jedem Zeitschritt werden die (Aktion,Zustand)-Paare für den aktuellen Zustand untersucht. Die Aktion aus dem Paar mit dem höchsten geschätzten Q-Wert ist diejenige, die am wahrscheinlichsten gewählt wird. Abhängig von dieser Aktion und vom Reward wird der Q-Wert des entsprechenden (Aktion,Zustand)-Paares nach folgender Update-Regel angepaßt:

$$Q(s_t, a_t) \quad \leftarrow \quad r(s_t, a_t) + \gamma(\max_a Q(s_{t+1}, a)). \tag{4.27}$$

Dabei ist $Q(s_t, a_t)$ der geschätzte Q-Wert für die Ausführung von Aktion a_t in Zustand s_t (Zustand des Agenten zum Zeitpunkt t). Aktionen können zum Beispiel gemäß folgender Wahrscheinlichkeiten gewählt werden:

$$Pr(a|s_t) \quad = \quad \frac{e^{\frac{Q(s_t,a)}{T}}}{\sum_b e^{\frac{Q(s_t,b)}{T}}}. \tag{4.28}$$

Dabei ist T ein positiver Parameter (*temperature* genannt). Höhere Werte für T führen dazu, daß alle Aktionen gleichwahrscheinlich werden. In diesem Fall ist die Aktionsauswahl völlig zufällig. Niedrige Werte für T führen zu großen Unterschieden zwischen den Wahrscheinlichkeiten für Aktionen deren Q-Werte verschieden sind. Zufall spielt in diesem Fall keine Rolle. Die Aktionsauswahl ist deterministisch. Wenn die Q-Werte in einer Tabelle gespeichert werden und T asymptotisch abnimmt (um sicherzustellen, daß es möglich ist, jede Aktion in jedem Zustand zu testen), dann ist es garantiert (zumindest theoretisch), daß die Q-Werte zu den korrekten Werten konvergieren.

Q-Werte können in Tabellenform oder mit Hilfe von neuronalen Netzwerken gespeichert werden. In [86], zum Beispiel, wurde für jede Aktion ein separates Netzwerk benutzt. Die Netzwerke wurden mit folgendem Fehlersignal (error) trainiert:

$$\Delta Q_{a_t} \quad = \quad [r(s_t, a_t) + \gamma(\max_a Q(s_{t+1}, a))] \; - Q(s_t, a_t). \tag{4.29}$$

ΔQ_{a_t} ist dabei der Fehler, der durch das Netzwerk, das zur zuletzt ausgeführten Aktion gehört, ,,rückwärts" propagiert wird. In dieser Studie wurde festgestellt, daß, wenn AHC und Q-Learning mit Hilfe von neuronalen Netzwerken implementiert werden, ihre Leistungen vergleichbar sind. Die Konvergenz von Q-Learning ist allerdings nur bei genauer Speicherung der Q-Werte, z.B. in einer Tabelle, gesichert. Eine solche Garantie ist nicht gegeben, wenn die Werte nur approximiert werden, wie es bei neuronalen Netzwerken der Fall ist.

4.6 Anwendung auf dem Gebiet der Robotik

Die Robotik stellt ein wichtiges Anwendungsgebiet für neuronale Netze dar. Neuronale Netze können, aufgrund ihrer Adaptierungs- und Lerneigenschaften, für die Steuerung von Systemen mit unbekanntem und komplexem Verhalten benutzt werden. Neben Anwendungen zur Lösung des inversen kinematischen Problems [126] und des inversen dynamischen Problems [30] werden neuronale Netze zum Auswerten und Interpretieren von Sensorwerten und zur sensorgesteuerten kollisionsfreien Navigation bei mobilen Robotern eingesetzt.

Ein Beispiel für die Anwendung neuronaler Netze zur Klassifikation von Sensordaten stellt die Simulation eines Folgealgorithmus mit Hilfe eines Backpropagation-Netzes dar

[68]. Die Aufgabenstellung hierbei war, mittels 3 an der Frontseite eines mobilen Roboters angebrachten Ultraschallsensoren einem zweiten Roboter zu folgen. Interessanterweise brauchte das Netz zum Erlernen der Zuordnung zwischen Sensormessungen und Steuerinformation an die Räder nur eine sehr geringe Anzahl von repräsentativen Mustern (2000 von 360 000 möglichen Mustern) und relativ wenige Wiederholungen.

Ein ähnliches Anwendungsbeispiel stellten Sekiguchi, Nagata und Asakawa in [145] vor. Sie entwickelten einen nur etwa 31 cm großen und im Durchmesser 26 cm breiten Roboter, dessen Verhalten antrainiert wurde. Beispielsweise hat er die Aufgabe, einen anderen Roboter, der sich zufällig durch den Raum bewegt, zu fangen oder vor einem ihm folgenden Roboter zu flüchten. Zum Erfassen der Umgebung, Lokalisieren eines zweiten Roboters und zur Bestimmung interner Zustände stehen 12 Sensoren zur Verfügung (4 Infrarotsensoren, 3 Ultraschallsensoren, 2 taktile Sensoren und 3 Näherungsschalter). Zusätzlich ist der mobile Roboter mit einem Buzzer und einer Lampe ausgestattet, die beispielsweise einem verfolgenden Roboter zur Ortung dienen können. Die Verhaltenskontrolle wird mittels neuronaler Netze durchgeführt. Hierzu werden zwei Multi-Layer-Netze verwendet, die den Sensoreingängen Motorbefehle zuordnen. Die Ausgabe des Gesamtnetzes klassifiziert die nächste Aktion des Roboters. Hierzu wurden 5 Aktionsklassen definiert (Vorwärts, Rückwärts, Rechts, Links und Buzzer). Zum Einlernen des Verhaltens also der Zuordnung der verschiedenen Eingaben zu den 5 Ausgabeklassen genügten für das erste Netz 62 und für das zweite Netz 10 Muster. Als Lernstrategie wurde eine Modifikation des Backpropagation-Algorithmus angewandt, die mit ,,pseudo impedance control" bezeichnet wird. Die Konvergenz des Lernalgorithmus stellte sich nach etwa 500 Wiederholungen ein.

Eine weiteres Beispiel einer sensomotorischen Abbildung stellt das ALVINN-System (Autonomous Land Vehicle In a Neural Net) dar [123]. Bei dieser Anwendung wird das CMU-Navlab-System durch ein neuronales Netz gesteuert. Die Aufgabe besteht darin, einer Fahrbahn zu folgen. Im Gegensatz zu anderen Algorithmen, die für dieses CMU-Fahrzeug entwickelt wurden [79] und im Gegensatz z.B. zur Arbeit von Dickmanns [37], besitzt ALVINN kein Modell der Fahrbahn. ALVINN lernt wie Assoziationen zwischen visuellen Mustern und Steuerkommandos hergestellt werden können. Es wird ein neuronales Feed-Forward-Netz benutzt. Die Eingabe zum Netz besteht aus einem 30x32-Pixel-Bild, das von einer Kamera, die auf dem Dach des Fahrzeuges montiert ist,

erzeugt wird. Jedes Pixel entspricht einer Eingabe-Unit des Netzes. Die Eingabeschicht ist mit einer Hidden-Schicht, die aus 5 Units besteht, vollständig verbunden. Die Ausgabeschicht besteht aus 30 Units und stellt eine lineare Repräsentation der Richtung dar, der das Fahrzeug folgen soll, damit es auf der Fahrbahn bleibt. Die Ausgabe-Unit mit der höchsten Aktivierung bestimmt die Steuerrichtung während des Operierens. Ursprünglich wurde das Netz durch Präsentation von 1200 Fahrbahnbildern mit den entsprechenden Steueraktionen trainiert. Als Lernalgorithmus wurde Backpropagation benutzt. Das Trainingset wurde dem Netz 30 - 40 mal präsentiert. Bei einer zweiten Serie von Experimenten wurden während der Lernphase reale Situationen benutzt. Während ein Fahrer (Mensch) das Navlab-Fahrzeug fährt, wird Backpropagation, mit dem aktuellen Kamerabild als Eingabe und der Richtung in die der Fahrer das Fahrzeug im Moment lenkt als gewünschte Ausgabe, benutzt. Nach einer Lernfahrt von etwa 5 Minuten, konnte das trainierte Netz das Fahrzeug mit einer Geschwindigkeit von 32 km/h fahren. Diese Geschwindigkeit war doppelt so hoch wie die Geschwindigkeit, die mit anderen nicht-neuronalen Algorithmen erreicht werden konnte.

Um kollisionsfreie Navigation zu ermöglichen, werden neben Bildern auch Ultraschallsensoren zur Entfernungsmessung benutzt. In [118] zum Beispiel, wird ein simuliertes Fahrzeug beschrieben, das mit Sensoren ausgestattet ist, die den Abstand zum nächsten Objekt im jeweiligen Sichtfeld zurückliefern. Es gibt 16 verschiedene Abstandswerte, die die Sensitivität zu näheren Hindernissen widerspiegeln. Diese werden in 4 Bits codiert. Es werden 9 Sensoren und somit ein neuronales Netz mit 36 binären Eingabe-Units benutzt. Das Netz besitzt 8 Hidden-Units und eine Ausgabeschicht aus 3 Units, die eine codierte Repräsentation der 8 verschiedenen Aktionen erlauben. Das Netz wird trainiert indem das Fahrzeug um 10 verschiedene Hindernisse herum bewegt wird. Nach dem Training ist das Fahrzeug in der Lage, ohne Kollisionen zu navigieren.

Auch Reinforcement Lernen wurde in den letzten Jahren in vielen Robotik-Anwendungen benutzt. Kröse und Dam [83] beschreiben einen Kontroller, der in der Lage ist zu lernen, einen mobilen Roboter in einer komplexen Umgebung ohne Kollisionen zu führen. Der Roboter besitzt 8 Entfernungssensoren, die auf einem halben Kreis auf der Vorderseite des Fahrzeuges verteilt sind, sowie einen Kollisionssensor. Es wird mit einer konstanten Geschwindigkeit gefahren, und der Roboter braucht keine anderen Aufgaben zu erledigen. Das einzige Feedback ist ein negatives Bewertungssi-

gnal, das nach einer Kollision generiert wird. Die Eingabe für den Kontroller ist ein 8-dimensionaler Entfernungsvektor, und die Ausgabe ist ein binäres Steuerungssignal. Es wurde eine diskrete Repräsentation des Eingaberaumes benutzt. Für die Diskretisierung wurden neuronale Netze benutzt.

Ein ähnlicher Ansatz zum Lernen von kollisionsfreiem Fahren wurde von Prescott und Mayhew [125] benutzt. Sie beschreiben auch eine Technik des Reinforcement Lernens für einen mobilen Roboter, der auf der Basis von 3 Entfernungssensoren navigieren soll. Der Unterschied zum obigen Ansatz ist, daß die Ausgabe keine binäre Einheit ist, sondern in der Lage ist, die Geschwindigkeit und die Fahrtrichtung des simulierten Roboters kontinuierlich zu variieren. Da auch ein völlig stationärer Roboter mit Hindernissen nicht kollidieren wird und somit eine Lösung des Problems darstellt, mußte ein auf der momentanen Geschwindigkeit basierendes Bewertungssignal hinzugefügt werden, um lokale Minima, mit einer Geschwindigkeit von 0 als Lösung, zu vermeiden.

Mahadevan und Connell diskutieren in [89] eine Aufgabe, in der ein mobiler Roboter große Kisten über längere Zeitabschnitte schieben soll. Kisten-Schieben ist ein bekanntes, schwieriges Robotik-Problem, das durch immense Unsicherheiten in den Aktionsresultaten gekennzeichnet ist. Q-Learning wurde in Verbindung mit einigen neuen Clusterungstechniken benutzt, die einen höherdimensionierten Input als Tabellen erlauben. Der lernende Roboter konnte eine Performanz erreichen, die mit der einer festprogrammierten Lösung vergleichbar war. Ein anderer Aspekt dieser Arbeit war die Zerlegung einer schwierigen Aufgabenbeschreibung in eine Menge von einfacheren Tasks, die gelernt werden sollen.

Mataric beschreibt in [94] ein Robotik-Experiment mit einem, aus der Sicht des theoretischen Reinforcement Lernens, hochdimensionalen Zustandsraum, der mehrere Freiheitsgrade enthält. Vier mobile Roboter bewegten sich innerhalb eines Zaunes, um kleine Scheiben zu sammeln und sie zu einer Zielregion zu transportieren. An dem ursprünglichen Q-Learning-Algorithmus wurden drei Verbesserungen vorgenommen. Erstens, vorprogrammierte Signale, *progress estimators* genannt, wurden dazu benutzt, den schwierigen Task in einfachere Subtasks zu zerlegen. Dies wurde auf eine robuste Art und Weise gemacht, bei der die Roboter nicht gezwungen waren, diese Signale zu benutzen, sondern die Freiheit hatten, von den Empfehlungen (advice), die diese Signale zur

Verfügung stellten, zu profitieren. Zweitens, die Kontrolle wurde dezentralisiert. Jeder Roboter lernte seine eigene Entscheidungsstrategie unabhängig und ohne explizite Kommunikation mit den anderen. Drittens, der Zustandsraum wurde, gemäß den Werten von vorprogrammierten booleschen Features der zugrundeliegenden Sensoren, in eine kleine Anzahl von diskreten Zuständen quantisiert. Die Performanz der durch Q-Learning gelernten Entscheidungsstrategien war mit der Performanz von für diese Aufgabe handcodierten Kontrollern vergleichbar.

Kapitel 5

EINE SICH SELBST VERBESSERNDE STEUERUNGSARCHITEKTUR

Nachdem in den vorhergehenden Kapiteln ein Überblick über bisherige Ansätze zur Realisierung von Intelligenz und Autonomie gegeben wurde, wollen wir in diesem Kapitel einige wichtige Anforderungen an mobile Roboter, die in der realen Welt agieren, stellen und eine Architektur, die diese Anforderungen erfüllen soll, vorschlagen.

5.1 Anforderungen an mobile Roboter

Eine erste wichtige Anforderung an Agenten (Roboter), die in der realen Welt agieren, ist die *Reaktivität* (reactivity). Damit der Agent in einer dynamischen Umgebung überleben kann, sind rechtzeitige Antworten und Aktionen notwendig. Der Begriff der ,,Rechtzeitigkeit" von Antworten und Aktionen ist allerdings verschwommen und läßt sich im allgemeinen nicht exakt definieren. Im Einzelfall aber, kann ein Beobachter entscheiden, ob ein Roboter diese Bedingung erfüllt oder nicht. Die Verwendung von schneller Hardware kann sicherlich dieses Problem weniger kritisch machen. Die Verbesserung wird aber bestimmt nicht so hoch ausfallen, wie sich viele Forscher wünschen. Das tatsächliche Problem liegt oft , wie von Niklaus Wirth [182] aufgeführt, in dem Mangel an simplerem Design und an klaren Konzepten. Man sollte sich mit der existierenden Rechenleistung zufrieden geben und versuchen, geschicktere und bessere Software zu entwickeln.

Zielorientierung (goal-orientedness) ist ein weiterer Aspekt der für die Autonomie von Agenten von großer Bedeutung ist. Die Aktionen des Agenten sollen im Kontext der Erfüllung eines Ziels (goal) erfolgen. Ein Benutzer zum Beispiel, stellt Anfragen auf hohem Niveau. Der Agent entscheidet, wie er die komplexe Anfrage aufteilt und durch Lösen von Teilaufgaben das Ziel (die Benutzeranfrage) erreicht. Eine vernünftige Weise, zielorientiertes Verhalten zu implementieren, besteht darin, eine Art von Absicht oder Zweck von Anfang an in die Steuerungsarchitektur des Roboters einzubetten: Nicht den Lösungsweg beschreiben, sondern den Agenten so stimulieren, daß das Ziel (die Lösung einer Aufgabe) zu seinem Wunsch oder Verlangen wird.

Reaktivität und Zielorientierung sollen miteinander passend kombiniert werden. Reaktivität soll dem System erlauben, sich schnell an veränderliche Situationen anzupassen. Zielorientierung soll die Bearbeitung von komplexen und variablen Tasks ermöglichen. Reaktivität soll an den auszuführenden Task angepaßt werden. Das Kontrollsystem soll eine kontrollierte Reaktivität erlauben, so daß die Reaktionen des Roboters mit dem Ziel (die zu lösende Aufgabe) konsistent (das Verhalten ist ziel- und ereignisgesteuert) bleiben.

5.2 Kombination von Reaktivität und Zielorientierung

Wie wir in Kapitel 3 gesehen haben, sind deliberative Architekturen durch das Vorhandensein eines explizit repräsentierten, symbolischen Modells der Welt gekennzeichnet. Mit Hilfe dieses Weltmodells werden Entscheidungen gefällt, z.B. über auszuführende Aktionen. Reaktive Architekturen enthalten keine Art von zentralem symbolischem Weltmodell und keine komplexen symbolischen Schlußfolgerungen. Stattdessen wird von einer engen Kopplung von Sensoren und Aktoren in kleinen aufgabenorientierten Einheiten ausgegangen. Beide Architekturformen haben Vor- und Nachteile. Der Hauptvorteil bei deliberativen Architekturen ist die globale Sicht, die für den Agenten zur Verfügung steht. Aufgrund der verfügbaren globalen Information (Weltmodell) kann die Planungskomponente entweder eine Sequenz von Aktionen berechnen, die garantiert zum Ziel führen oder zeigen, daß die Aufgabe unlösbar ist. Ein weiterer Vorteil ist die Möglichkeit von Optimierungen. Erfolgreiche Pläne können vor ihrer Anwendung optimiert werden. Der Hauptnachteil bei deliberativen Architekturen ist die mangelnde

Reaktivität. Der deliberative Ansatz arbeitet nach dem Schema: *sense-model-plan-act*. Wegen der hohen Komplexität (viele Details, intensive Berechnungen) dauert dieser Zyklus lange. Wenn die Umwelt sich zwischen Momentaufnahme und Aktion ändert (dynamische Umgebungen), dann wird der Plan ungültig, und der Agent kann nicht rechtzeitig reagieren. Ein weiterer Nachteil ist die mangelnde Robustheit. Wegen der engen gegenseitigen Abhängigkeit der Module (Komponenten), führt der Ausfall einer Komponente (z.B. wenn der Planer keinen Plan konstruieren kann) zum Ausfall des ganzen Systems. Der Hauptvorteil bei reaktiven Systemen ist die Reaktivität. Da es keine Weltmodelle und keine Sensorfusion gibt, existieren kaum Engpässe bei den Berechnungen. Die Verzögerung zwischen Sensoren und Aktoren ist minimal und der Roboter kann schnell reagieren. Ein weiterer Vorteil ist die Robustheit des resultierenden Systems. Da alle Verhaltensmuster die Glieder des Roboters direkt steuern können, bewirkt der Ausfall eines Verhaltensmusters nur eine Leistungsminderung. Das System kann mit den restlichen Verhaltensmustern weiterarbeiten. Der Hauptnachteil bei reaktiven Systemen ist der Mangel an Zielorientierung (Planungsfähigkeit). Globale Information und Planer fehlen. Pure Reaktionen garantieren nicht, daß das Ziel (Lösung der gestellten Aufgabe) erreicht wird. Ein weiterer Nachteil ist die mangelnde Flexibilität. Der Designer wählt eine Menge von Verhaltensmuster, von denen er denkt, daß sie zum gewünschten Ziel führen könnten, und codiert somit implizit die Designstrategie in die Architektur des Agenten. Das resultierende Steuerungssystem ist festverdrahtet (pre-wired) und der Roboter ist nicht programmierbar (für verschiedene Aufgabenstellungen). Seine Fähigkeiten beschränken sich auf die Verhaltensmuster, die der Designer implementiert hat.

Wie wir sehen, sind die Vorteile des einen Ansatzes die Nachteile des anderen. Reaktivität ist charakteristisch für reaktive verhaltensbasierte Systeme. Zielorientierung ist charakteristisch für klassische deliberative Systeme. Systeme, die sich in der realen Welt gut zurechtfinden können, liegen irgendwo zwischen diesen beiden Extrema. Die Frage ist, wie man sich einer Lösung nähern soll. Soll man von der reaktiven oder von der deliberativen Seite des Spektrums anfangen.

Bisherige Lösungsansätze (siehe Abschnitt 3.4) fangen meistens von der deliberativen Seite an. Es wird versucht, das klassische Planen so zu erweitern, daß es mit der Dynamik und Unsicherheit der realen Welt fertig werden kann. Diese Systeme sind meistens so aufgebaut, daß beim Auftreten eines Fehlers, d.h. der Plan ist ungültig geworden,

ein neuer Planungsvorgang gestartet wird (replaning) oder es wird versucht, alle potentiellen Fehler vorherzusagen, und sie vom Anfang an bei der Erstellung des Plans zu berücksichtigen. Mitte der achtziger Jahre hat Chapman [27] jedoch gezeigt, daß die Unterschiede zwischen den verschiedenen Planern nicht so wichtig sind. Er zeigte auch, daß das Planungsproblem für einen korrekten und vollständigen Planer unentscheidbar ist, und daß es somit keine obere Schranke für die Zeit, die gebraucht wird, gibt. Obwohl seit der Veröffentlichung dieser Untersuchungsergebnisse einige Erkenntnisse für die bessere Unterscheidung zwischen den Komplexitäten der Planer (abhängig von der Anzahl der Vor- und Nachbedingungen) gewonnen wurden, ist ein Durchbruch noch nicht gelungen. Die Lockerung der Anforderung der Korrektheit und Vollständigkeit führt oft nur zu heuristischen Planern, die nicht immer in der Lage sind, einen Plan zur Lösung der gestellten Aufgabe erzeugen zu können. Ein anderes Problem bei Planern, für das noch keine theoretische Lösung existiert, ist das Frame-Problem [96, 127]. Es besteht darin, zu beschreiben, welche Eigenschaften sich ändern, und welche unverändert bleiben, wenn Aktionen ausgeführt werden. Für einen Agenten, der in einer dynamischen Umgebung agiert, ist dies von großer Wichtigkeit, weil er dadurch sein Weltmodell und sein Wissen im allgemeinen mit der realen Welt synchron halten kann. Diese theoretischen Ergebnisse geben zu erkennen, daß in jedem System, in dem Zeiteinschränkungen existieren, selbst sehr verfeinerte Techniken sich letztendlich als unbrauchbar zeigen werden. Diese Vorgehensweise scheint also die Aufgabe nur komplexer zu machen und löst die grundsätzlichen Probleme nicht.

Eine andere Möglichkeit für die Kombination von Reaktivität und Zielorientierung besteht darin, ein deliberatives und ein reaktives System gleichzeitig laufen zu lassen. Wenn der Planer keinen Plan konstruieren kann, soll das reaktive System die Kontrolle übernehmen. Diese Ansätze leiden aber an dem sogenannten ,,Horizonteffekt": Sie lieferten mit der reaktiven Komponente eine etwas bessere Gesamtleistung, haben aber einfach die Einschränkungen des Reasoning-Systems nur etwas weiter in die Zukunft verschoben. Wenn die Umwelt dynamisch genug ist, ist es fraglich, ob der Planer überhaupt zum Einsatz kommt.

Eine vielversprechendere Lösung besteht in der Erweiterung eines reaktiven Basissystems um Zielorientierung. Weil ein einzelnes Verhaltensmuster, das Sensordaten direkt in passende Aktionen umsetzt, nur ein Teilproblem und nicht die ganze Aufgabe lösen

kann, ist es nötig, so viele Algorithmen (Verhaltensmuster) wie es Teilprobleme gibt zu entwickeln und diese dann zu verbinden. Um dies tun zu können, müssen die Beziehungen zwischen den einzelnen Verhaltensmustern verstanden und die Interaktionen zwischen ihnen modelliert werden. Damit verschiedene Arten solcher Interaktionen (Lösung von verschiedenen Aufgaben) ermöglicht werden, reicht eine starre Arbitrierungsstrategie wie die ,,Subsumption Architecture" nicht aus. Zielorientierung setzt eine intelligente Verhaltensauswahlstrategie voraus. Im folgenden wird eine Alternativarchitektur vorgestellt, die einige Aspekte dieses Problems löst. Der Schwerpunkt liegt dabei auf der *Adaptivität* (Anpassungsfähigkeit) des Agenten: Der Agent soll selbständig und erfahrungsgestützt eine gute Auswahlstrategie lernen.

5.3 Eine auf Prioritäten basierende Architektur

Die Steuerungsarchitektur, die in diesem Abschnitt vorgeschlagen wird, beruht auf den Ideen, die in der letzten Kombinationsvariante erläutert wurden. Das Basissystem ist reaktiv und wird um die Fähigkeiten der Run-Time-Arbitrierung (um das Steuerungssystem flexibler zu machen) und Zielorientierung (die nötigen Mechanismen zur Bewerkstelligung von verschiedenen Tasks) erweitert. Lernfähigkeiten sorgen für Adaptivität und führen zur Vereinfachung des Designprozesses.

5.3.1 Die Grundidee

Der Ansatz, der hier zum Entwurf von autonomen Systemen vorgeschlagen wird, basiert auf dem Entstehen (*Emergence*) von globalerem Verhalten aus der Interaktion von kleineren Verhaltenseinheiten (siehe Abbildung 5.1).

Ein Agent wird durch eine Menge von *Basis-Verhaltensmustern* charakterisiert. Es wird vorausgesetzt, daß diese Menge für die Erledigung einer Klasse von Aufgaben ausreicht. Für die Realisierung der Basis-Verhaltensmuster können beliebige Methoden benutzt werden. Aufgaben oder Tasks (goals) können erledigt (erfüllt) werden, indem für jede Situation das passende Basis-Verhaltensmuster gewählt wird. Es entstehen somit komplexere Verhaltensmuster. Für die Wahl des passenden Basis-Verhaltensmusters werden

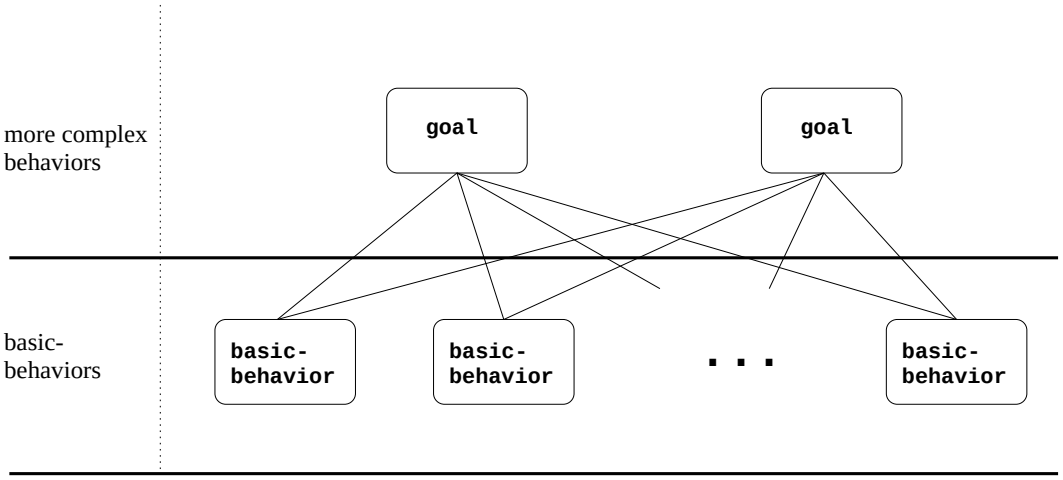


Abbildung 5.1: *Das Entstehen (*Emergence*) von globalerem Verhalten aus der Interaktion von kleineren Verhaltenseinheiten.*

in jedem Zeitschritt *Prioritäten* für alle Basis-Verhaltensmuster berechnet. Das Basis-Verhaltensmuster mit der höchsten Priorität übernimmt die Kontrolle über die Aktoren.

Die wichtigste Komponente in dieser Architektur ist das Basis-Verhaltensmuster. Ein Basis-Verhaltensmuster ist zuständig für einen begrenzten Aspekt der Leistung des autonomen Agenten. Es besteht aus Wahrnehmung (sensing), Aktion und Kommunikation mit den anderen Basis-Verhaltensmustern (siehe Abbildung 5.2).

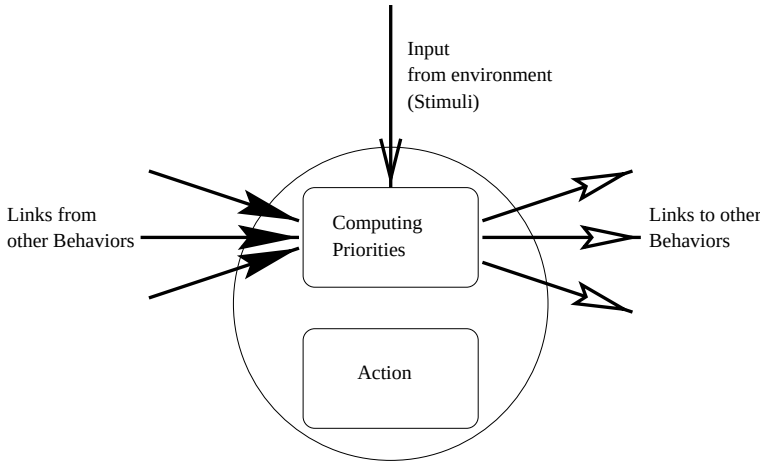


Abbildung 5.2: *Struktur eines Basis-Verhaltensmusters.*

Die Trennung der Aktionskomponente und der Prioritätsberechnungskomponente ei-

nes Basis-Verhaltensmusters sowie die Tatsache, daß die Prioritäten dynamisch sind, erlauben, im Gegensatz zu Lösungen in denen die Verhaltensauswahlstrategie durch Festverdrahtung zum Zeitpunkt der Konstruktion des Roboters vordefiniert ist, eine flexible Arbitrierung zwischen den Verhaltensmustern. Diese Flexibilität erlaubt ihrerseits die Erweiterbarkeit der Kontrollstruktur des Roboters: Jedesmal wenn ein neues Basis-Verhaltensmuster gebraucht wird, kann es ohne Modifikation der globalen Struktur hinzugefügt werden. Ein weiteres Merkmal dieser Architektur ist die Universalität. Universalität bedeutet, daß der Roboter für die Erledigung anderer Aufgaben als diejenigen, für die er ursprünglich konzipiert war, benutzt werden kann. Universalität involviert erstens die Wiederverwendbarkeit von Verhaltensmustern (Verwendung eines Verhaltensmusters in verschiedenen Kontexten (verschiedene Ziele)), zweitens, die Benutzung von nicht festen Interaktionsmodellen zwischen den Verhaltensmustern und drittens, die Benutzung einer variablen Menge von Basis-Verhaltensmustern (wenn erforderlich, werden neue Basis-Verhaltensmuster hinzugefügt). Diese Arten von Agenten sind dann programmierbar: Sie bekommen eine Spezifikation der Aufgabe (nicht wie die Aufgabe gelöst werden kann), die erledigt werden soll, und müssen sie dann selbständig lösen.

5.3.2 Berechnung der Prioritäten

Die Umgebung des Agenten wird in jedem Zeitschritt durch einen Vektor $S = (s_1, s_2, ..., s_n)$ von n Perzeptualwerten beschrieben. Die Perzeptualwerte sind Sensordaten, die durch passende Sensoren erzeugt werden. Der interne Zustand des Agenten kann auch als ,,aus der externen Umgebung entstehend" angesehen werden, indem man annimmt, daß einige Aktoren Teil der externen Umgebung sind. Somit wird die Grenze zwischen Roboter und Umgebung neu entworfen. Einige der Roboterglieder zum Beispiel, können als Teil der Umgebung, und ihre Positionen als Sensordaten für die anderen Aktoren betrachtet werden. Die einzelnen Aktoren sind in diesem Fall eigenständige Agenten, die parallel arbeiten und kooperieren, so daß der Roboter als Ganzes die richtige Funktion erbringt. Ähnliche Ideen sind in [157, 75] zu finden. In den dort beschriebenen Systemen, wird die Kopplung zwischen einem Roboter und seiner Umgebung durch ein Interaktionsmodell definiert, in dem Teile des Roboters sowie alle anderen Agenten (technische und biologische Systeme) als Teil der externen Umgebung angenommen

werden.

Die Prioritäten der einzelnen Basis-Verhaltensmuster bezüglich eines bestimmten Ziels werden in jedem Zeitschritt mit Hilfe der Sensordaten und des Wissens der Designers berechnet. Die Priorität eines Basis-Verhaltensmusters bezüglich eines Ziels reflektiert, wie gut das Basis-Verhaltensmuster ist, um das gegebene Ziel zu erfüllen. Um die Prioritäten der Basis-Verhaltensmuster bezüglich eines Ziels zu berechnen, ist der Designer gefragt, all sein Wissen über das System zu nutzen, um die Abhängigkeit zwischen den Basis-Verhaltensmustern und den Sensoren zu modellieren. Die Abhängigkeit zwischen einem Basis-Verhaltensmuster und den Sensoren wird mit Hilfe von mehreren Funktionen spezifiziert, die die Wirkung der Perzeptualwerte auf die Aktivität des Basis-Verhaltensmusters (bezüglich des gegebenen Ziels) festlegen (siehe Abbildung 5.3). Die Priorität des Basis-Verhaltensmusters wird dann durch die Summierung all dieser Wirkungen berechnet. Für ein Basis-Verhaltensmuster B_i und ein Ziel G, ist die Priorität von B_i zum Zeitpunkt t und bezüglich G gegeben durch:

$$P_G(B_i, t) = \frac{1}{2}\left[1 + \frac{1}{n}\sum_{j=1}^{n} f_{i,j}(s_j(t))\right] \tag{5.1}$$

$f_{i,j}(s_j(t))$ bezeichnet dabei die Abhängigkeitsfunktion, die die Wirkung des Perzeptualwertes s_j auf Basis-Verhaltensmuster B_i zum Zeitpunkt t und bezüglich G beschreibt. n ist die Anzahl der Perzeptualwerte. Das Ergebnis ist ein Wert aus $[0, 1]$.

Die Abhängigkeitsfunktionen brauchen nicht unbedingt in geschlossener Form angegeben zu werden. Sie können auch durch eine Tabelle, ein neuronales Netz oder eine andere passende Methode dargestellt werden. Sie können auch mehrdimensional sein, um die Abhängigkeit mehrerer Sensoren (Perzeptualwerte) auszudrücken.

Damit auch komplexere Sensoren, wie zum Beispiel eine CCD Kamera, benutzt werden können, ist es manchmal notwendig, das Sensorsignal zu zerlegen. Es werden dabei aus dem eingegebenen Sensorsignal Quantitäten extrahiert, die die Basis-Verhaltensmuster unabhängig voneinander beeinflussen (z.B. Abstand- oder Winkelinformation aus einem Bild). Diese Quantitäten können dann so interpretiert werden, als ob sie von verschiedenen Sensoren erzeugt wurden. Für die Modellierung der Abhängigkeit der

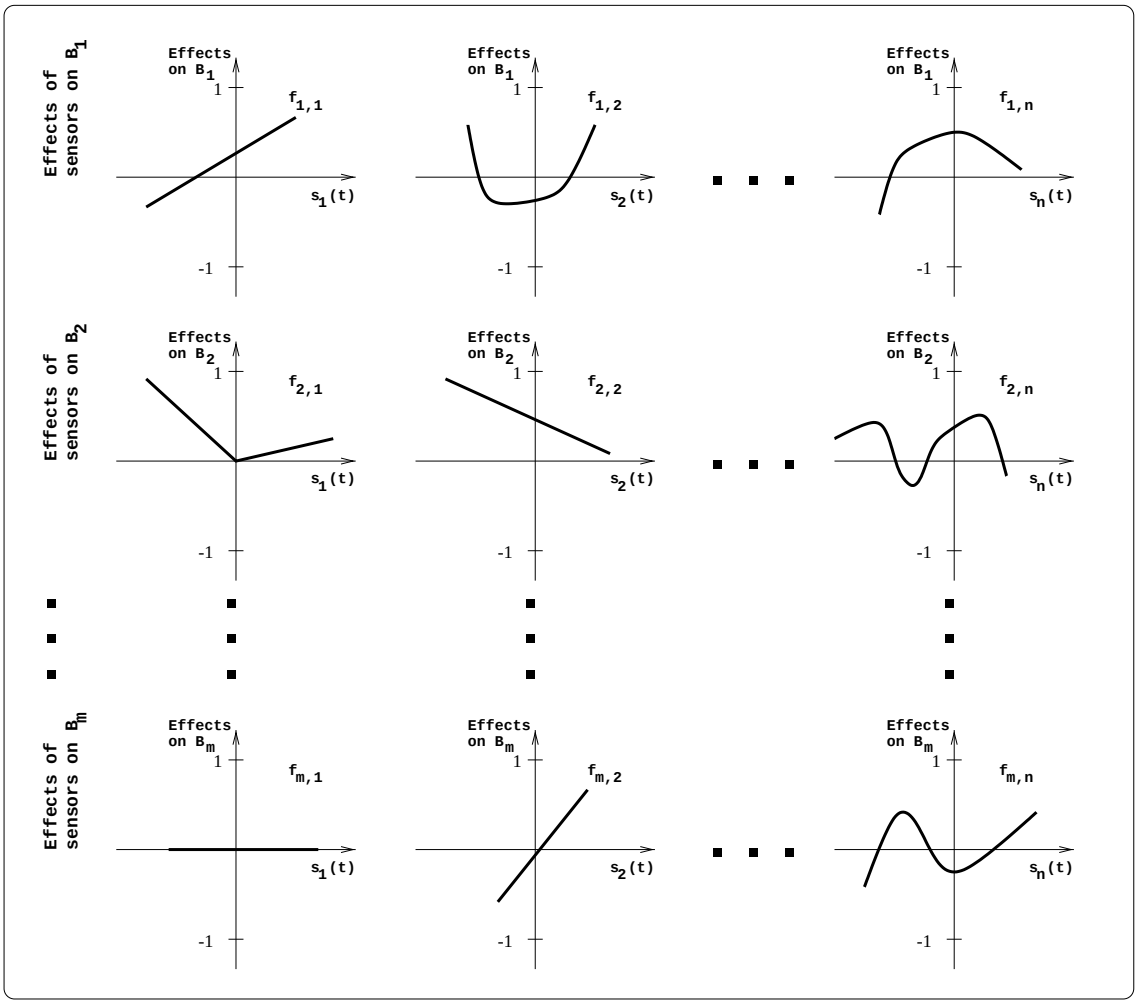


Abbildung 5.3: *Entwurf eines Ziels (goal) durch die Spezifikation der Abhängigkeit zwischen Sensoren und Basis-Verhaltensmustern. Um ein bestimmtes Ziel zu erfüllen, werden die Basis-Verhaltensmuster mit Hilfe von Prioritäten ausgewählt, die in jedem Zeitschritt durch die Summierung der Wirkungen aller Perzeptualwerte auf jedes Basis-Verhaltensmuster berechnet werden. $s_1(t)$ ist der Perzeptualwert, der durch den Sensor Nummer 1 erzeugt wurde. $s_2(t)$ ist der Perzeptualwert, der durch den Sensor Nummer 2 erzeugt wurde usw. .... $B_1, B_2, \ldots, B_m$ sind die Basis-Verhaltensmuster.*

Basis-Verhaltensmuster von nichtlinearen Quantitäten können entweder mehrdimensionale Abhängigkeitsfunktionen benutzt werden oder es soll versucht werden, den Wertebereich dieser Quantitäten durch Clusterung der möglichen Werte zu segmentieren. Ähnliche Ideen für die Segmentierung des Sensordatenraumes für einen mobilen Roboter, der mit Hilfe einer Kamera versucht, einen Ball in ein Tor zu schießen, sind in [10] zu finden. Zum Beispiel wird dort der Bereich für das Bild des Balles in neun Cluster zerlegt. Die Cluster werden durch die Kombination von drei Positionen (Ball links, Ball in der Mitte, Ball rechts) und drei Größen (Ball groß (nahe), Ball mittelgroß, Ball klein (fern)) definiert.

5.3.3 Ein Simulationsbeispiel

Um die oben erwähnten Ideen zu veranschaulichen, wird hier ein Simulationsbeispiel gegeben. Das Simulationsbeispiel ist sehr einfach und wird auch in den nächsten Kapiteln benutzt. Das Beispiel kann man sicherlich mit Hilfe von anderen Methoden besser bearbeiten. In der vorliegenden Arbeit soll es nur dazu dienen, die Ideen und Algorithmen zu verdeutlichen.

In dem Simulationsbeispiel wird eine simulierte dynamische Umwelt als Testfeld benutzt. Ein Agent soll in dieser Umwelt überleben. Es wird gezeigt wie das Ziel „Survival" (Überleben) durch die Spezifikation passender Abhängigkeitsfunktionen entworfen werden kann.

Versuchsanordnung

Die dynamische Umwelt ist eine 25 x 25 Zellenwelt. Abbildung 5.4 zeigt eine Musterumwelt. Es gibt vier Arten von Objekten in dieser Umwelt: Der Agent ("R"), Nahrung ("$"), Feinde ("E") und Hindernisse ("O"). Es wird angenommen, daß die Ränder der Umwelt durch Hindernisse besetzt sind. Am Anfang werden der Agent sowie vier Feinde in ihren initialen Positionen, wie in Abbildung 5.4 gezeigt, plaziert. Fünfzehn Stück Nahrung werden zufällig in unbesetzten Zellen plaziert. In jedem Zeitschritt muß der Agent eins der vier Basis-Verhaltensmuster *„gehe vorwärts"* (move forward), *„gehe rückwärts"* (move backward), *„gehe nach links"* (move left) oder *„gehe nach rechts"* (move right)

auswählen. Jedes dieser Basis-Verhaltensmuster erlaubt dem Agenten, sich zu einer der vier angrenzenden Zellen zu bewegen.

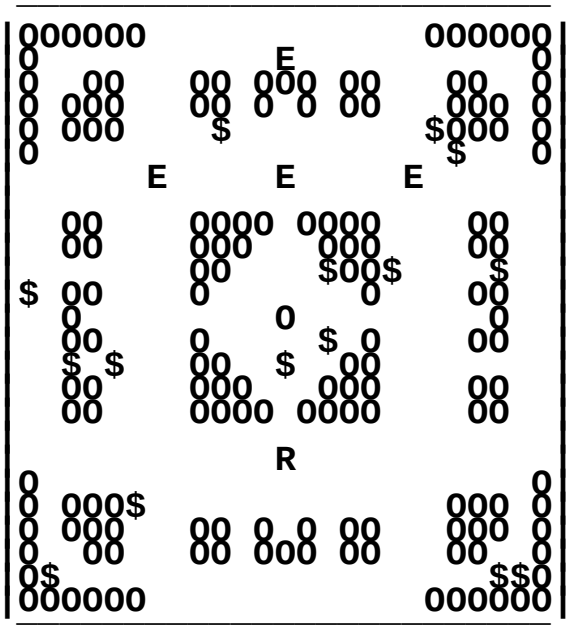


Abbildung 5.4: *Eine dynamische Umwelt bestehend aus einen Agenten (R), Feinden (E), Nahrung ($) und Hindernissen (O).*

Nachdem der Agent sich bewegt hat, darf jeder der Feinde in seiner Zelle bleiben oder sich zu einer angrenzenden Zelle bewegen, die nicht durch Hindernisse, Feinde oder Nahrung besetzt ist. Um dem Agenten die Möglichkeit zu geben, sich vor den angreifenden Feinden zu retten und zu fliehen, beträgt die Geschwindigkeit der Feinde nur 80 % der Geschwindigkeit des Agenten. Die Feinde bewegen sich zwar zufällig, sind aber geneigt, sich in Richtung des Agenten zu bewegen. Die Neigung wird stärker, wenn der Abstand zwischen Agent und Feind kleiner wird.

Ein Spiel endet, wenn der Agent stirbt. Der Agent stirbt, wenn er mit einem Feind oder Hindernis kollidiert. Am Anfang eines jeden neuen Spiels bekommt der Agent eine feste Anzahl von Energieeinheiten. Jedes Stück Nahrung versorgt den Agenten mit fünfzehn Einheiten zusätzlicher Energie, und jede Bewegung kostet den Agenten eine Einheit an Energie. Wenn der Agent keine Energie mehr hat, bleibt er in der gleichen Zelle stehen bis er von einem Feind getötet wird.

Der Agent kann nur einen kleinen Bereich um sich wahrnehmen (sehen). Er hat acht Sensoren: S_F, S_B, S_L, S_R, S_{FL}, S_{FR}, S_{BL}, und S_{BR}. Diese Sensoren werden dazu benutzt, Objekte wahrzunehmen, die jeweils in den Richtungen vorne, hinten, links, rechts, vorne-links, vorne-rechts, hinten-links und hinten-rechts liegen. Jeder dieser Sensoren liefert einen reellen Perzeptualwert aus $[-1, 1]$, der von der Art des entdeckten Objektes und von dem Abstand des Objektes zum Agenten abhängt. Den Perzeptualwert kann man sich als eine abstoßende Kraft (positiver Perzeptualwert) oder eine anziehende Kraft (negativer Perzeptualwert) vorstellen, die auf den Agenten wirkt. Wenn der Agent zum Beispiel Nahrung sieht, würde er sich gerne zur entsprechenden Zelle bewegen. Die Kraft ist in diesem Fall anziehend und der Sensorwert ist negativ. Wenn der Agent aber einen Feind sieht, sollte er fliehen. Die Kraft ist in diesem Fall abstoßend und der Sensorwert ist positiv. Die genauen Sensorwerte, die durch den Sensoren S_F, S_B, S_L und S_R erzeugt werden, sind gegeben durch den Wert s:

$$s = \begin{cases} 0.75/distance^3 & : \quad if \quad object = \text{“}O\text{”} \\ -1.0/distance & : \quad if \quad object = \text{“\$”} \\ 1.0/distance & : \quad if \quad object = \text{“}E\text{”} \end{cases}$$

Die Sensoren S_{FL}, S_{FR}, S_{BL}, und S_{BR} liefern den Wert s':

$$s' = \begin{cases} 0 & : \quad if \quad object = \text{“}O\text{”} \\ -0.5/distance & : \quad if \quad object = \text{“\$”} \\ 0.5/distance & : \quad if \quad object = \text{“}E\text{”} \end{cases}$$

Entwurf des Ziels „Survival”

Aus der Sicht des Agenten ist die Welt nichtdeterministisch, nicht nur, weil die Feinde sich zufällig bewegen, sondern auch, weil die Welt nur teilweise wahrnehmbar ist. Überleben (Survival) in dieser Umgebung bedeutet, daß der Agent ständig versuchen muß, Nahrung zu bekommen, damit er Energie sammelt und sich somit bewegen kann. Gleichzeitig muß er auch Hindernisse vermeiden und sich vor den angreifenden Feinden retten.

Abbildung 5.5 zeigt eine Möglichkeit wie man das Ziel „Survival” entwerfen kann. Es wurden die einfachen Abhängigkeitsfunktionen $f_1(x) = x$ und $f_2(x) = -x$ benutzt. Die Abhängigkeit zwischen dem Basis-Verhaltensmuster „gehe vorwärts” und dem Sensor

S_F (Sensor für Vorwärtsrichtung), zum Beispiel, wird durch die Funktion $f_2(x) = -x$ modelliert. Wenn der Sensorwert groß ist (nahe bei 1), unterliegt der Roboter einer starken abstoßenden Kraft (z.B. Gefahr durch Feind) und sollte sich nach Möglichkeit nicht nach vorne bewegen. Dies wird dadurch erreicht, daß die Aktivierung des Basis-Verhaltensmusters ,,gehe vorwärts", das für die Vorwärtsbewegung zuständig ist, niedrig gehalten wird (nahe bei -1). Auf der anderen Seite, wenn der Sensorwert klein ist (nahe bei -1), unterliegt der Roboter einer starken anziehenden Kraft (z.B. Nahrung ist unmittelbar in der Nähe) und hohe Aktivierung für ,,gehe vorwärts" ist notwendig.

Behaviors \ Sensors	S_F	S_B	S_L	S_R	S_{FL}	S_{FR}	S_{BL}	S_{BR}
move forward	f_2	f_1	f_1	f_1	f_2	f_2	f_1	f_1
move backward	f_1	f_2	f_1	f_1	f_1	f_1	f_2	f_2
move left	f_1	f_1	f_2	f_1	f_2	f_1	f_2	f_1
move right	f_1	f_1	f_1	f_2	f_1	f_2	f_1	f_2

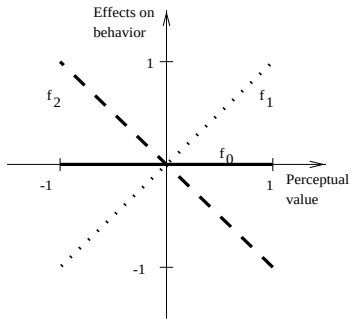


Abbildung 5.5: *Entwurf für das Ziel ,,Survival": Abhängigkeit zwischen den Basis-Verhaltensmustern und den Sensoren modelliert durch die Funktionen* $f_0(x) = 0$, $f_1(x) = x$ *und* $f_2(x) = -x$.

Abbildung 5.6 zeigt die Leistung des Agenten und insbesondere wie diese Leistung mit steigender Input-Information immer besser wird. Auf der X-Achse sind die Zeitschritte eingetragen. Auf der Y-Achse ist für jeden Zeitschritt x die Wahrscheinlichkeit eingetragen mit der der Agent in Schritt x stirbt, d.h. die Wahrscheinlichkeit, daß ein Spiel in diesem Schritt endet (Dichtefunktion für die Zufallsvariable ,,Dauer eines Spiels"). Bei einer guten Kontrollstrategie dauert ein Spiel länger als bei einer schlechten Kontrollstrategie. Im ersten Fall (Kurve $c1$) hat der Agent überhaupt keine Input-Information benutzt. In jedem Zeitschritt hat er ein Basis-Verhaltensmuster zufällig ausgewählt. Im zweiten Fall (Kurve $c2$) wurden nur die vier Sensoren S_F, S_B, S_L und S_R benutzt. Im dritten Fall (Kurve $c3$) wurden alle acht Sensoren benutzt.

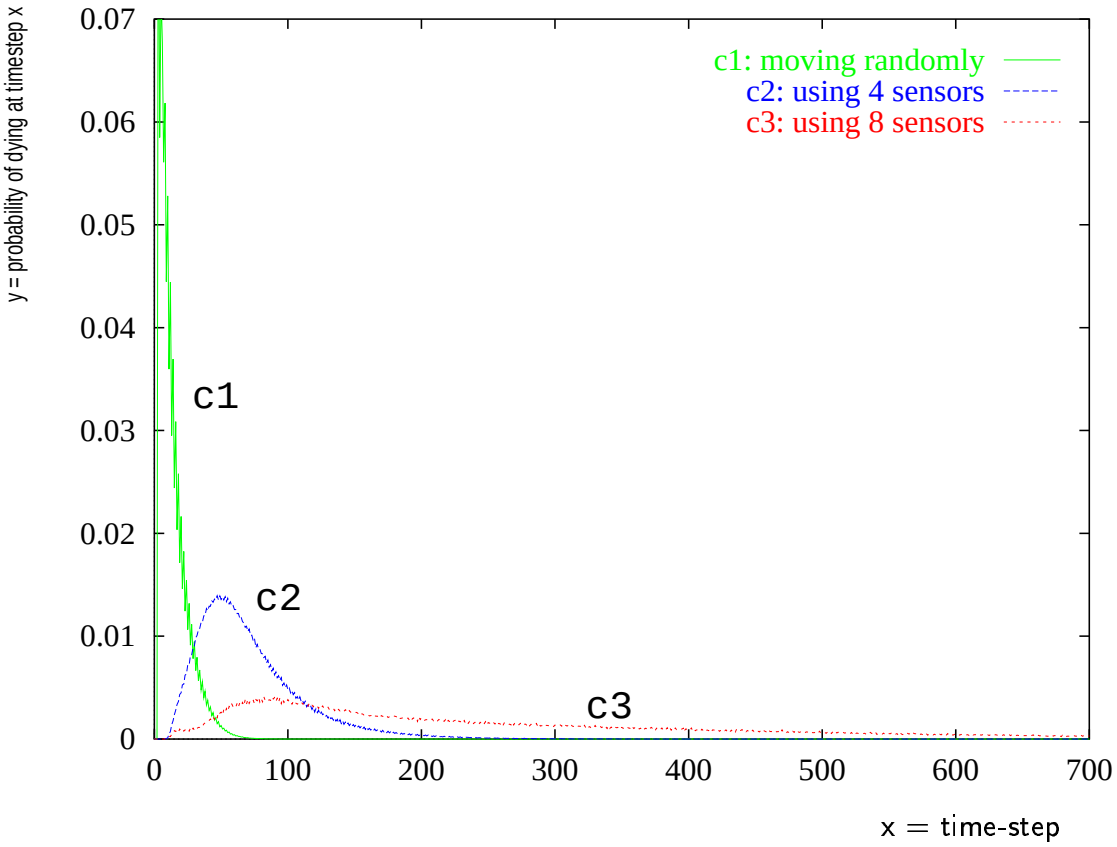


Abbildung 5.6: *Die Leistung des Agenten verbessert sich, wenn mehr Input-Information benutzt wird. c1: Der Agent entscheidet sich zufällig für eins der Basis-Verhaltensmuster. c2: Der Agent benutzt die ersten vier Sensoren und die entsprechenden Abhängigkeitsfunktionen (siehe Abbildung 5.5). c3: Der Agent benutzt alle acht Sensoren und die entsprechenden Abhängigkeitsfunktionen (siehe Abbildung 5.5).*

5.4 Notwendige Erweiterungen

Durch die bisher vorgestellte Architektur wurde eine gute Strecke Richtung Erfüllung der oben erwähnten Anforderungen zurückgelegt. Die Architektur ist äußerst reaktiv. Für die Ermittlung der Prioritäten sind nur sehr einfache Berechnungen erforderlich. Der Agent kann auch bei sehr dynamischen Umgebungen schnell reagieren. Die Prioritäten können sogar in parallel durch die einzelnen Basis-Verhaltensmuster berechnet

werden. An Zielorientierung sind wir auch etwas näher gekommen, denn, es können beliebige Aufgaben (goals) durch die Angabe der passenden Abhängigkeitsfunktionen spezifiziert werden. Die Architektur weist außerdem ein hohes Maß an Flexibilität auf. Die Abhängigkeiten zwischen Sensoren und Basis-Verhaltensmustern können auch zur Laufzeit geändert und angepaßt werden.

Die Architektur wirft allerdings viele neue Fragen und Probleme auf:

- Die Sensoren können kaputt gehen oder falsche Daten liefern. In diesem Fall sind die berechneten Prioritäten meistens falsch und es werden somit die falschen Basis-Verhaltensmuster ausgewählt:

  Wie kann das System robuster gemacht werden?

- Das Wissen des Designers ist meistens unvollständig. Der Designer kann auch nicht alles vorhersagen. Abhängigkeitsfunktionen sind im allgemeinen schwer zu entwerfen:

  Wie kann der Designprozeß automatisiert werden?
  Wie kann dem Designer geholfen werden?
  Wie können die Abhängigkeiten automatisch bestimmt werden?

- Die direkte Lösung von komplexen Aufgaben auf niedriger Ebene, durch Arbitrierung zwischen Basis-Verhaltensmustern, ist meistens schwierig:

  Wie können bereits erworbene Kompetenzen ausgenutzt werden?
  Wie kann das System inkrementell entwickelt werden?

In dem restlichen Teil dieser Arbeit werden diese Fragen und Probleme besprochen und Lösungsansätze vorgeschlagen. Durch Erweiterung und Kombination der Techniken des maschinellen Lernens, die in Kapitel 4 vorgestellt wurden, soll der Agent in die Lage versetzt werden, seine Leistung selbständig zu verbessern und aus seiner Erfahrung zu lernen. Eine sich selbst verbessernde Architektur wird schrittweise entwickelt.

Kapitel 6

VERBESSERUNG DER ROBUSTHEIT

In diesem Kapitel wird versucht, Lösungen für die erste Frage von Abschnitt 5.4 zu finden, nämlich wie die Robustheit der im letzten Kapitel vorgestellten Steuerungsarchitektur verbessert werden kann. Es werden Lernmethoden beschrieben und Simulationsergebnisse diskutiert (siehe auch [57, 59, 58, 60, 61]).

6.1 Robustheit durch Selbstorganisation

Die Komponente eines Basis-Verhaltensmusters, die für die Berechnung seiner Priorität bezüglich eines Ziels zuständig ist (im folgenden ,,Prioritätskomponente" genannt), kann als die Formulierung einer Abbildung zwischen Inputs und Outputs angesehen werden. Der Input besteht aus den Sensordaten, die durch den Vektor $S = (s_1, s_2, \ldots, s_n)$ dargestellt werden. Der Output ist ein Wert, der die Wirkung der aktuellen Situation der Umgebung auf die Aktivierung des Basis-Verhaltensmusters widerspiegelt. Diese Abbildung, die ursprünglich vom Designer des Systems spezifiziert wird, kann durch den Einsatz von geeigneten Lernmethoden angepasst und justiert werden.

Eine geeignete Möglichkeit zur Abbildungsanpassung besteht in der Verwendung von künstlichen neuronalen Netzen. Neuronale Netze sind in der Lage on-line (d.h. während des Betriebs) zu lernen und haben nützliche Generalisierungseigenschaften. Verrauschte und unvollständige Eingabemuster (Sensordaten) können dabei immer noch zu Systemantworten führen, die eigentlich nur mit vollständigen und korrekten Daten möglich wären. Viele Arbeiten, wie zum Beispiel [134, 163, 122] haben gezeigt, daß neuronale Netze zur Entwicklung von *selbstorganisierenden* Kontrollern für Roboter, die in der

realen Welt agieren, benutzt werden können. Wichtig dabei ist vor allem, wie solche adaptive Mechanismen konfiguriert werden sollen, damit eine korrekte und fehlertolerante Input-Output-Assoziation erreicht werden kann.

Die Lernstrategie für die Anpassung und Justierung der Prioritätskomponente eines Basis-Verhaltensmusters, die in diesem Abschnitt vorgestellt wird, basiert auf solchen adaptiven und assoziativen Netzen, die in der Lage sind, Assoziationen für komplexe Sensordaten aus einer begrenzten Menge von einfachen Schlüsselmustern zu lernen. Die Idee dabei ist, daß das neuronale Netzwerk durch die Prioritätskomponenten der verschiedenen Basis-Verhaltensmuster trainiert wird. Es wird eine *topologieerhaltende Karte* benutzt, bei der, als Erweiterung des ursprünglichen Algorithmus für *Selbstorganisation* von Kohonen [81], jeder Unit (Neuron) des Netzwerkes Output-Werte zugeordnet werden. Dieses Lernverfahren wurde vor den bekannteren Multi-Layer-Perceptron- und Backpropagation-Trainingsmethoden bevorzugt, hauptsächlich aufgrund von Berichten über kürzere Zeiten für die Konvergenz der Gewichte [167] und der Robustheit des Algorithmus von Kohonen, die in realen Roboteranwendungen demonstriert werden konnte [111].

Die neuronale Lernstruktur besteht aus der Kombination von zwei Komponenten. Die erste Komponente ist eine selbstorganisierende Karte, die für alle Basis-Verhaltensmuster gemeinsam zur Verfügung steht. Die zweite Komponente ist über alle Basis-Verhaltensmuster verteilt: Jedes Basis-Verhaltensmuster speichert eine Menge von Werten, die seine eigene Version des Outputs der ersten Komponente darstellt. Diese Werte kann man sich als Gewichte für Verbindungen zwischen den Units der ersten Komponente und einer neuen Neuronenschicht (für jedes Ziel (goal) ein Neuron) im Basis-Verhaltensmuster vorstellen. Bild 6.1 zeigt die Integration des assoziativen Netzwerkes in das bisherige Modell zur Berechnung von Prioritäten.

In jedem Zeitschritt werden die Eingabedaten (Sensordaten) gleichzeitig der Kohonen-Schicht und den Prioritätskomponenten der Basis-Verhaltensmuster präsentiert. Die Ausgabe einer Prioritätskomponente wird zum Trainieren des entsprechenden Netzwerkteils benutzt und beeinflußt gleichzeitig, abhängig vom Gewichtsfaktor ψ, die resultierende Priorität. ψ wächst von 0 am Anfang der Lernphase (Priorität wird allein durch die Prioritätskomponente bestimmt) zu 1 am Ende der Lernphase (Priorität wird allein

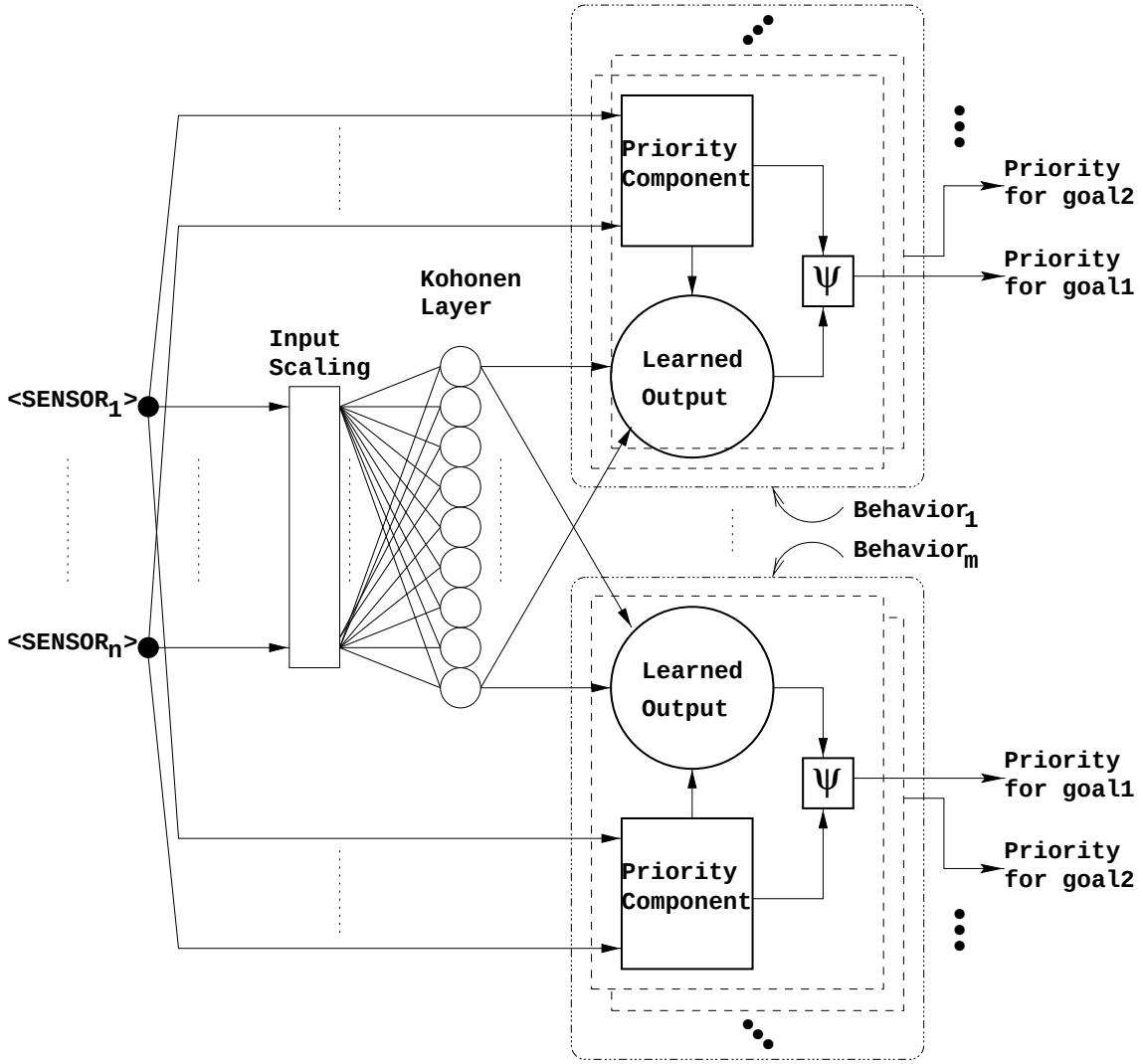


Abbildung 6.1: *Integration eines assoziativen Netzwerkes in das bisherige Modell zur Berechnung der Prioritäten. Es werden zwei Arten von Lernen gleichzeitig angewendet: Unüberwachtes Lernen zur Bildung der selbstorganisierenden Karte und überwachtes Lernen in jedem Basis-Verhaltensmuster zur Integration des Wissens des Designers (Information enthalten in den Prioritätskomponenten) in die neuronale Struktur. Dabei wird jede Region (jedes Neuron) der selbstorganisierenden Karte mit einem Prioritätswert (abhängig vom Ziel) für das Basis-Verhaltensmuster assoziiert.*

durch die Lernstruktur bestimmt). Wenn die Lernphase beendet ist, dann verhält sich die neuronale Lernstruktur wie eine Look-Up-Tabelle, die Generalisierung ermöglicht. Eine einigermaßen genaue Reproduktion des Outputs (Prioritäten für die verschiedenen Basis-Verhaltensmuster) ist auch bei unvollständigem oder falschem Input (Sensordaten) immer noch möglich.

Die Kohonen-Schicht in Abbildung 6.1 besteht aus einem zweidimensionalen Gitter A formaler Neuronen. Die Neuronen dieser Schicht werden durch ihre zweidimensionalen Position $r \in A$ identifiziert. Jedes Neuron r hat einen Vektor $K_r^{(in)}$ von n Gewichten (die gleiche Anzahl von Komponenten wie beim Sensorvektor S) und einen Output-Wert $K_r^{(out)}$ (bei p Zielen (goals) ist $K_r^{(out)}$ ein p-dimensionaler Vektor). Das Erlernen der Priorität eines Basis-Verhaltensmusters bezüglich eines Ziels besteht in der Konstruktion einer Abbildung Φ von Raum V aller möglichen Input-Vektoren (Sensorvektoren) S auf Raum U aller möglichen Prioritätswerte u. Φ erhält man, indem man die Abbildung $\phi : V \rightarrow A$, die jedem Input-Vektor eine Unit des Gitters A zuordnet (*neuronale Karte*), durch das Abbilden von Units von A auf Output-Werte $K_r^{(out)}$ erweitert (siehe Abbildung 6.2).

In Abbildung 6.2 bildet die schattierte Fläche V die Gesamtheit aller möglichen Input-Vektoren. Aus dieser Fläche wird, abhängig von der aktuellen Situation der Umgebung, ein Punkt S als ,,Stimulus" für das Netzwerk ausgewählt. Dieser führt zur Auswahl eines Erregungszentrums r' im Neuronengitter A ($\phi(S) = r'$). Alle Neuronen r in der Nachbarschaft dieses Erregungszentrums (heller) nehmen an einem Adaptationsschritt teil. Dieser besteht in einer ,,Verschiebung" der Vektoren $K_r^{(in)}$ auf S hin. Das Ausmaß der Verschiebung wird durch die Lernschrittweite ϵ und die Funktion $d_{r,r'}$ festgelegt, und ist in Abbildung 6.2 durch unterschiedlich helle Grauwerte für die betroffenen Neuronen angedeutet. ϵ bestimmt die Größe eines einzelnen Adaptationsschritts ($0 \leq \epsilon \leq 1$). Wählt man ϵ als Funktion, die mit der Anzahl der erfolgten Lernschritte von großen Anfangswerten allmählich auf kleine Endwerte absinkt, so kann das System zu Beginn rasch grob die richtigen Gewichte lernen. Für gute Ergebnisse und rasche Konvergenz ist allerdings eine geeignete Wahl für die Abnahme von ϵ wichtig. Bei zu rascher zeitlicher Abnahme frieren die Gewichte ein, noch bevor die Karte einen Gleichgewichtszustand erreicht hat. Bei zu langsamer Abnahme dauert der Vorgang dagegen unnötig lange. Die Wechselwirkungsfunktion $d_{r,r'}$ ist eine vom Abstand $(r - r')$ abhängige Funktion

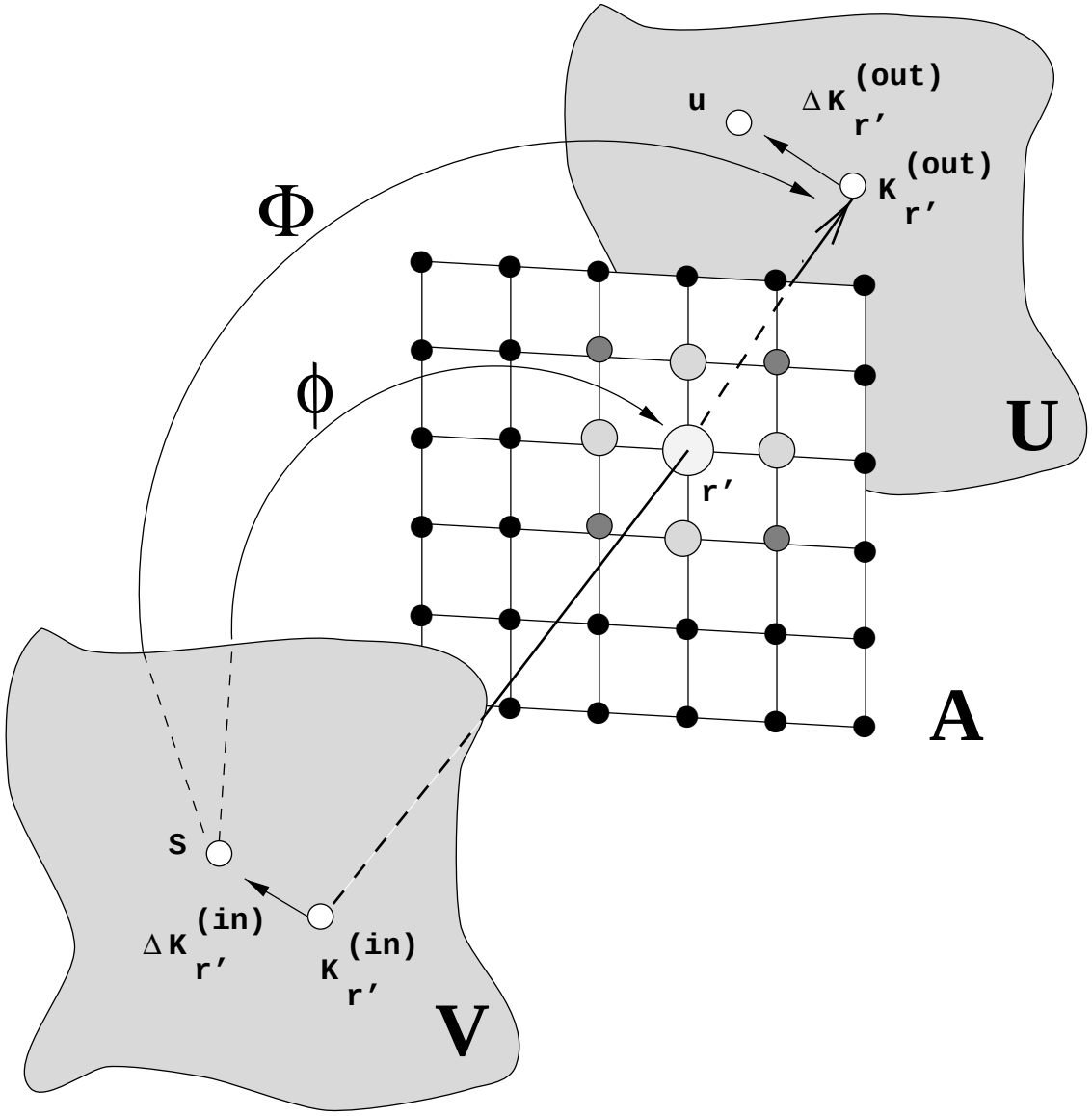


Abbildung 6.2: *Der Adaptationsschritt in der neuronalen Lernstruktur: Der Input-Vektor S selektiert ein Zentrum r' ($\phi(S) = r'$), in dessen Nachbarschaft alle Neuronen ihre Gewichtsvektoren $K_r^{(in)}$ in Richtung auf S „verschieben". Die Adaptation der Output-Werte $K_r^{(out)}$ geschieht gänzlich analog wie auf der Input-Seite: Die Output-Werte der Neuronen in der Umgebung des Erregungszentrums werden in Richtung auf die Vorgabe u verschoben.*

mit Maximum bei $r = r'$, die für große Abstände gegen Null strebt. Eine geeignete Wahl bietet z.B. die Gaußglocke:

$$d_{rr'} = e^{-\frac{(r-r')^2}{2\sigma^2}}. \tag{6.1}$$

Der Radius σ dieser Erregungsfunktion bestimmt die Längenskala, auf der die Eingangsreize Korrekturen an der Karte bewirken. In der Regel ist es vorteilhaft, wenn sich die Grobstruktur der Karte bilden kann, bevor die lokale Feinstruktur entsteht. Dies wird ermöglicht, indem man auch σ als langsam mit der Anzahl der Lernschritte abnehmende Funktion wählt.

Die Adaptation der Output-Werte der Units erfolgt völlig analog zur Adaptation der Gewichtsvektoren. Jedes Basis-Verhaltensmuster benutzt das Erregungszentrum r', um seine eigene Version des Outputs von Gitter A zu verändern. Diese Veränderung ist im hinteren Teil von Abbildung 6.2 angedeutet. Der Output-Wert $K_r^{(out)}$ eines Neurons r, das sich in der Nachbarschaft von r' befindet, wird in Richtung auf die Vorgabe u verschoben. Der Wert von u für die aktuelle Umweltsituation wird von der entsprechenden Prioritätskomponente erzeugt. Die Verschiebung erfolgt gegebenenfalls mit einer eigenen Lernschrittweite ϵ' und einer eigenen Wechselwirkungsfunktion $d'_{r,r'}$.

6.1.1 Lernalgorithmus

Obige Überlegungen lassen sich formal im folgenden Algorithmus zusammenfassen. Anfangszustand ist eine Belegung der Gewichtsvektoren $K_r^{(in)}$ und der Output-Werte $K_r^{(out)}$ mit Werten ohne a-priori-Information, beispielsweise mit Zufallszahlen. Aufgabe des Lernvorgangs ist die allmähliche Verbesserung der so gegebenen Anfangsabbildung Φ mit dem Ziel, die vom Designer des Systems vorgegebene Steuerungsstruktur (Prioritätskomponenten) durch Φ besser und besser zu imitieren. Dazu wird Folgendes in jedem Zeitschritt wiederholt:

- in der Kohonen-Schicht (*unüberwachtes Lernen* zur Bildung einer selbstorganisierenden Karte):
  - bereite den Eingabevektor $S \in V$ vor (z.B. Eingabewerte skalieren)

– bestimme das Erregungszentrum (center of excitation) r' wie folgt:

$$\sum_{j=1}^{n} K_{r',j}^{(in)} s_j = \max_r \sum_{j=1}^{n} K_{r,j}^{(in)} s_j \tag{6.2}$$

– verändere die Gewichtsvektoren der Neuronen wie folgt:

$$K_r^{(in)}(t) = K_r^{(in)}(t-1) + \epsilon d_{r,r'}(S - K_r^{(in)}(t-1)) \tag{6.3}$$

- in jedem Basis-Verhaltensmuster (*überwachtes Lernen* zur Integration des Wissens des Designers (Information enthalten in den Prioritätskomponenten) in die neuronale Lernstruktur):

  – benutzte die Prioritätskomponente (priority component) um einen Wert u zu berechnen.

  – berechne die Priorität P des Basis-Verhaltensmusters wie folgt:

$$P = (1-\psi)u + \psi \frac{1}{1+e^{-K_{r'}^{(out)}}} \tag{6.4}$$

  – verändere die entsprechende Version der Output-Werte der Neuronen des Gitters A wie folgt:

$$K_r^{(out)}(t) = K_r^{(out)}(t-1) + \epsilon' d'_{r,r'}(u - K_r^{(out)}(t-1)) \tag{6.5}$$

In Gleichung (6.4) wird statt $K_{r'}^{(out)}$ der Ausdruck $\frac{1}{1+e^{-K_{r'}^{(out)}}}$ benutzt, um für die resultierende Priorität einen Wert in $[0,1]$ zu erhalten. Der Gewichtsfaktor ψ wächst, wie schon oben erwähnt, von 0 am Anfang der Lernphase zu 1 am Ende der Lernphase und spiegelt in jedem Zeitschritt die Aussagekraft der gelernten Werte wieder. Damit der Lernalgorithmus vernünftige Ergebnisse liefern kann, ist eine geeignete Input-Skalierung sowie eine passende Parameterwahl notwendig. Reelle Zahlen als Input für Kohonen Netze, zum

Beispiel, werden oft vor ihrer Anwendung normalisiert. Eine Einfache Input-Skalierung kann zum Beispiel erreicht werden, indem man jeden Input-Wert durch eine Schätzung seines Maximums teilt. Viele Parameter wie zum Beispiel die Dauer der Lernphase T (Anzahl der Zeitschritte), die Anzahl der Units im Gitter A, ψ, σ, ϵ und ϵ' müssen sorgfältig gewählt werden. Die Dauer der Lernphase ist von der Anzahl der Units im Netzwerk abhängig. Die Anzahl der Units hängt ihrerseits von der gesuchten Abbildung und von der Anzahl der verschiedenen Cluster des Input ab. Die von der Zeit t abhängigen Parameter ψ, σ, ϵ und ϵ' können zum Beispiel durch die allgemeine Funktion

$$f_{xy} \quad = \quad x \cdot e^{y \cdot \frac{t}{T}} \tag{6.6}$$

modelliert werden. Dabei sind x und y Skalierungsfaktoren, die passend gewählt werden sollen, damit das nötige Wachsen oder Schrumpfen für den betroffenen Parameter erreicht werden kann.

Das vorgeschlagene Verfahren erzeugt im Verlaufe der Lernphase eine Wertetabelle für die Funktion Φ. Der besondere Vorteil des Verfahrens liegt dabei in der hohen Adaptivität der Tabellenstruktur. Die Zuordnung zwischen Tabelleneinträgen und Input-Werten ist nicht von Anfang an starr vorgegeben, sondern sie entwickelt sich erst im Laufe einer Lernphase zusammen mit der Belegung der Tabelle. Dies geschieht so, daß die Tabelleneinträge $(K_r^{(in)}, K_r^{(out)})$ entsprechend der Dichte der benötigten Paare $(S, u) \in V \times U$ verteilt werden. Regionen in $V \times U$, die häufig benutzt werden, erhalten entsprechend mehr Tabelleneinträge zugewiesen, so daß dort automatisch eine höhere Auflösung der Input-Output-Abbildung möglich wird. Selten oder nie benutzte Tabelleneinträge werden ,,umgewidmet". Dies ermöglicht eine sehr ökonomische Ausnutzung der verfügbaren Speicherplätze. Eine permanente Plastizität der Tabelle läßt sich durch einen kleinen, nichtverschwindenden Endwert für ϵ realisieren.

Aufgrund des topologieerhaltenden Charakters der Zuordnung zwischen Paaren $(S, u) \in V \times U$ und Neuronen erhalten im Gitter A benachbart liegende Neuronen in der Regel ähnliche Paare zugewiesen. Im Falle eines stetigen Zusammenhangs zwischen Input- und Output-Werten müssen benachbarte Neuronen daher auch ähnliche Output-Werte $K_r^{(out)}$ lernen. Die durch die Wechselwirkungsfunktion $d'_{r,r'}$ bewirkte Verteilung der Lernschritte für die Output-Werte $K_r^{(out)}$ in die Umgebung des bei jedem Lernschritt ausgewählten Erregungszentrums r' stellt somit eine rudimentäre Form der Verallgemeinerung dar,

die den Lernvorgang beschleunigt. Gibt man der Funktion $d_{r,r'}$ zu Beginn der Lernphase eine große Reichweite (σ groß), so nehmen an jedem Lernschritt sehr viele Neuronen teil, auch wenn der Lernschritt für die meisten von ihnen nur ungefähr ,,richtig" ist. Dadurch entwickelt sich schon nach wenigen Lernschritten eine vollständig belegte Tabelle, die eine gute Näherung der korrekten Input-Output-Abbildung bietet. Indem man im weiteren Verlauf der Lernphase die Reichweite der Funktion $d_{r,r'}$ immer weiter verringert, wird die Teilnahme der Neuronen an einem Lernschritt immer selektiver, so daß nach und nach auch feinere Details der Input-Output-Abbildung gelernt werden können.

Die somit erhaltene Generalisierungseigenschaft ist nicht nur wichtig, weil sie die Empfindlichkeit des Systems gegenüber falschen Sensordaten hemmt (verfälschte Input-Signale, deren Merkmale den Originalsignalen noch ähnlich sind, werden auf ein Neuron abgebildet, das sich in der Umgebung des durch die Originalsignale ausgewählten Neurons befindet, und führen daher zu ähnlichen Output-Werten) und somit die Robustheit verbessert, sondern auch, weil sie eine geeignete Diskretisierung des kontinuierlichen Zustandsraumes (Zustand der Umgebung) bietet. Dabei wird, ohne großen Informationsverlust, eine in der Regel sehr große Zustandsmenge auf eine kleine Anzahl von Neuronen (Neuronen in Gitter A) abgebildet. Diese Diskretisierung ermöglicht die Anwendung von Algorithmen des Reinforcement Lernens, die in den Fällen, in denen kein Lehrer vorhanden ist, d.h., die Input-Output-Abbildung a-priori nicht oder nur teilweise bekannt ist, hilfreich sein können. Ein weiterer Vorteil des Lernverfahrens besteht darin, daß parallele Berechnungen möglich sind. Zweck der Berechnungen in Gleichung (6.2) ist die Bestimmung des Erregungszentrums r'. Wenn dies geschehen ist, kann jedes Basis-Verhaltensmuster, unabhängig von den anderen, die Adaptation seiner Version vom Output der Kohonen-Schicht vornehmen. Jedes Basis-Verhaltensmuster ist somit in der Lage, selbständig seine eigene Input-Output-Abbildung zu lernen.

6.1.2 Simulationsergebnisse

In diesem Abschnitt wird der obige Lernalgorithmus mit Hilfe der simulierten, dynamischen Umwelt von Abschnitt 5.3.3 getestet. Es soll die Robustheit des Agenten im Kontext des Ziels ,,Survival" (Überleben) verbessert werden. ,,Survival" bedeutet, daß der Agent Hindernisse vermeiden und sich vor den angreifenden Feinden retten soll.

Gleichzeitig muß er versuchen, Nahrung zu bekommen, damit er Energie sammelt und sich somit bewegen kann. Ein Entwurf für das Ziel ,,Survival" im Sinne der Architektur von Kapitel 5 ist in Abbildung 5.5 dargestellt. Die dadurch definierten Prioritätskomponenten für die verschiedenen Basis-Verhaltensmuster werden während des Lernens als ,,Lehrer" benutzt. Der Adaptationsprozeß wird versuchen, eine neuronale Struktur zu bilden und die Information, die in den Prioritätskomponenten enthalten ist, in diese Struktur zu integrieren.

Abbildung 6.3 zeigt die Leistung des Agenten nach einer Lernphase von 100000 Zeitschritten (Kurve $c2$). Auf der X-Achse sind wiederum die Zeitschritte eingetragen. Auf der Y-Achse ist für jeden Zeitschritt x die Wahrscheinlichkeit eingetragen mit der der Agent in Schritt x stirbt, d.h. die Wahrscheinlichkeit, daß ein Spiel in diesem Schritt endet (Dichtefunktion für die Zufallsvariable ,,Dauer eines Spiels"). Die durch Kurve $c2$ dargestellte Leistung (Leistung des Agenten, wenn er die selbstorganisierende neuronale Lernstruktur benutzt) ist fast so gut wie die des Lehrers (Kurve $c1$: Leistung des Agenten wenn nur die durch den Designer spezifizierten Prioritätskomponenten benutzt werden). Dies bedeutet, daß eine genügend konsistente topologieerhaltende Karte gebildet werden konnte, und daß die in den Prioritätskomponenten enthaltene Steuerungsinformation in die neuronale Karte erfolgreich integriert wurde.

Während der Lernphase, wurde der Agent mit Hilfe von Prioritäten gesteuert, die nur aufgrund von Werten aus den Sensoren S_F, S_B, S_L, und S_R berechnet wurden. Die übrigen Parameter des Lernalgorithmus wurden wie folgt gewählt: ϵ und ϵ' fallend von 1 am Anfang der Lernphase auf 0.000001 am Ende der Lernphase; σ und σ' fallend von 100 am Anfang der Lernphase auf 0.1 am Ende der Lernphase. Die Abhängigkeit dieser 4 Parameter von der Zeit t (Abnahme der Parameter mit der Anzahl der Lernschritte) wurde durch $x(t) = x_{initial}(\frac{x_{final}}{x_{initial}})^{\frac{t}{T}}$ spezifiziert. Dabei steht x für einen dieser 4 Parameter, $x_{initial}$ für dessen Startwert, x_{final} für dessen Endwert und T für die Länge der Lernphase. Für die Kohonen-Schicht wurde ein Gitter von 50x50 Units benutzt. Es wurden auch andere Einstellungen für die Parameter des Lernalgorithmus ausprobiert. Obige Einstellungen lieferten aber die besseren Ergebnisse. Der Parameter ψ wurde während der Lernphase auf 0 gesetzt (Agent wurde nur von den Prioritätskomponenten gesteuert) und danach auf 1 (Agent wurde nur mit den gelernten Prioritäten gesteuert).

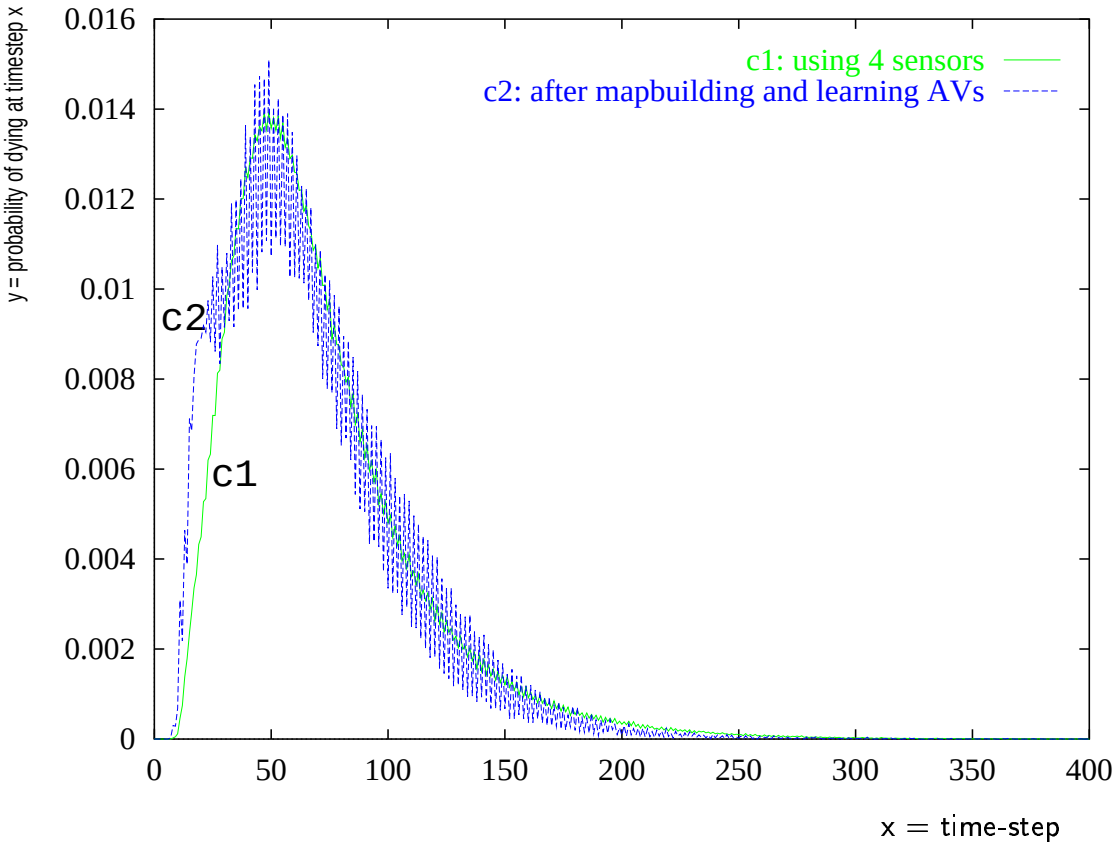


Abbildung 6.3: *Die Leistung des mit Hilfe einer topologieerhaltenden Karte gesteuerten Agenten ist fast genau so gut wie die des „Lehrers" (die vom Designer spezifizierten Prioritätskomponenten). c1: Leistung des Agenten ohne neuronale Karte c2: Leistung des Agenten nachdem die neuronale Karte gebildet wurde.*

Einer der Hauptgründe für die Benutzung des neuronalen Netzwerks, war seine Fähigkeit, mit verrauschten und unvollständigen Sensordaten umzugehen. Es ist daher zu erwarten, daß die Robustheit des Agenten sich verbessert. Die Resultate in Abbildung 6.4 heben dies sehr klar hervor. In diesem Experiment wurde angenommen, daß der Sensor S_F defekt war und immer nur den Wert 0 lieferte (der Agent war auf einem Auge blind). Kurve $c1n$ zeigt die schlechte Performanz, wenn die neuronale Struktur nicht benutzt wird. Kurve $c2n$ hingegen zeigt, daß wenn der Agent die topologieerhaltende Karte und die in ihr gespeicherten Prioritäten benutzt, der defekte Sensor nur eine kleine und unbedeutende Performanzverschlechterung verursacht. Die Kurven $c1$ und $c2$ sind

die von Abbildung 6.3.

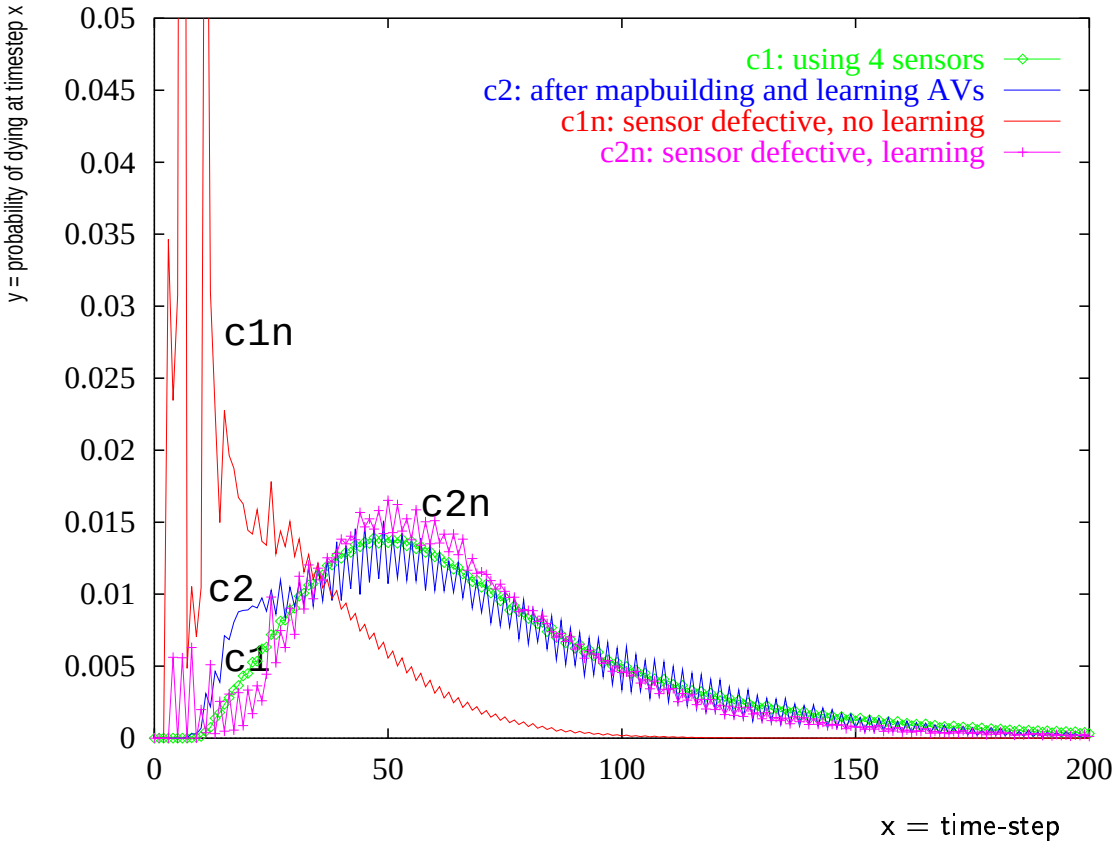


Abbildung 6.4: *Verbesserung der Robustheit des Agenten mit Hilfe einer topologieerhaltenden Karte. c1: Leistung des Agenten ohne neuronale Karte (Kurve c1 von Abbildung 6.3). c2: Leistung des Agenten nachdem die neuronale Karte gebildet wurde (Kurve c2 von Abbildung 6.3). c1n: Leistung des Agenten ohne neuronale Karte bei defektem Sensor S_F. c2n: Leistung des Agenten mit neuronaler Karte bei defektem Sensor S_F.*

Um die Eignung der durch die neuronale Struktur vorgenommene Diskretisierung des Zustandsraums für die Anwendung von Algorithmen des Reinforcement Lernens zu testen, wurde noch ein weiteres Experiment durchgeführt. In diesem Experiment wurde angenommen, daß der Designer des Systems nur die Abhängigkeiten der Basis-Verhaltensmuster von den Sensoren S_F, S_B, S_L, und S_R kennt. Die Abhängigkeiten der Basis-Verhaltensmuster von den Sensoren S_{FL}, S_{FR}, S_{BL}, und S_{BR} waren unbekannt

und wurden mit Hilfe der Abhängigkeitsfunktion $f_0(s) = 0$ modelliert. Der Agent hatte zuerst die Möglichkeit eine topologieerhaltende Karte zu bilden und die vorhandene Steuerungsinformation in sie zu integrieren. Dies geschah genau wie bei der vorherigen Studie mit dem Unterschied, daß in diesem Fall alle acht Sensoren an der Bildung der Karte beteiligt waren. Die Leistung, die der Agent mit der topologieerhaltenden Karte erreichen konnte, ist in Abbildung 6.5 dargestellt (Kurve $c1$).

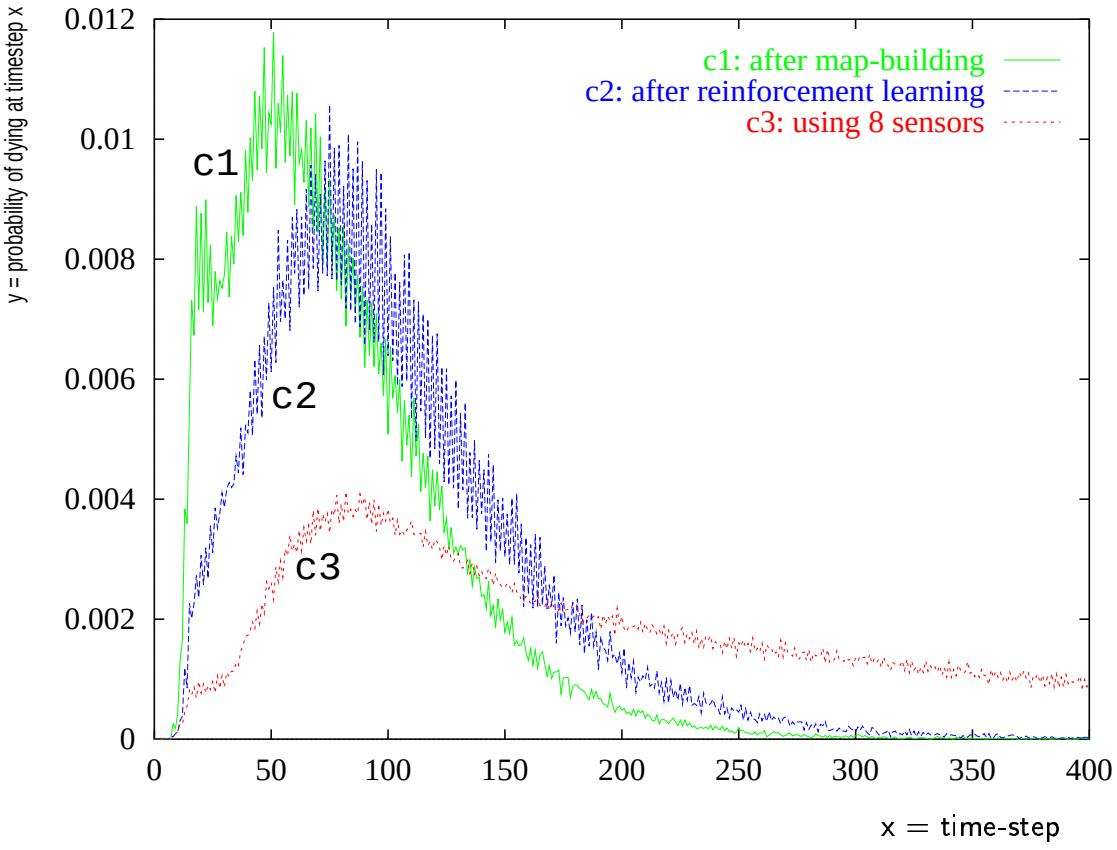


Abbildung 6.5: *Verbesserung der Leistung des Agenten durch Adaptation der topologieerhaltenden Karte mit Hilfe von Reinforcement Lernen. c1: Leistung des Agenten mit einer neuronalen Karte, die ohne Kenntnis der Abhängigkeitsfunktionen für die Sensoren S_{FL}, S_{FR}, S_{BL}, und S_{BR} gebildet wurde. c2: Leistung des Agenten nach Anwendung von Reinforcement Lernen. c3: Leistung des Agenten wenn die Abhängigkeitsfunktionen für alle acht Sensoren bekannt sind und keine neuronale Karte benutzt wird.*

Danach wurden die Output-Werte der Kohonen-Schicht (die erhaltenen Prioritätswerten) mit Hilfe von Reinforcement Lernen verbessert. Reinforcement Lernen wurde wie folgt angewendet: Jedesmal wenn ein Basis-Verhaltensmuster in einer bestimmten Umweltsituation ausgeführt wurde, dann wurde der Output des entsprechenden Neurons des Netzwerks einer Update-Operation unterzogen. Bei der Update-Operation wurde ein Reinforcement-Signal (Bewertungssignal) benutzt, das vom Erfolg des Basis-Verhaltensmusters in der Erfüllung des Ziels des Überlebens abhing. Es wurde eine einfache Version des Q-Learning-Algorithmus (siehe Abschnitt 4.5.3) verwendet. Die Reinforcement-Signale wurden wie folgt festgelegt: -0.8 wenn der Agent stirbt, 0.4 wenn der Agent ein Stück Nahrung findet und 0.0 sonst. Kurve $c2$ in Abbildung 6.5 zeigt die Leistung der resultierenden Kontrollstruktur.

Die Leistung des Agenten hat sich nach Anwendung von Reinforcement Lernen deutlich verbessert. Der Agent konnte in diesem Beispiel aber, selbst nach 10000000 Zeitschritten, die durch Kurve $c3$ (Abbildung 6.5) dargestellte Zielleistung (Leistung des Agenten wenn die Abhängigkeitsfunktionen für alle acht Sensoren bekannt sind und die entsprechenden Prioritätskomponenten direkt, ohne neuronale Struktur, benutzt werden) nicht erreichen. Dies kann daran liegen, daß die Information, die für Reinforcement Lernen zur Verfügung steht, nicht ausreicht, um eine solche Leistung zu erreichen oder daß die gewählte Parameterkonfiguration nicht gut genug ist. Eine effektivere Methode zur Verbesserung der Leistung des Agenten, die auch schneller zu konvergieren scheint, wird im nächsten Kapitel vorgestellt.

6.2 Nutzung der Erfahrung

Die im vorigen Abschnitt vorgestellte Methode zur Verbesserung der Robustheit des Agenten bei falschen oder unvollständigen Sensorsignalen kann nur wirksam eingesetzt werden, wenn zwischen den Merkmalen solcher Signale und den Merkmalen der entsprechenden Originalsignale (unverfälschten Signale) noch ausreichende Ähnlichkeiten bestehen. Wenn dies nicht der Fall ist, dann werden die verfälschten Signale auf ganz andere Regionen als die Originalsignale auf der topologieerhaltenden Karte abgebildet, und es werden somit falsche Entscheidungen getroffen. Abhilfe kann in diesem Fall geschaffen werden, wenn andere Informationsquellen als die Sensordaten herangezogen

werden. Eine solche Quelle stellt die bisherige Erfahrung des Agenten dar. Die bisherige Sequenz von Entscheidungen (Auswahl des passenden Basis-Verhaltensmusters) kann dazu benutzt werden, zukünftige Entscheidungen vorherzusagen. Eine solche Vorhersage kann zum Beispiel in der Form von Auswahlwahrscheinlichkeiten für die verschiedenen Basis-Verhaltensmuster ausgedrückt werden.

In diesem Abschnitt werden solche Auswahlwahrscheinlichkeiten, im folgenden *Erfahrungswerte (EW)* genannt, mit den bisherigen Prioritätswerten, im folgenden *Aktivierungswerte (AW)* genannt, kombiniert, um Robustheitsverbesserungen zu erzielen. Die Priorität eines Basis-Verhaltensmusters B_i zum Zeitpunkt t im Kontext eines Ziels G wird wie folgt berechnet:

$$P_G(B_i, t) = \alpha AW_G(B_i, t) + (1 - \alpha) EW_G(B_i, t). \tag{6.7}$$

Durch den Parameter $\alpha \in [0, 1]$ in Gleichung (6.7) wird festgelegt, wie der Aktivierungswert und der Erfahrungswert auf die resultierende Priorität des Basis-Verhaltensmusters wirken sollen. Der Wert von α kann als Ausdruck für die verläßlichkeit der Sensoren verstanden werden. Ein kleiner Wert von α bedeutet, daß die Sensordaten oft falsch sind. Die daraus berechneten Aktivierungswerte dürfen daher nur eine geringe Wirkung auf die Prioritäten haben. Ein großer Wert von α bedeutet, daß die Sensoren einigermaßen zuverlässig sind. Die Aktivierungswerte werden in diesem Fall in stärkerem Maße bei der Berechnung der Prioritäten berücksichtigt.

Um die Erfahrungswerte berechnen zu können, muß zuerst die Information, die aus der Erfahrung gewonnen wird, passend repräsentiert werden. Die Repräsentationsmethode, die hier verwendet wird, basiert auf der Kommunikation zwischen den Basis-Verhaltensmustern und zwar in Form von *Verstärkung* (Erhöhung der Priorität des anderen) und *Abschwächung* (Minderung der Priorität des anderen). Die Struktur eines Basis-Verhaltensmusters wird daher so erweitert, daß der Austausch von Informationen zwischen den Basis-Verhaltensmustern möglich wird (siehe Abbildung 5.2).

Intuitiv soll ein Basis-Verhaltensmuster ein anderes Basis-Verhaltensmuster verstärken (dessen Priorität und damit auch dessen Chance ausgewählt zu werden erhöhen), wenn bekannt ist, daß, falls das erste ausgewählt wird, dann das zweite (mit einer gewissen Wahrscheinlichkeit) im nächsten Zeitschritt ausgewählt wird. Ein Basis-Verhaltensmuster schwächt ein anderes Basis-Verhaltensmuster ab (dessen Priorität und

damit auch dessen Chance ausgewählt zu werden verringern), wenn bekannt ist, daß, falls das erste ausgewählt wird, dann das zweite (mit einer gewissen Wahrscheinlichkeit) im nächsten Zeitschritt nicht ausgewählt wird.

Die Interaktion zwischen den Basis-Verhaltensmustern wird mit Hilfe eines Netzwerks modelliert. Die Knoten im Netzwerk stellen Basis-Verhaltensmuster dar. Die Verbindungen zwischen den Knoten stellen Beziehungen zwischen den Basis-Verhaltensmustern dar und sind mit Gewichten versehen. Das Gewicht der Verbindung zwischen Knoten B_i und Knoten B_j ist eine Zahl in $[-1, 1]$ und wird mit $w_{i,j}$ bezeichnet.

Basis-Verhaltensmuster B_i verstärkt zum Zeitpunkt t Basis-Verhaltensmuster B_j durch Zusendung von (positiver) Aktivierungsenergie. Der Bertrag dieser Energie ist gleich dem Aktivierungswert von B_i zum Zeitpunkt $(t-1)$ multipliziert mit einem positiven Gewicht $w_{i,j}$ ($w_{i,j} \in (0, 1]$). Basis-Verhaltensmuster B_i schwächt zum Zeitpunkt t Basis-Verhaltensmuster B_j ab durch Zusendung von (negativer) Aktivierungsenergie. Der Bertrag dieser Energie ist gleich dem Aktivierungswert von B_i zum Zeitpunkt $(t-1)$ multipliziert mit einem negativen Gewicht $w_{i,j}$ ($w_{i,j} \in [-1, 0)$). Wenn das Gewicht $w_{i,j}$ gleich 0 ist, dann wird B_j von B_i weder verstärkt noch abgeschwächt. $w_{i,j}$ ist somit ein Ausdruck für die Wahrscheinlichkeit oder Unwahrscheinlichkeit einer Transition von B_i nach B_j. Eine Transition von B_i nach B_j ist ein Übergang von B_i nach B_j , d.h., B_j wird im Zeitschritt t ausgewählt nachdem B_i im Zeitschritt $(t-1)$ ausgewählt wurde. Die Multiplikation mit dem Aktivierungswert von B_i zum Zeitpunkt $(t-1)$ bei der Bestimmung des Betrages der zu sendenden Aktivierungsenergie sorgt dafür, daß Basis-Verhaltensmuster, die zum Zeitpunkt $(t-1)$ einen höheren Aktivierungswert als andere haben (und somit eine höhere Chance haben ausgewählt zu werden), mehr Aktivierungsenergie als andere senden und deshalb den zukünftigen Auswahlprozeß mehr als andere beeinflussen.

Abbildung 6.6 zeigt ein Beispiel für ein Netzwerk von Basis-Verhaltensmustern. Die Basis-Verhaltensmuster B_2 und B_3 senden überhaupt keine Aktivierungsenergie zueinander. Die Basis-Verhaltensmuster B_1 und B_3 benutzen die Verbindungen mit dem Gewicht -0.01, um sich gegenseitig abzuschwächen. Die Basis-Verhaltensmuster B_1 und B_2 benutzen die Verbindungen mit den Gewichten 0.01 und 0.05, um sich gegenseitig zu verstärken. Einige Basis-Verhaltensmuster können auch Aktivierungsenergie zu sich

selbst schicken (z.B. B_3).

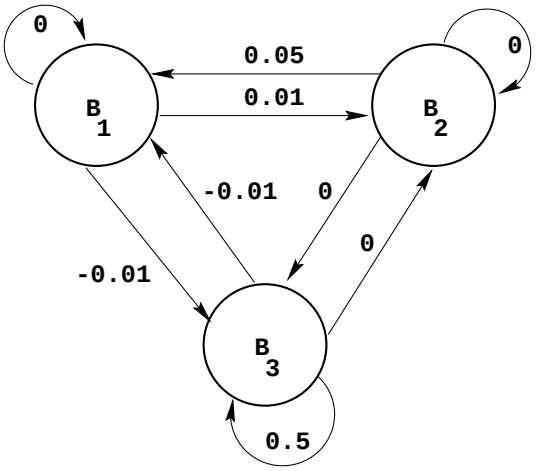


Abbildung 6.6: *Ein Beispiel für ein Netzwerk von interagierenden Basis-Verhaltensmustern.*

Mit Hilfe dieses Interaktionsmodells zwischen den Basis-Verhaltensmustern läßt sich der Erfahrungswert eines Basis-Verhaltensmusters B_i zum Zeitpunkt t bestimmen, indem man alle Aktivierungsenergiebeträge, die B_i zu diesem Zeitpunkt von den anderen Basis-Verhaltensmustern erhält, summiert. Wenn für den Erfahrungswert ein Bereich von $[0, 1]$ gewählt wird, dann läßt sich der Erfahrungswert von B_i zum Zeitpunkt t bezüglich des Ziels G wie folgt berechnen:

$$EW_G(B_i, t) \;=\; \frac{1}{2}\left[1 + \frac{1}{m}\sum_{j=1}^{m} w_{j,i} AW_G(B_j, (t-1))\right]. \qquad (6.8)$$

In Gleichung (6.8) bezeichnet $AW_G(B_j, (t-1)) \in [0, 1]$ den Aktivierungswert von B_j zum Zeitpunkt $(t-1)$ bezüglich des Ziels G. $w_{j,i} \in [-1, 1]$ ist das Gewicht der Verbindung zwischen B_j und B_i und m ist die Gesamtzahl der konkurrierenden Basis-Verhaltensmuster. Wenn $AW_G(B_j, (t-1))$ mit $w_{j,i}$ multipliziert wird, dann ist das Ergebnis in $[-1, 1]$ und somit gilt: $-m \leq \sum_{j=1}^{m} w_{j,i} AW_G(B_j, (t-1)) \leq m$. Die Division durch m, Addition von 1 und anschließende Division durch 2 sorgen dafür, daß das Ergebnis einen Wert in $[0, 1]$ annimmt. Ein Erfahrungswert für B_i in der Nähe von 1 bedeutet, daß B_i in der vorliegenden Situation erfahrungsmäßig eine hohe Priorität haben soll. Ein Erfahrungswert für B_i in der Nähe von 0 bedeutet, daß B_i in der vorliegenden Situation erfahrungsmäßig eine niedrige Priorität haben soll.

Um obige Berechnungen durchführen zu können, sind gute Schätzungen für die Gewichte der Verbindungen im Netzwerk notwendig. Im folgenden Abschnitt wird ein Algorithmus vorgestellt, mit dem diese Gewichte und somit die Erfahrungswerte der Basis-Verhaltensmuster gelernt werden können. Jedes Basis-Verhaltensmuster im Netzwerk beobachtet dabei die Prioritätsänderungen der verschiedenen Basis-Verhaltensmuster und versucht, die Gewichte seiner Verbindungen zu anderen so anzupassen, daß die gegenwärtigen Transitionswahrscheinlichkeiten besser reflektiert werden können.

6.2.1 Lernalgorithmus

Die Priorität eines Basis-Verhaltensmusters zu einem gegebenen Zeitpunkt bezüglich eines Ziels reflektiert die Relevanz dieses Basis-Verhaltensmusters für das Erreichen des Ziels. Ein Basis-Verhaltensmuster wird daher in einer bestimmten Umweltsituation als ,,*passend*" (appropriate) bezeichnet, wenn seine Priorität einen Schwellwert $\theta_{above} \in [0,1]$ überschreitet. Das Basis-Verhaltensmuster wird mit ,,*nicht-passend*" (not appropriate) bezeichnet, wenn seine Priorität einen Schwellwert $\theta_{below} \in [0,1]$, mit $\theta_{below} \leq \theta_{above}$, unterschreitet. Basis-Verhaltensmuster deren Prioritäten zwischen θ_{below} und θ_{above} liegen, gelten als unkritisch in der gegebenen Umweltsituation. Die Lernaufgabe besteht darin, die Gewichte der Verbindungen im Netzwerk inkrementell so zu verändern, daß die daraus berechneten Erfahrungswerte allmählich nur Transitionen zwischen Basis-Verhaltensmustern, die ,,passend" sind, vorhersagen.

Wie schon oben erwähnt, beschreibt das Gewicht einer Verbindung die Wahrscheinlichkeit für eine Transition zwischen den entsprechenden Basis-Verhaltensmustern. Basis-Verhaltensmuster die zum Zeitpunkt $(t-1)$,,passend" waren, erhöhen daher die Gewichte ihrer Verbindungen zu anderen Basis-Verhaltensmustern, die im Zeitpunkt t ,,passend" sind und verringern die Gewichte ihrer Verbindungen zu anderen Basis-Verhaltensmustern, die im Zeitpunkt t ,,nicht-passend" sind. Die Gewichte der übrigen Verbindungen bleiben unverändert. Abbildung 6.7 zeigt wie die Gewichte der Verbindungen eines Basis-Verhaltensmusters, das im vorigen Zeitpunkt ,,passend" war, im Zeitpunkt t (aktueller Zeitpunkt) geändert werden. Gewichte von Verbindungen, die mit ,,+" markiert sind, werden erhöht. Gewichte von Verbindungen, die mit ,,−" markiert sind, werden verringert. Alle anderen Gewichte bleiben unverändert.

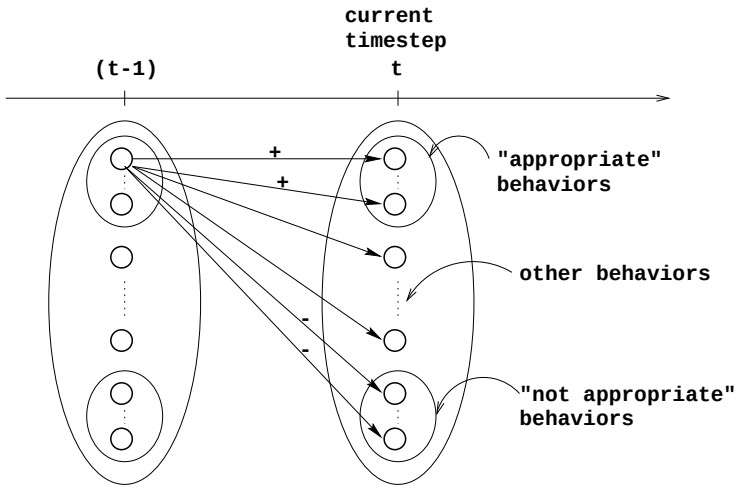


Abbildung 6.7: *Modifikation der Gewichte der Verbindungen während des Lernens. Nur Verbindungen von Basis-Verhaltensmustern, die im vorigen Zeitpunkt ,,passend" waren, werden angepaßt.*

Das Gewicht einer Verbindung von einem Basis-Verhaltensmuster B_i, das im vorigen Zeitschritt ,,passend" war, zu einem Basis-Verhaltensmuster B_j wird in jedem Zeitschritt t wie folgt modifiziert:

$$w_{i,j}(t) = \begin{cases} w_{i,j}(t-1) + \Delta(\eta_i) & : \quad if\ app(B_j, t) \\ w_{i,j}(t-1) - \Delta(\eta_i) & : \quad if\ notapp(B_j, t) \\ w_{i,j}(t-1) & : \quad otherwise \end{cases} \tag{6.9}$$

In Gleichung (6.9) bezeichnet $w_{i,j}(t-1) \in [-1, 1]$ das Gewicht der Verbindung zwischen B_i und B_j im Zeitschritt $(t-1)$. $app(B_j, t) = true$, wenn B_j im Zeitschritt t ,,passend" ist und $notapp(B_j, t) = true$, wenn B_j im Zeitschritt t ,,nicht-passend" ist. η_i ist die Anzahl der Zeitschritte in denen Basis-Verhaltensmuster B_i in der Vergangenheit ,,passend" war und somit auch die Anzahl der Zeitschritte in denen das Gewicht der Verbindung zwischen B_i und B_j einer Adaptationsoperation (gemäß Gleichung (6.9)) unterzogen wurde. Das Ausmaß der Veränderung der Gewichte wird als Funktion von η_i definiert:

$$\Delta(\eta_i) = \begin{cases} h(\eta_i) - h(\eta_i - 1) & : \quad if\ \eta_i \geq 1 \\ 0 & : \quad if\ \eta_i = 0 \end{cases} \tag{6.10}$$

Dabei ist h eine zwischen -1 und 1 monoton wachsende Funktion. h beschreibt wie die Gewichte verändert werden sollen. Ein Beispiel ist $h(x) = tanh(\frac{x}{k})$. Der Parameter

k kann dabei dazu benutzt werden, die Form der Kurve von h zu ändern und den Lernprozeß zu beeinflussen.

6.2.2 Simulationsergebnisse

Es wird wiederum die simulierte dynamische Umwelt von Abschnitt 5.3.3 benutzt. Der Agent soll seine Erfahrung nutzen, um seine Robustheit im Kontext des Ziels ,,Survival" (Überleben) zu verbessern. Am Anfang waren die Gewichte der Verbindungen zwischen den Basis-Verhaltensmustern unbekannt und der Agent mußte sie lernen. Während des Lernens wurde der Agent mit Hilfe von Aktivierungswerten gesteuert, die gemäß des Entwurfs in Abbildung 5.5 berechnet wurden. Es wurden allerdings nur die vier Sensoren S_F, S_B, S_L, und S_R benutzt. Das resultierende Netzwerk ist in Abbildung 6.8 zu sehen. Die dabei benutzen Parametereinstellungen waren wie folgt: die initialen Gewichte und der initiale Wert von η waren für alle Basis-Verhaltensmuster 0; als Wert für θ_{above} wurde in jedem Zeitschritt die höchste aller vier Prioritäten der Basis-Verhaltensmuster genommen; als Wert für θ_{below} wurde in jedem Zeitschritt die kleinste aller vier Prioritäten der Basis-Verhaltensmuster genommen; $h(x) = tanh(\frac{x}{k})$ mit $k = 100$; und die Dauer der Lernphase war 10000 Zeitschritte.

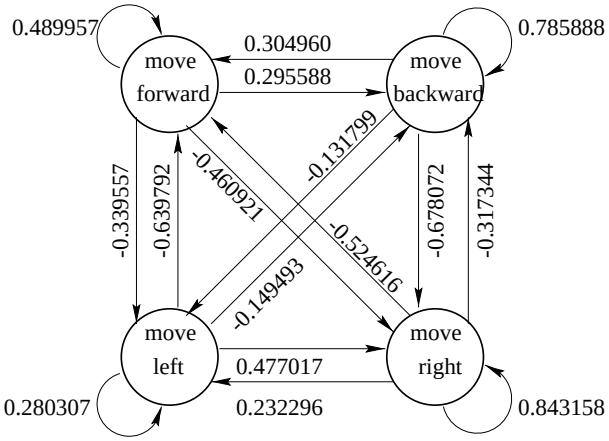


Abbildung 6.8: *Darstellung der Erfahrung, die der Agent nach 10000 Zeitschritten sammeln konnte, als Netzwerk von Basis-Verhaltensmustern.*

Das Netzwerk von Abbildung 6.8 wurde dann anschließend dazu benutzt, um die Erfahrungswerte (EW) zu berechnen. Die Erfahrungswerte wurden mit den Aktivierungs-

werten (AW) kombiniert und zur Steuerung des Agenten verwendet. Der Agent konnte damit eine Leistung erbringen, die überraschenderweise sogar viel besser als die der ursprünglichen Kontrollstruktur (Kontrolle nur mit Hilfe von Aktivierungswerten) war (siehe Kurve $c2$ in Abbildung 6.9). Die Robustheit des Systems hat sich wie erwartet ebenfalls verbessert. Kurve $c3$ in Abbildung 6.9 zeigt die Leistung des Agenten wenn der Sensor S_F defekt ist und immer nur den Sensorwert 0 liefert. Diese Leistung ist trotz Schwankungen immer noch mit der in Kurve $c2$ dargestellten Leistung (Leistung wenn der Sensor nicht defekt ist) vergleichbar. In diesen Experimenten wurde für den Parameter α von Gleichung (6.7) der Wert 0.7 benutzt.

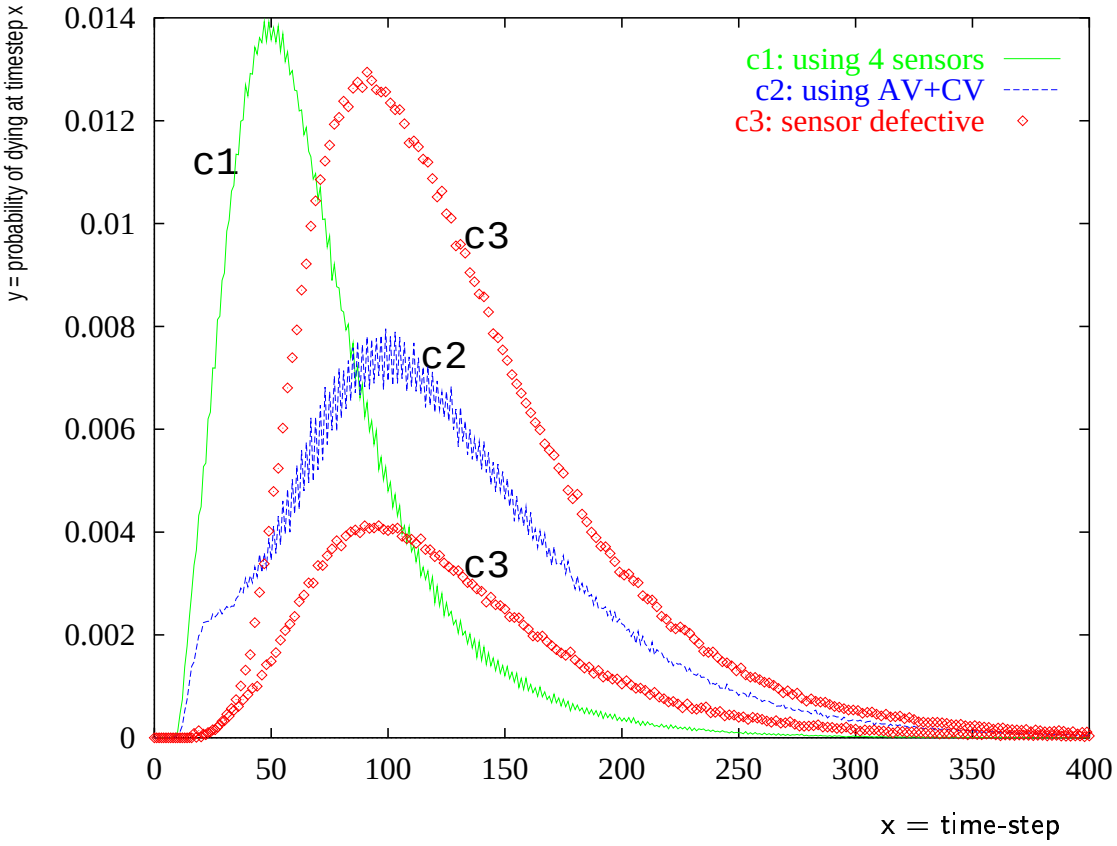


Abbildung 6.9: *Verbesserung der Leistung und der Robustheit des Agenten durch Nutzung der Erfahrung. c1: Leistung des Agenten, wenn nur Aktivierungswerte benutzt werden. c2: Leistung des Agenten, wenn Aktivierungswerte mit Erfahrungswerten kombiniert werden. c3: Leistung des Agenten, wenn der Sensor S_F defekt ist.*

Kapitel 7

AUTOMATISIERUNG DES DESIGNPROZESSES

Bis jetzt wurde angenommen, daß der Designer des Agenten in der Lage ist, die Abhängigkeiten zwischen den Basis-Verhaltensmustern und den Sensoren im Kontext von allen Zielen (goals) zu modellieren und die notwendigen Funktionen anzugeben. Wie schon in Abschnitt 5.4 erwähnt wurde, ist dies in der Regel eine sehr schwierige Aufgabe. Das Wissen des Designers ist meistens unvollständig und reicht nicht aus, um das zukünftige Verhalten des Systems vorherzusagen. Es ist deshalb sehr hilfreich, wenn der Agent versucht, die Abhängigkeiten selber zu bestimmen (erlernen). Der Designprozeß kann auf dieser Weise automatisiert werden. Der Designer braucht dann nicht alles vorzugeben.

Reinforcement Lernen (RL) stellt einen geeigneten Rahmen für die Lösung dieses Problems dar. RL zielt darauf ab, einen Agenten an eine unbekannte Umgebung mit Hilfe von Rewards (Bewertungssignalen) zu adaptieren. Da kurzfristig ausgeführte Aktionen langfristige Konsequenzen nach sich ziehen können, können die Rewards manchmal nur verzögert zur Verfügung stehen. Mittlerweile existieren viele RL-Methoden, zum Beispiel Q-Learning, die verzögerte Rewards (delayed Rewards) und Unsicherheiten berücksichtigen können. Die Ziele, die von diesen Systemen gelernt werden, sind allerdings implizit, d.h., was im Kontext eines Ziels gelernt wird, kann nicht im Kontext eines anderen Ziels angewendet werden (alle bei Q-Learning gelernten Q-Werte zum Beispiel, sind abhängig vom vorgegebenen Ziel). Außerdem konvergieren die meisten dieser Systeme nur langsam [29].

In diesem Kapitel wird eine Lernmethode vorgestellt, die den gleichen Zweck wie RL erfüllt und gleichzeitig eine Lösung für diese Probleme bietet. Die Lernmethode besteht darin, zuerst, mit Hilfe von Exploration, die Umgebung zu identifizieren und danach, mit Hilfe von *Dynamischer Programmierung*, eine optimale Entscheidungsstrategie zu bestimmen. Die Bezeichnung ,,Dynamische Programmierung" steht für eine Sammlung von Algorithmen, die dazu benutzt werden können, um optimale Entscheidungsstrategien zu berechnen. Es wird dabei oft davon ausgegangen, daß ein Modell der Umgebung als Markoff-Entscheidungsprozeß zur Verfügung steht. Im folgenden wird zuerst das Problem formuliert, dann werden die Explorationsmethode sowie die benutzten Algorithmen der Dynamischen Programmierung beschrieben. Danach wird gezeigt, wie diese Methoden kombiniert werden sollen. Anschließend werden Simulationsergebnisse diskutiert. (siehe auch [62]).

7.1 Formulierung des Problems

Die Aufgabe besteht darin, aus der Interaktion zu lernen, wie Ziele (goals) gelöst (erreicht/erledigt) werden können. Der Lernende ist der *Agent*, und die Sache, mit der er interagiert, ist die *Umgebung*. Während der Interaktion wählt der Agent Aktionen (Basis-Verhaltensmuster) aus. Die Umgebung antwortet auf diese Aktionen indem sie dem Agenten neue Situationen präsentiert. Der Agent benutzt in jedem Zeitschritt eine *Entscheidungsstrategie (policy)*, um Situationen auf Aktionen abzubilden. Die Umgebung ist auch die Quelle von Rewards (Bewertungssignale, im folgenden mit r bezeichnet), deren Werte der Agent mit der Zeit zu maximieren versucht.

Die *Dynamik* der Umgebung wird durch die Größen $\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$ spezifiziert mit:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{7.1}$$

und

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}. \tag{7.2}$$

$\mathcal{P}^a_{ss'}$ bezeichnet die Wahrscheinlichkeit für einen Übergang von Zustand (Situation) s nach Zustand s' mit Aktion a und $\mathcal{R}^a_{ss'}$ den Erwartungswert des nächsten Rewards,

wenn ein solcher Übergang stattfindet. Da jedes Ziel eine eigene Reward-Funktion hat, ist die Größe $\mathcal{R}^a_{ss'}$ zielabhängig.

Um die Suche nach guten Entscheidungsstrategien zu strukturieren und zu vereinfachen, werden *Werte-Funktionen (value functions)* benutzt. Diese sind Funktionen der Zustände und schätzen, wie gut es für den Agenten ist (bzgl. eines bestimmten Ziels), sich in einem bestimmten Zustand zu befinden. ,,Wie gut" wird anhand der diskontierten Summe der erwarteten zukünftigen Rewards

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \tag{7.3}$$

definiert ($\gamma \in [0,1]$ Diskontierungsfaktor). Da die Rewards von den Aktionen, die der Agent ausführt, abhängig sind, werden die Werte-Funktionen mit Bezug auf bestimmte Entscheidungsstrategien definiert:

$$V^\pi(s) = E_\pi\{R_t | s_t = s\}. \tag{7.4}$$

Eine grundlegende Eigenschaft von Werte-Funktionen ist, daß sie spezielle rekursive Beziehungen erfüllen. Für jede Entscheidungsstrategie π und jeden Zustand s gilt folgende Beziehung zwischen dem Wert von Zustand s und dem Wert des möglichen Nachfolgezustands s':

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]. \tag{7.5}$$

Dabei ist V^π die Werte-Funktion für Entscheidungsstrategie π, $\pi(s,a)$ die Wahrscheinlichkeit, daß unter π Aktion a in Zustand s ausgeführt wird, und $\gamma \in [0,1]$ der Diskontierungsfaktor. Gleichung (7.5) ist die *Bellman-Gleichung* für V^π (siehe auch Abschnitt 4.5.1).

Eine Entscheidungsstrategie, die besser als alle anderen ist, wird *optimale Entscheidungsstrategie (optimal policy)* genannt. Die Werte-Funktion V^* einer optimalen Entscheidungsstrategie wird durch die Gleichung

$$V^*(s) = \max_\pi V^\pi(s) \tag{7.6}$$

beschrieben und heißt *optimale Werte-Funktion*. Durch Umformung, kann die Bellman-Gleichung für V^* unabhängig von einer bestimmten Entscheidungsstrategie geschrieben

werden:

$$V^*(s) \quad = \quad max_a \sum_{s'} \mathcal{P}^a_{ss'} \; [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \, . \tag{7.7}$$

Gleichung (7.7) ist die *Bellman-Optimalitätsgleichung* (siehe auch Abschnitt 4.5.1). Sie stellt ein Gleichungssystem dar (für jeden Zustand eine Gleichung). Wenn es N Zustände gibt, dann besteht das Gleichungssystem aus N Gleichungen mit N Unbekannten. Wenn die Dynamik der Umgebung ($\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$) bekannt ist, dann kann das Gleichungssystem mit Hilfe von Algorithmen der Dynamischen Programmierung nach V^* gelöst werden. Wenn V^* gefunden wurde, dann ist es relativ einfach, eine optimale Entscheidungsstrategie zu bestimmen. Im nächsten Abschnitt wird eine Methode zur Bestimmung der Dynamik der Umgebung beschrieben.

7.2 Bestimmung der Dynamik der Umgebung

In diesem Abschnitt wird Exploration benutzt, um eine möglichst gute Schätzung für die Dynamik der Umgebung zu erhalten. Der Agent soll während einer Explorationsphase versuchen, jede Aktion in jedem Zustand so oft wie möglich auszuführen. Die Wahrscheinlichkeiten für Zustandsübergänge und die Erwartungswerte der Rewards werden dann durch:

$$\mathcal{P}^a_{ss'} \quad = \quad \frac{TRANS^a_{ss'}}{EXEC^a_s} \tag{7.8}$$

und

$$\mathcal{R}^a_{ss'} = \frac{SUMR^a_{ss'}}{TRANS^a_{ss'}} \tag{7.9}$$

geschätzt. Dabei bezeichnet $TRANS^a_{ss'}$ die Anzahl der Transitionen (Übergänge) von Zustand s nach Zustand s' mit Aktion a, $EXEC^a_s$ die Anzahl der Ausführungen von a in s, und $SUMR^a_{ss'}$ die Summe aller unmittelbaren Rewards für Übergänge von Zustand s nach Zustand s' mit Aktion a. Um eine möglichst gute Schätzung für diese Größen zu finden und somit eine effiziente Identifizierung der Dynamik der Umgebung zu ermöglichen, ist es notwendig, daß die Aktionsauswahl während der Exploration möglichst gleichmäßig erfolgt. Es ist auch notwendig, daß ein möglichst großer Teil des

Zustandsraums entdeckt wird. Es folgen einige Definition, dann wird der Explorationsalgorithmus beschrieben.

Eine Aktion a heißt *n-sampled* in Zustand s, wenn a mindestens n-mal in s ausgeführt wurde. Wenn diese Bedingung nicht erfüllt ist, dann ist a *non-n-sampled* in s. Ein Zustand, der mindestens einmal aufgetreten ist, wird *bekannter Zustand* genannt. Der Prozeß der Identifizierung der Dynamik der Umgebung wird mit Hilfe eines *Identifizierungslevels* ℓ gesteuert. Am Anfang wird ℓ mit 1 initialisiert. In jedem Zeitschritt wird ℓ wie folgt verändert (s bezeichnet den aktuellen Zustand):

if es gibt non-1-sampled Aktionen in Zustand s

then $\ell = 1$.

else if in allen bekannten Zuständen sind alle Aktionen ℓ-sampled

then $\ell = \ell + 1$.

Nachdem der Identifizierungslevel ℓ aktualisiert wurde, wird eine Aktion gemäß der folgenden nichdeterministischen Strategie ausgewählt und ausgeführt (s bezeichnet wiederum den aktuellen Zustand):

if es gibt non-ℓ-sampled Aktionen in s

then wähle eine davon zufällig.

else

mit der Wahrscheinlichkeit p_{random}: wähle eine Aktion zufällig.

mit der Wahrscheinlichkeit $(1 - p_{random})$: wähle die Aktion mit der höchsten Wahrscheinlichkeit p_{max} für einen Übergang zu einem Zustand, für den non-ℓ-sampled Aktionen existieren oder zu einem Zustand, von dem ein Übergang zu einem solchen Zustand möglich ist.

p_{max} wird mit Hilfe der Statistiken für Zustandsübergänge berechnet, die der Agent vom Anfang der Explorationsphase bis zum aktuellen Zeitpunkt ermitteln konnte. p_{random} wird benutzt um das ,,Sammeln von neuem Wissen" und das ,,Benutzen von verfügbarem Wissen" auszugleichen. p_{random} sorgt auch dafür, daß Zyklen (in jedem Zustand immer die gleiche Aktion auswählen) vermieden werden.

7.3 Algorithmen der Dynamischen Programmierung

In diesem Abschnitt werden die Algorithmen der Dynamischen Programmierung, die zur Automatisierung des Designprozesses benutzt werden, kurz beschrieben. Mehr Details und ausführlichere Beschreibungen können zum Beispiel in [158] gefunden werden. Algorithmen der Dynamischen Programmierung erhält man, indem man die Bellman-Gleichungen (Gleichungen (7.5) und (7.7)) in Update-Regeln zur Verbesserung von Näherungen von Werte-Funktionen umwandelt.

Policy Evaluation

„*Policy Evaluation*" bezeichnet die iterative Berechnung der Werte-Funktion einer gegebenen Entscheidungsstrategie. Für eine Entscheidungsstrategie π erhält man eine Sequenz $V_0, V_1, V_2, \ldots$ von Näherungen für die Werte-Funktion, indem man V_0 willkürlich wählt und jede weitere Näherung durch Verwendung der Bellman-Gleichung für V^π als Update-Regel berechnet. Ein Algorithmus für „Policy Evaluation" kann wie folgt formuliert werden:

- Input: π (Entscheidungsstrategie)
- Initialisierung: $V(s) = 0$, für alle s.
- Repeat
  - $\Delta \leftarrow 0$
  - For each s:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    - $\Delta \leftarrow max(\Delta, |v - V(s)|)$

  until $\Delta < \theta$ (eine kleine positive Zahl).
- Output: $V \approx V^\pi$ (Werte-Funktion für π)

Policy Improvement

Die Berechnung der Werte-Funktion für eine Entscheidungsstrategie kann dabei helfen, bessere Entscheidungsstrategien zu finden. Bei „*Policy Improvement*" geht es darum, ausgehend von einer gegebenen Entscheidungsstrategie, für die die Werte-Funktion bekannt ist, eine solche Verbesserung zu erzielen. Der Grundgedanke bei „Policy Improvement" ist es, in jedem Zustand s, diejenige Aktion zu wählen, die laut V^π die beste ist, d.h., einen höheren Zustandswert für s als alle anderen Aktionen liefern würde. Für eine Entscheidungsstrategie π, ist die Entscheidungsstrategie π' mit:

$$\pi'(s) \quad = \quad argmax_a \sum_{s'} \mathcal{P}^a_{ss'} \; [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \tag{7.10}$$

garantiert besser als π (es sei denn π ist bereits optimal). $argmax_a(.)$ ist dabei eine Funktion, die die Aktion a liefert, die den Ausdruck in ihrem Argument maximiert. Die Berechnung von π' wird „Policy Improvement" genannt.

Value Iteration

Sobald eine Entscheidungsstrategie π, mit Hilfe von V^π und „Policy Improvement" verbessert wurde und eine bessere Entscheidungsstrategie π' berechnet wurde, kann man $V^{\pi'}$ bestimmen und π' wiederum verbessern um eine noch bessere Entscheidungsstrategie π'' zu finden. Dieser Prozeß kann dann beliebig oft wiederholt werden. Da jede durch „Policy Improvement" berechnete Entscheidungsstrategie garantiert besser als die vorhergehende ist, muß der Verbesserungsprozeß zu einer optimalen Entscheidungsstrategie und zu einer optimalen Werte-Funktion konvergieren. „*Value Iteration*" stellt eine Methode dar, in der „Policy Evaluation" und „Policy Improvement" wirkungsvoll kombiniert werden, um optimale Entscheidungsstrategien zu berechnen. Ein Algorithmus für „Value Iteration" kann wie folgt formuliert werden:

- Initialisiere $V(s)$
- Repeat
  - $\Delta \leftarrow 0$
  - For each s:

* $v \leftarrow V(s)$
* $V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
* $\Delta \leftarrow max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (eine kleine positive Zahl).

- Output: Eine deterministische Entscheidungsstrategie π mit:
  $\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$

7.4 Der Gesamtalgorithmus

Es sei hier noch einmal daran erinnert, daß es darum geht, den Designprozeß durch Verwendung von Techniken des maschinellen Lernens zu automatisieren. Der Agent soll, selbständig lernen, Ziele zu erreichen (Aufgaben zu erledigen). Dies geschieht indem er versucht, möglichst gute Näherungen für die Prioritäten der Basis-Verhaltensmuster (im Kontext von allen Zielen gleichzeitig) zu bestimmen (lernen). Eine erste Näherung für die Prioritäten existiert bereits (Wissen des Designers).

Der Gesamtalgorithmus für die Automatisierung des Designprozesses durch Lernen sieht wie folgt aus:

- bestimme mit Exploration eine möglichst gute Näherung für die Dynamik ($\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$) der Umwelt.
- für jedes Ziel:
  - berechne mit Hilfe von *Policy Evaluation* die Werte-Funktion für die bereits vorhandene Entscheidungsstrategie (gegeben durch die vorhandenen Prioritäten)
  - benutze diese Werte-Funktion und *Value Iteration* um eine optimale Entscheidungsstrategie für das Ziel zu bestimmen.
  - transformiere diese Entscheidungsstrategie in Prioritäten für die verschiedenen Basis-Verhaltensmuster. Die Priorität für Basis-Verhaltensmuster a in Situation s ist gegeben durch: $\sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$. V ist die Werte-Funktion der Entscheidungsstrategie.

7.5 Simulationsergebnisse

Es wird wiederum die simulierte dynamische Umwelt von Abschnitt 5.3.3 benutzt (mit folgenden geringfügigen Änderungen: Der Agent bekommt jetzt am Anfang eines Spiels nur noch 49 Energieeinheiten (früher waren es 2000 Einheiten) und stirbt sofort, wenn er keine Energie mehr hat (früher ist er stehen geblieben bis er mit einem Feind kollidierte). Jedes Stück Nahrung versorgt den Agenten mit 10 zusätzlichen Energieeinheiten (früher waren es 15 Einheiten). Der Agent stirbt jetzt insgesamt schneller, vor allem wegen Energiemangels. Dies erklärt auch die Spitzen in den Kurven in diesem Abschnitt.). Der Agent soll den obigen Lernalgorithmus benutzen, um das Ziel „Survival" (Überleben) zu lernen. Er erforscht zuerst die Umwelt um eine Schätzung für ihre Dynamik zu bestimmen. Danach benutzt er die gewonnene Information um seine Entscheidungsstrategie zu optimieren.

Um die Leistung des Agenten zu evaluieren, werden neben den bisherigen Kurven (Dichtefunktionen für die Zufallsvariable „Dauer eines Spiels") zusätzlich die Zeitintervalle $I_0, \ldots, I_{16}$ (siehe Tabelle in Abbildung 7.1) benutzt. Für jedes Intervall $I_i (i = 0, \ldots, 16)$ bezeichnet $Pr(I_i)$ die Wahrscheinlichkeit mit der der Agent innerhalb von Intervall I_i stirbt (d.h. ein Spiel endet). Die Leistung des Agenten wird dann gemäß folgender Formel berechnet:

$$Performance \;=\; \sum_{i=1}^{16} i.Pr(I_i). \tag{7.11}$$

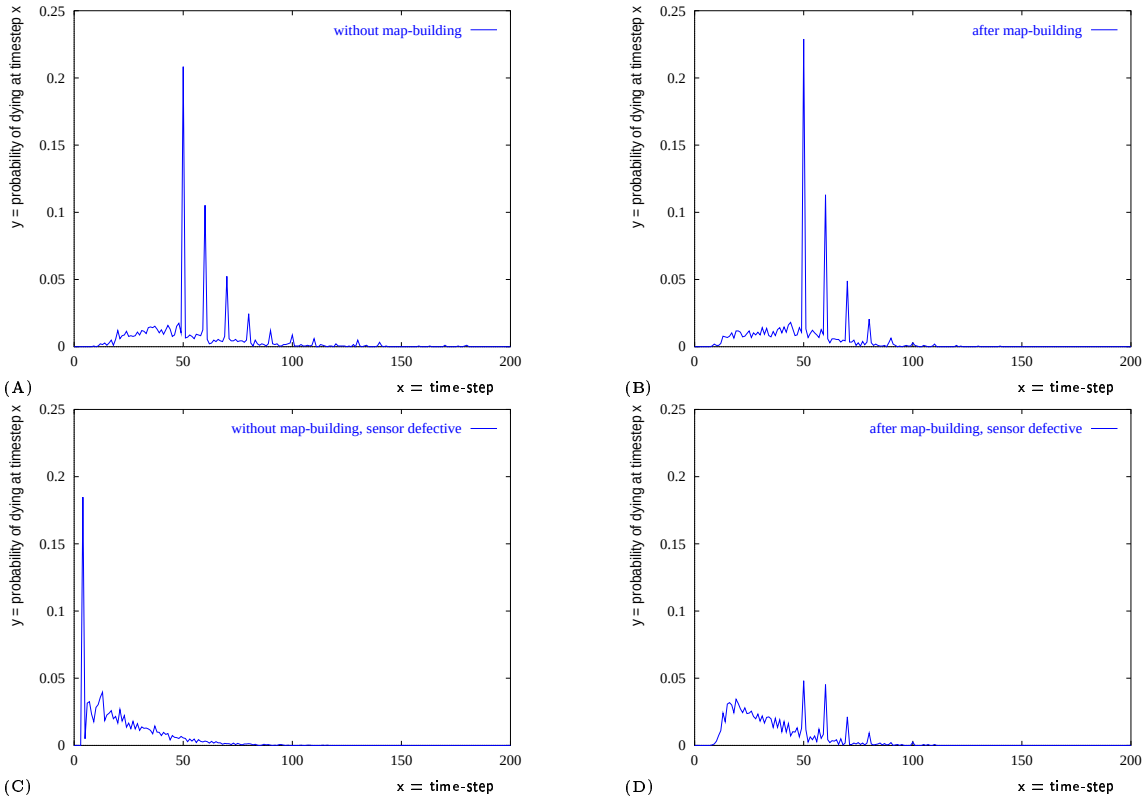
Dadurch werden gute Kontrollstrukturen (d.h. Kontrollstrukturen, die zu längeren Spielen führen) besser bewertet.

In einem ersten Experiment sollte untersucht werden, wie sich der Lernalgorithmus verhält, wenn von Anfang an genügend Information über das System vorhanden ist, d.h., ob durch den Lernalgorithmus eine bereits gute Kontrollstruktur sich noch verbessern läßt. Der Agent benutzte die vier Sensoren S_F, S_B, S_L, und S_R. Für die initiale Kontrollstruktur wurden die Abhängigkeiten zwischen den Sensoren und den Basis-Verhaltensmustern im Kontext des Ziels „Survival" wie bisher modelliert (siehe Abbildung 5.5). Um eine passende Diskretisierung des Zustandsraums zu ermöglichen und damit eine wichtige Voraussetzung für die Anwendung des obigen Lernalgorithmus, nämlich die Generalisierung über Zustände, zu schaffen, hatte der Agent zuerst versucht,

mit Hilfe der Lernmethode, die in Abschnitt 6.1 beschrieben wurde, eine topologieerhaltende Karte zu bilden und das vorhandene Wissen in sie zu integrieren. Das benutzte selbstorganisierende Netzwerk war ein zweidimensionales Netz mit 50x50 Zellen. Abbildung 7.1 zeigt die Leistung des Agenten mit und ohne topologieerhaltende Karte ((A): Leistung ohne Karte. (B): Leistung mit Karte. (C): Leistung ohne Karte; Sensor defekt. (D): Leistung mit Karte; Sensor defekt). In der Tabelle sind die Wahrscheinlichkeiten für das Sterben in den Intervallen $I_0, .., I_{16}$ zusammengefaßt. Die Leistungswerte wurden gemäß Gleichung (7.11) berechnet. Für die Bildung der Karte wurden 10000 Zeitschritte benötigt. Wie zu erwarten war, hat sich durch die Benutzung der topologieerhaltenden Karte die Leistung des Agenten, wegen der Diskretisierung des Zustandsraums, zwar etwas verschlechtert (vergleiche Kurve (A) mit Kurve (B) in Abbildung 7.1). Die Robustheit hat sich aber deutlich verbessert (vergleiche Kurve (C) mit Kurve (D) in Abbildung 7.1; der Sensor S_B war dabei defekt und lieferte immer den Sensorwert 0).

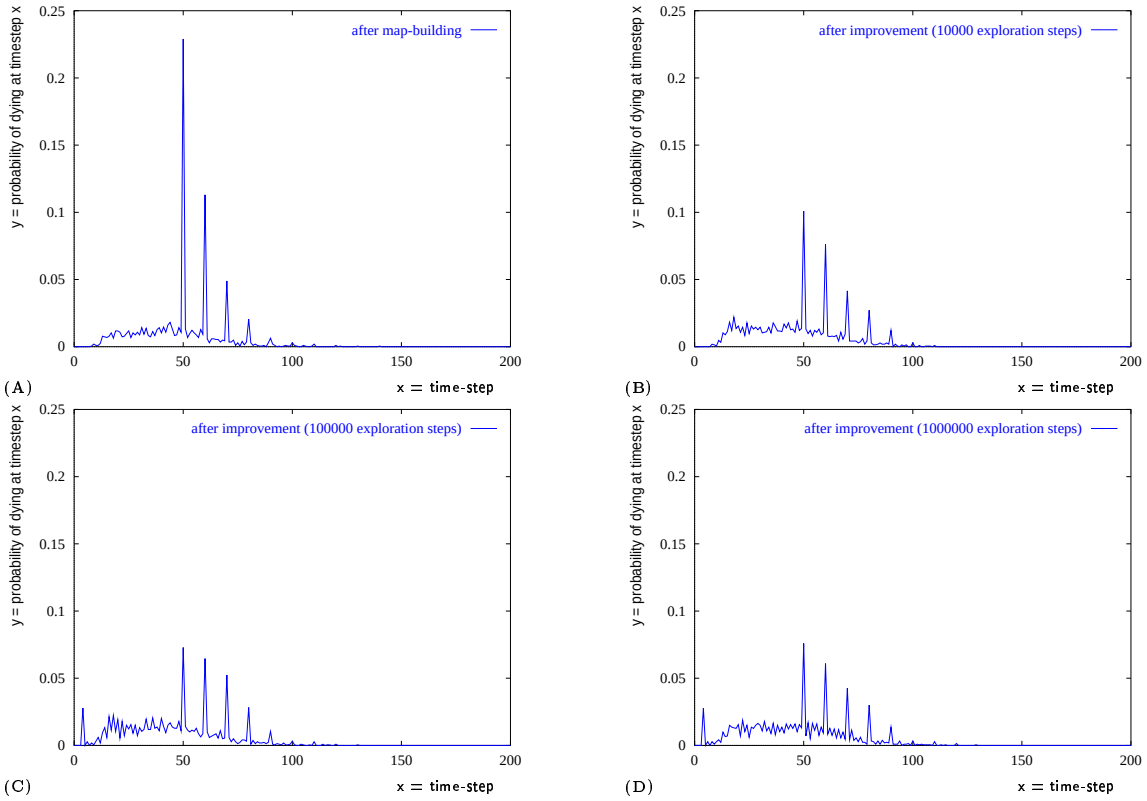
Danach wurde der Lernalgorithmus vom vorigen Abschnitt auf die erhaltene Kontrollstruktur angewendet, um die Prioritäten der Basis-Verhaltensmuster im Kontext des Ziels „Survival" zu optimieren. Die benutzten Reward-Signale waren wie folgt: -0.8 wenn der Agent stirbt, 0.4 wenn der Agent ein Stück Nahrung findet, und 0.0 sonst. Abbildung 7.2 zeigt die Leistung des Agenten nach unterschiedlich langen Explorationsphasen ((B): Leistung nach 10000 Explorationsschritten. (C): Leistung nach 100000 Explorationsschritten. (D): Leistung nach 1000000 Explorationsschritten). Kurve (A) stellt die Anfangsleistung (Leistung vor der Optimierung: Die gleiche Kurve wie in Abbildung 7.1(B)) dar. Die Leistung wurde in allen Fällen deutlich besser. Die „optimale" Leistung in Abbildung 7.1(A) (die Leistung wenn keine Diskretisierung des Zustandsraums vorgenommen wird und wenn die Abhängigkeiten genau spezifiziert werden können) konnte aber selbst nach langen Explorationsphasen nicht erreicht werden. Dies liegt unter anderem an dem durch Diskretisierung verursachten Informationsverlust.

Ein weitere interessante Frage ist, wie sich der Lernalgorithmus verhält, wenn kein oder nur wenig Wissen über das System vorhanden ist. In einem zweiten Experiment wurde angenommen, daß der Designer nur die Abhängigkeiten für Sensor S_R kennt. Die Wirkungen aller anderen Sensoren waren unbekannt und wurden mit Hilfe der Funktion $f_0(s) = 0$ modelliert. Der Agent durfte zuerst eine topologieerhaltende Karte bilden und diese Information in sie integrieren. Die Leistung, die mit der resultierenden Kontroll-



	(A)	(B)	(C)	(D)
I0 = [0, 50]	0.566830	0.623288	0.925614	0.811394
I1 =] 50, 60]	0.180033	0.203033	0.036007	0.105449
I2 =] 60, 70]	0.092744	0.092955	0.019621	0.046355
I3 =] 70, 80]	0.065466	0.045499	0.010565	0.020170
I4 =] 80, 90]	0.030005	0.018102	0.004959	0.009908
I5 =] 90,100]	0.024004	0.008806	0.001940	0.004246
I6 =]100,110]	0.013093	0.005382	0.000647	0.002123
I7 =]110,120]	0.006547	0.000978	0.000431	0.000000
I8 =]120,130]	0.009274	0.000978	0.000000	0.000000
I9 =]130,140]	0.006001	0.000489	0.000000	0.000000
I10 =]140,150]	0.001091	0.000000	0.000000	0.000000
I11 =]150,160]	0.000546	0.000000	0.000000	0.000000
I12 =]160,170]	0.001637	0.000000	0.000000	0.000000
I13 =]170,180]	0.002182	0.000000	0.000000	0.000000
I14 =]180,190]	0.000000	0.000000	0.000000	0.000000
I15 =]190,200]	0.000000	0.000000	0.000000	0.000000
I16 =]200,...[	0.000000	0.000000	0.000000	0.000000
Performance	1.119476	0.693249	0.143381	0.332272

Abbildung 7.1: *Diskretisierung Zustandsraums mit Hilfe einer topologieerhaltenden Karte.*

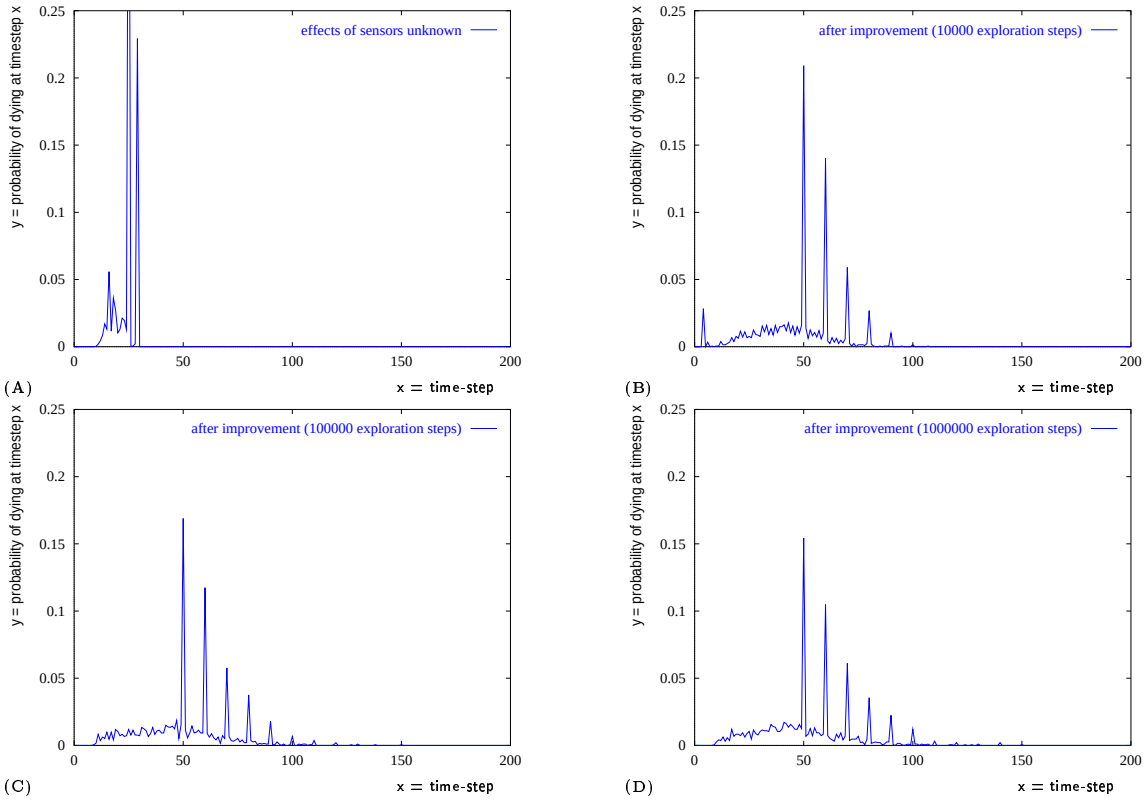


	(A)	(B)	(C)	(D)
I0 = [0, 50]	0.623288	0.609528	0.615034	0.588097
I1 =] 50, 60]	0.203033	0.174218	0.158542	0.164011
I2 =] 60, 70]	0.092955	0.110696	0.118451	0.120900
I3 =] 70, 80]	0.045499	0.062588	0.059681	0.069353
I4 =] 80, 90]	0.018102	0.030827	0.028702	0.034677
I5 =] 90,100]	0.008806	0.008407	0.010934	0.012652
I6 =]100,110]	0.005382	0.003270	0.005011	0.007029
I7 =]110,120]	0.000978	0.000000	0.002733	0.002343
I8 =]120,130]	0.000978	0.000000	0.000456	0.000469
I9 =]130,140]	0.000489	0.000000	0.000000	0.000000
I10 =]140,150]	0.000000	0.000000	0.000000	0.000000
I11 =]150,160]	0.000000	0.000000	0.000000	0.000000
I12 =]160,170]	0.000000	0.000000	0.000000	0.000000
I13 =]170,180]	0.000000	0.000000	0.000000	0.000000
I14 =]180,190]	0.000000	0.000000	0.000000	0.000000
I15 =]190,200]	0.000000	0.000000	0.000000	0.000000
I16 =]200,...[	0.000000	0.000000	0.000000	0.000000
Performance	0.693249	0.768333	0.796811	0.878163

Abbildung 7.2: *Optimierung einer Kontrollstruktur, die bereits gut ist.*

struktur erreicht werden konnte ist in Abbildung 7.3(A) dargestellt.

Danach wurde der Lernalgorithmus angewendet, um diese Leistung zu optimieren. Ab-



	(A)	(B)	(C)	(D)
I0 = [0, 50]	0.999758	0.618705	0.549949	0.553702
I1 =] 50, 60]	0.000000	0.226379	0.207003	0.187174
I2 =] 60, 70]	0.000000	0.097842	0.111226	0.115224
I3 =] 70, 80]	0.000000	0.039808	0.071061	0.069864
I4 =] 80, 90]	0.000000	0.014388	0.033986	0.039625
I5 =] 90,100]	0.000000	0.001918	0.012873	0.017727
I6 =]100,110]	0.000000	0.000480	0.008239	0.008342
I7 =]110,120]	0.000000	0.000000	0.002575	0.003128
I8 =]120,130]	0.000000	0.000000	0.001545	0.002086
I9 =]130,140]	0.000000	0.000000	0.000515	0.002086
I10 =]140,150]	0.000000	0.000000	0.000515	0.000521
I11 =]150,160]	0.000000	0.000000	0.000000	0.000000
I12 =]160,170]	0.000000	0.000000	0.000000	0.000000
I13 =]170,180]	0.000000	0.000000	0.000000	0.000000
I14 =]180,190]	0.000000	0.000000	0.000000	0.000000
I15 =]190,200]	0.000000	0.000000	0.000000	0.000000
I16 =]200,...[	0.000000	0.000000	0.000000	0.000000
Performance	0.000000	0.611511	0.932544	0.986966

Abbildung 7.3: *Lernen mit geringem Anfangswissen.*

bildung 7.3 zeigt die Leistung des Agenten nach unterschiedlich langen Explorationsphasen ((A): Leistung am Anfang. (B): Leistung nach 10000 Explorationsschritten. (C):

Leistung nach 100000 Explorationsschritten. (D): Leistung nach 1000000 Explorationsschritten). Diese Ergebnisse zeigen deutlich die Wirksamkeit des Lernalgorithmus. Nach nur 10000 Explorationsschritten konnte der Agent eine Leistung erreichen, die mit der von Abbildung 7.1(B) vergleichbar ist. Der Lernalgorithmus scheint auch schneller als andere Reinforcement-Lernen-Algorithmen zu konvergieren. Dies wurde zumindest beobachtet, als versucht wurde, ähnliche Optimierungen mit Hilfe von Q-Learning durchzuführen (siehe Abschnitt 6.1.2). Eine andere interessante Beobachtung ist die Tatsache, daß es ziemlich schwierig ist, im ,,letzten zehntel" des Leistungsspektrums Leistungsverbesserungen zu erzielen, d.h. Leistungen zu erreichen, die besser als 90% der optimalen Leistung sind. Dies kann man leicht feststellen, wenn man die Leistung nach 100000 Explorationsschritten (Kurve (C)) mit der Leistung nach 1000000 Explorationsschritten (Kurve (D)) vergleicht. Obwohl man die Länge der Explorationsphase mit dem Faktor 10 multipliziert hat, ist die Leistungsverbesserung nur sehr gering ausgefallen.

Kapitel 8

ERWEITERUNG DER FÄHIGKEITEN DES SYSTEMS

Bis jetzt wurde nur versucht, das Kontrollproblem auf niedriger Ebene, durch Arbitrierung zwischen Basis-Verhaltensmustern, zu lösen. Wie schon in Abschnitt 5.4 erwähnt wurde, läßt sich nicht jedes beliebig komplexe Verhalten auf diese Weise realisieren (erlernen). Durch Koordination von höheren Verhaltensweisen kann die Lösung von komplexen Aufgaben erleichtert werden. Bereits erworbene Kompetenzen können ausgenutzt werden, und eine inkrementelle Entwicklung des Systems wird möglich. Die Goal-Hierarchie in Abbildung 5.1 läßt sich dann beliebig erweitern (siehe Abbildung 8.1) und somit auch die Fähigkeiten des Systems.

In diesem Kapitel wird eine Methode vorgestellt, mit der Ziele (goals) koordiniert werden können. Dadurch können komplexere Ziele (Aufgaben) erreicht (gelöst) werden. Die Methode basiert wiederum auf Techniken des maschinellen Lernens. Der Agent soll selbständig lernen, die Koordination durchzuführen. Die Lernmethode besteht aus der Kombination eines Prozesses zur Bildung einer dynamischen, selbstorganisierenden Karte und Reinforcement Lernen. Die selbstorganisierende Karte ermöglicht dem Agenten, sich an Kontrollparameter (Koordinationsparameter) zu erinnern, die früher zum Erfolg geführt haben, und diese dann in ähnlichen Situationen zu verwenden. Reinforcement Lernen sorgt für die Anpassung und Verfeinerung des Inhaltes der Karte. Eine grobe Skizze für das Zusammenspiel beider Komponenten sieht wie folgt aus: Zuerst wird die selbstorganisierende Karte benutzt, um Kontrollparameter für die jeweilige Situation zu generieren. Danach wird das Verhalten des mit diesen Parametern operierenden Agenten beobachtet. Anschließend wird die Karte abhängig vom Ergebnis angepaßt. Im folgenden

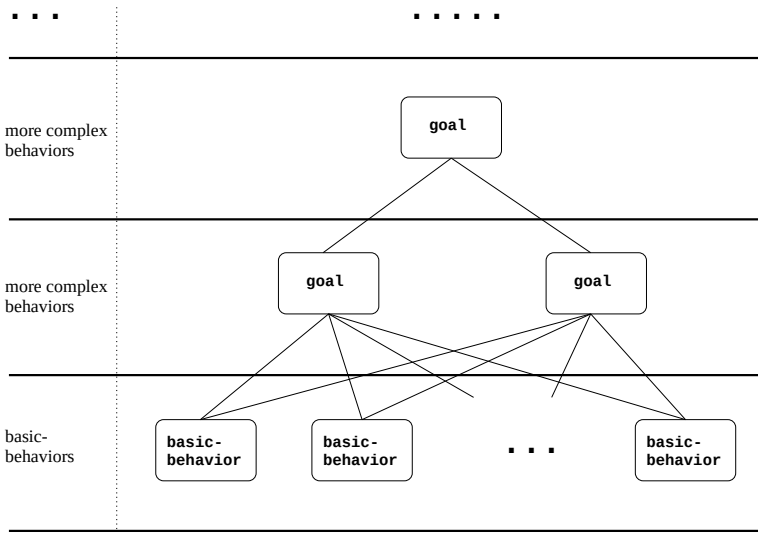


Abbildung 8.1: *Erweiterung der Fähigkeiten eines Agenten. Leichtere Ziele (Verhaltensweisen) werden koordiniert, um komplexere Ziele zu erreichen.*

wird zuerst das Problem formuliert, und dann werden die Details der Lösung beschrieben. Danach wird die Methode durch mehrere Simulationsstudien evaluiert. Siehe auch [64, 63].

8.1 Formulierung des Problems

Die Aufgabe besteht darin, einen Agenten in die Lage zu versetzen, seine Fähigkeiten selbständig zu erweitern. Komplexere Verhaltensweisen sollen durch Kombination von Verhaltensweisen, die der Agent bereits beherrscht, ermöglicht werden. Betrachten wir zum Beispiel einen mit der Basis-Verhaltensmuster-Menge $\{B_1, B_2, \ldots, B_m\}$ ausgerüsteten Agenten, der das komplexe Ziel (Task) G durch Koordination der leichteren Ziele $G_1, G_2, \ldots, G_n$ lösen soll. Es sei angenommen, daß der Agent bereits in der Lage ist, jedes einzelne Ziel G_i für $i = 1, \ldots, n$ zu lösen, d.h., der Agent ist in der Lage, in jedem Zeitschritt t, $P_{G_i}(B_j, t)$ (die Priorität von Basis-Verhaltensmuster B_j bezüglich G_i) für $j = 1, \ldots, m$ zu berechnen (siehe Abbildung 8.2).

Um G zu lösen, muß der Agent in jedem Zeitschritt die Prioritäten der Basis-

basic-behaviors	goal G_1	goal G_2	...	goal G_n	goal G
B_1	$P_{G1}(B1)$	$P_{G2}(B1)$	...	$P_{Gn}(B1)$	?
B_2	$P_{G1}(B2)$	$P_{G2}(B2)$	...	$P_{Gn}(B2)$	?
⋮	...	...	...	...	...
B_m	$P_{G1}(Bm)$	$P_{G2}(Bm)$	...	$P_{Gn}(Bm)$	?

Abbildung 8.2: *In jedem Zeitschritt sind die Prioritäten der Basis-Verhaltensmuster bezüglich der leichteren Ziele $G_1, G_2, \ldots, G_n$ bekannt. Gesucht sind die Prioritäten der Basis-Verhaltensmuster bezüglich des komplexen Ziels G.*

Verhaltensmuster bezüglich G berechnen. Da aber der Agent jedes einzelne Ziel G_i lösen kann, kann dies erreicht werden, indem die Prioritäten der Basis-Verhaltensmuster bezüglich der leichteren Ziele kombiniert werden:

$$P_G(B_j,t) \;=\; \Omega_1(t)P_{G_1}(B_j,t) + \ldots + \Omega_n(t)P_{G_n}(B_j,t) \quad \text{für} \quad j = 1,\ldots,m \quad (8.1)$$

In Gleichung 8.1 ist

$$\Omega(t) \;=\; \begin{pmatrix} \Omega_1(t) \\ \ldots \\ \Omega_n(t) \end{pmatrix} \quad \text{mit} \quad \Omega_1(t) + \ldots + \Omega_n(t) = 1 \qquad (8.2)$$

ein zeitabhängiger Vektor von Koordinationsparametern. Er beschreibt, wie in jedem Zeitschritt t die einzelnen Ziele G_i zur Lösung von G beitragen. G zu lösen bedeutet also jetzt, daß für jede Situation (d.h. eine Konstellation der Prioritäten $P_{G_i}(B_j,t)$) ein passender Vektor $\Omega(t)$ bestimmt werden soll. Das Problem der Erweiterung der Fähigkeiten des Agenten kann daher als das Aufbauen einer Abbildung zwischen Input-Parametern (die Prioritäten der Basis-Verhaltensmuster bezüglich der leichteren Ziele,

die koordiniert werden sollen) und Kontrollparametern (Koordinationsparametern) formuliert werden. Eine solche Abbildung kann zur Designzeit nur aufgestellt werden, wenn genügend Information über das System und seine Dynamik vorhanden ist. Da dies in der Regel nicht der Fall ist, muß das System nicht nur in der Lage sein, die Abbildung anzuwenden, sondern auch, sie durch Einbeziehung seiner Erfahrung zu erlernen und zu verfeinern. Wie dies erreicht werden kann, wird im restlichen Teil dieses Kapitels diskutiert.

8.2 Kartenbildungsprozeß

Wie in Abschnitt 6.1 ist unter Kartenbildung nicht das Konstruieren einer geographischen Karte zu verstehen, vielmehr sollte man sich so etwas wie ,,sich über bestimmte Erfahrungen Notizen machen" vorstellen. Zweck des Kartenbildungsprozesses ist es, Feature-Entdeckung oder Clustering durchzuführen, d.h., eine Abbildung zwischen Inputs und statistisch auffallenden Features der Input-Population zu konstruieren, die den Aufbau von Clustern von Input-Mustern mit ähnlichen Features erlaubt. Als grundlegende Lernstruktur wird ein selbstorganisierendes neuronales Netz [81] benutzt. Im folgenden wird beschrieben, wie diese Lernstruktur erweitert werden soll, um das Problem der Koordinierung von Zielen zu lösen.

Ausgangspunkt: Selbstorganisierendes Netzwerk

Ausgangspunkt sei ein eindimensionales selbstorganisierendes neuronales Netzwerk (siehe Abbildung 8.3(A)). Jede Zelle i des selbstorganisierenden Netzwerks hat einen eigenen Gewichtsvektor W_i. Jeder Input-Vektor I (in unserem Fall besteht der Input-Vektor aus den Prioritäten der Basis-Verhaltensmuster bezüglich der zu koordinierenden Ziele) wird allen Zellen zugeführt. Die ,,Best-Matching-Zelle" oder auch Erregungszentrum (die Zelle deren Gewichtsvektor den kleinsten Abstand zum Input-Vektor hat), d.h., die Zelle k, die die Bedingung:

$$d_k = \|I - W_k\| \quad \leq \quad d_i = \|I - W_i\| \qquad \text{für alle Zellen} \quad i \tag{8.3}$$

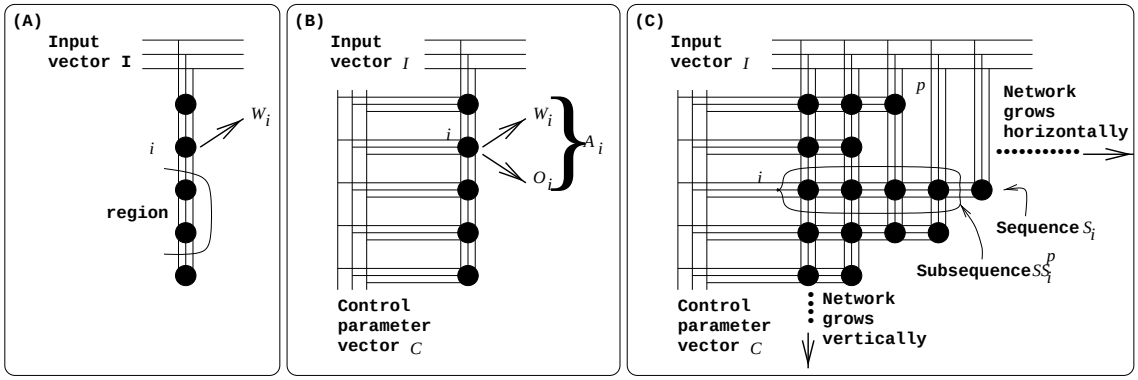


Abbildung 8.3: *(A) Ein eindimentionales selbstorganisierendes Netzwerk. (B) Ein selbstorganisierendes Netzwerk mit Ausgabe. (C) Ein dynamisches Selbstorganisierendes Netzwerk.*

erfüllt, wird ausgewählt. Der Gewichtsvektor dieser Zelle sowie die Gewichtsvektoren aller Zellen innerhalb eines definierten Nachbarschaftsgebiets werden gemäß folgender Gleichung modifiziert:

$$W_k^{new} = W_k^{old} + \eta(I - W_k^{old}). \tag{8.4}$$

η ist dabei ein Parameter, der das Ausmaß der Veränderung festlegt. Gewichtsvektoren außerhalb des spezifizierten Nachbarschaftsgebiets bleiben unverändert. Nach mehreren ,,Epochen", d.h., Präsentationen von Input-Vektoren zum Netzwerk, bilden sich innerhalb des Netzwerks Regionen, die bestimmte Arten von Input-Vektoren charakterisieren. Auf dieser Weise entsteht eine Abbildung, bei der Input-Vektoren mit verschiedenen Merkmalen auf verschiedene Regionen im Netzwerk abgebildet werden.

Erweiterung: Selbstorganisierendes Netzwerk mit Ausgabe

Um das Kontrollproblem zu lösen, ist es, neben der Partitionierung des Input-Raums in Regionen, notwendig, daß passende Kontrollparameter (es wird vorerst vorausgesetzt, daß es eine Methode gibt, mit der diese Parameter bestimmt werden können. Ziel ist es aber, diese Parameter automatisch zu bestimmen. Siehe später.) mit diesen Regionen assoziiert werden, d.h., eine Abbildung zwischen Situationen und Kontrollparametern konstruiert wird. Dies soll inkrementell und simultan zur fortschreitenden

Erfahrung erfolgen. Für diesen Zweck wurde das Netzwerk von Abbildung 8.3(A) um Output-Vektoren erweitert (siehe Abbildung 8.3(B)). Jede Zelle i des Netzwerks hat jetzt zusätzlich zu ihren Gewichtsvektor W_i einen Output-Vektor O_i, der die Kontrollparameter enthält, die für die mit dieser Zelle assoziierten Situation (d.h., die Situation in der diese Zelle Erregungszentrum ist) passend sind. Den Vektor $\begin{pmatrix} W_i \\ O_i \end{pmatrix}$ nennen wir *Assoziation* und bezeichnen ihn mit A_i. Ein Lernschritt erfordert jetzt, daß mit jeder Präsentation eines Input-Vektors I, ein entsprechender Vektor C von passenden Kontrollparametern vorhanden sein muß. Die Adaptation der Output-Vektoren der Zellen geschieht in der gleichen Art und Weise wie mit den Gewichtsvektoren: alle Zellen innerhalb des definierten Nachbarschaftsgebiets des Erregungszentrums (die durch den Input-Vektor selektierte Zelle) ,,verschieben" ihre Output-Vektoren in Richtung auf C. Die Adaptationsregel in Gleichung (8.4) kann jetzt wie folgt geschrieben werden:

$$A_k^{new} = A_k^{old} + \eta(H - A_k^{old}). \tag{8.5}$$

Der neue Vektor $H = \begin{pmatrix} I \\ C \end{pmatrix}$ assoziiert den Input-Vektor mit dem Vektor von Kontrollparametern. Er stellt die Historie der Abbildung von Situationen auf Kontrollparametern dar und hat die gleiche Dimension wie A_i. Die Partitionierung des Input-Raums in Regionen erlaubt jetzt, daß die Abbildung zwischen Situationen und Kontrollparametern auch in Situationen, die vorher nicht getestet wurden, benutzt werden kann. Die oft schwierige und manchmal unmögliche Aufgabe der Assoziierung von passenden Kontrollparametern mit jedem einzelnen Punkt im Input-Raum kann somit vermieden werden (Generalisierungseigenschaft).

Erweiterung: Dynamisches Selbstorganisierendes Netzwerk

Die obige Formulierung des Kontrollproblems wirft zwei wichtige Fragen auf:

i) Was tun, wenn die Best-Matching-Zelle nicht gut genug ist, d.h. sie beschreibt eine Situation, die sich von der aktuellen Situation sehr stark unterscheidet?

ii) Die Vektoren H und A_i wurden nur an einem bestimmten Zeitpunkt betrachtet, d.h. die Historie der Abbildung von Situationen auf Kontrollparameter war nur von

der Länge 1. Wie können Muster von H und A_i entlang der Zeitachse berücksichtigt werden, d.h. wie können konsistente Sequenzen von Mustern gelernt werden?

Diese Fragen können beantwortet werden, indem man das Netzwerk von Abbildung 8.3(B) dynamisch macht (d.h. die Anzahl der Zellen darf mit der Zeit wachsen).

Die erste Frage kann beantwortet werden, indem man zuläßt, daß das Netzwerk vertikal wächst (siehe Abbildung 8.3(C)). Die Erzeugung von neuen Reihen (Assoziationen) im Netzwerk erlaubt dem System, das Clustering des Input-Raums zu verfeinern. Neue Regionen werden kreiert, wenn neue Klassen von Situationen (Cluster) entdeckt werden. Für Klassen von Situationen, die nicht in ausreichender Detaillierung beschrieben sind, werden die entsprechenden Regionen erweitert und vergrößert. Die Entscheidung, wann eine neue Reihe kreiert werden soll, hängt von dem Abstand zwischen dem aktuellen Input-Vektor und der Best-Matching-Zelle (Best-Matching-Reihe) k (siehe Gleichung (8.3)) ab. Wenn dieser Abstand größer ist als ein Schwellwert T_k, dann unterscheidet sich die aktuelle Situation sehr stark von der Klasse von Situationen, die durch die Best-Matching-Zelle k beschrieben ist, und eine neue Reihe (Region/Klasse) muß kreiert werden. Um die Benutzung von festen Schwellwerten zu vermeiden und somit die Anzahl der Parameter, die manuell gesetzt werden sollen, zu reduzieren, wird ein automatischer, adaptiver Schwellwerteinstellungsmechanismus integriert, der auf der Basis von Mittelwertberechnungen arbeitet. Der Schwellwert T_k der Best-Matching-Zelle (Best-Matching-Reihe) k definiert explizit die Grenzen der durch k dargestellten Region. T_k wird ständig verändert bis er zu seinem besten Wert konvergiert. Es wird folgende Update-Regel benutzt:

$$T_k^{new} = T_k^{old} + \beta(\mu_{d_k} + \alpha\sigma_{d_k} - T_k^{old}). \tag{8.6}$$

Dabei ist β ein Lernparameter, α eine Integer-Zahl, und μ_{d_k} und $\sigma_{d_k}^2$ der Mittelwert und die Varianz des Abstands, den man bei frühreren Verwendungen von Zelle (Reihe) k (d.h. Fälle in denen k die Best-Matching-Zelle war) berechnet hat. Anders ausgedrückt, der Schwellwert T_k wird immer in Richtung auf einen Grenzwert verschoben, der gleich dem Mittelwert des Abstands plus einem Vielfachen seiner Standardabweichung ist. Dies bedeutet, daß eine neue Reihe nur dann kreiert wird, wenn der Abstand zur Best-Matching-Reihe größer als gewöhnlich ist.

Die zweite Frage kann beantwortet werden, indem man zuläßt, daß das Netzwerk horizontal wächst (siehe Abbildung 8.3(C)). Erlaubt man den Reihen im Netzwerk in der Länge zu wachsen, dann wird es möglich, längere Sequenzen von Mustern zu erfassen und somit Sequenzen von Situationen zurückzuverfolgen. Eine Reihe i des neuen Netzwerks besteht daher aus mehreren Zellen (Assoziationen). Wir nennen sie *eine Sequenz von Assoziationen* oder einfach *eine Sequenz* und bezeichnen sie mit S_i. Die Assoziation Nummer j ($j = 1, \ldots, l$; l: Anzahl der Assoziationen in S_i) der Sequenz S_i wird mit $S_{i,j}$ bezeichnet. Jede Position p ($p = 1, \ldots, (l-1)$) definiert eine echte (nicht leer und verschieden von S_i) *Teilsequenz* SS_i^p der Sequenz S_i. SS_i^p besteht aus den Assoziationen $\{S_{i,1}, S_{i,2}, \ldots, S_{i,p}\}$. Der Selektionsprozeß (Bestimmung der Best-Matching-Reihe) besteht jetzt aus der Bestimmung der Teilsequenz (aus der Menge aller möglichen Teilsequenzen der Sequenzen), die mit der bisherigen Input-Historie (zumindest die Historie neueren Datums) am ähnlichsten ist. Die Input-Historie wird mit Hilfe der Vektorliste I_t (Erweiterung von I entlang der Zeitachse) codiert. $t = 1, 2, \ldots$ bezeichnet einen diskreten Zeitindex wobei 1 den aktuellen Zeitpunkt und Werte größer als 1 die unmittelbare Vergangenheit darstellen. Die Vektorlisten C_t und H_t sind die entsprechenden Erweiterung für C und H. Die Best-Matching-Teilsequenz SS_k^q ist diejenige, die folgende Bedingung erfüllt:

$$\begin{array}{ccccc} d(SS_k^q, I_t) & \leq & d(SS_i^p, I_t) & & \\ \text{für alle Sequenzen} \quad S_i & \text{und alle} & 1 \leq p < length(S_i). & & (8.7) \end{array}$$

Dabei ist $d(.,.)$ die Erweiterung der alten Abstandsfunktion. $d(.,.)$ wird benutzt, um die Übereinstimmung zwischen der Input-Historie und einer Teilsequenz des neuronalen Netzwerks zu messen. Sie ist wie folgt definiert:

$$d(SS_i^p, I_t) \quad = \quad \frac{1}{p} \sum_j^p \| I_{(p+1-j)} - W_{i,j} \|. \qquad (8.8)$$

$W_{i,j}$ bezeichnet dabei den Gewichtsvektor der Assoziation Nummer j der Teilsequenz SS_i^p (siehe Abbildung 8.4(A) für eine graphische Darstellung). Gleichung (8.7) stellt einen Prozeß dar, bei dem alle Teilsequenzen mit der Input-Historie verglichen werden und diejenige Teilsequenz selektiert wird, die den minimalen mittleren Abstand pro Zeiteinheit liefert. Gleichung (8.7) ersetzt den alten Prozeß für die Bestimmung des Erregungszentrums (siehe Gleichung (8.3)).

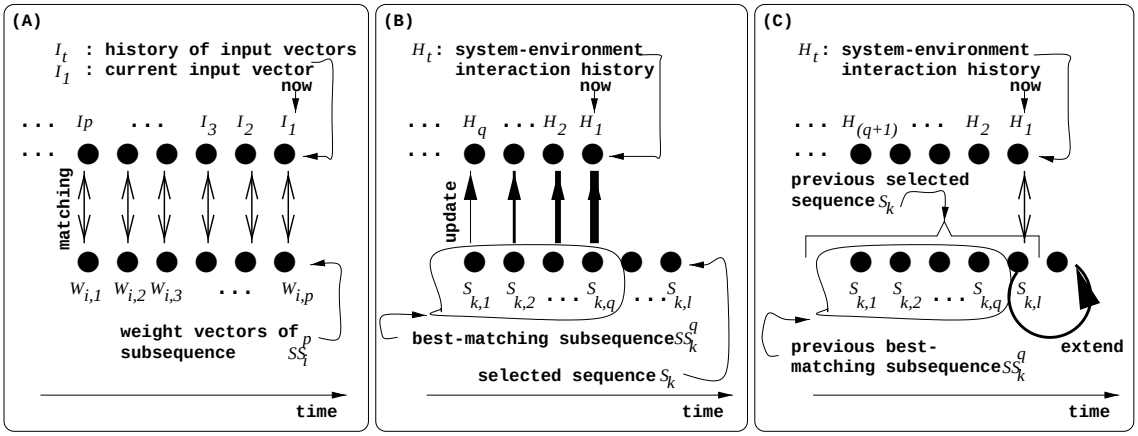


Abbildung 8.4: *(A) Vergleich der Teilsequenz SS_i^p mit der Input-Historie. Das Ergebnis ist der mittlere Abstand pro Zeiteinheit. (B) Anpassung der selektierten Sequenz S_k. Nur die Assoziationen der Best-Matching-Teilsequenz SS_k^q werden verändert. Sie werden in Richtung auf die Interaktionshistorie verschoben. Die Verschiebung ist für Assoziationen am Ende der Teilsequenz stärker als für Assoziationen am Anfang der Teilsequenz (angedeutet durch unterschiedlich dicke Pfeile). (C) Erweiterung der im vorigen Zeitschritt selektierten Sequenz S_k. Wenn im vorigen Zeitschritt die Best-Matching-Teilsequenz SS_k^q von S_k die längste echte Teilsequenz war und S_k die aktuelle Situation gut vorhersagen konnte, d.h. der Abstand zwischen dem Gewichtsvektor ihrer letzten Assoziation und dem aktuellen Input-Vektor ist kleiner als ein bestimmter Schwellwert, dann wird die Sequenz S_k durch Duplizierung ihrer letzten Assoziation erweitert.*

Die Update-Regel für die Sequenz S_k, die die Best-Matching-Teilsequenz SS_k^q enthält, muß auch so geändert werden, daß sie mehrere Muster entlang der Zeitachse berücksichtigen kann. Gleichung (8.9) zeigt die neue Version dieser Regel:

$$S_{k,j}^{new} = S_{k,j}^{old} + \frac{\eta}{(q+1-j)} \left(H_{(q+1-j)} - S_{k,j}^{old} \right) \quad \text{für} \quad j = 1, \ldots, q \tag{8.9}$$

Gleichung (8.9) stellt einen Prozeß dar, bei dem jede Assoziation $S_{k,j}$ der Best-Matching-Teilsequenz SS_k^q in Richtung auf das entsprechende Muster $H_{(p+1-j)}$ verschoben wird. Die Verschiebung erfolgt mit unterschiedlichen Stärken. Die Assoziation, die dem ak-

tuellen Zeitschritt entspricht, wird stärker verändert als diejenigen, die vorhergehenden Zeitschritten entsprechen. Der Koeffizient für die Veränderung der Assoziationen ist im aktuellen Zeitschritt ($t = 1$) gleich η. Wenn der Zeitindex t von $t = 1$ nach $t = q$ geht, nimmt dieser Koeffizient linear ab bis er $\frac{\eta}{q}$ erreicht (siehe Abbildung 8.4(B)). Dies ist notwendig, weil die ersten Assoziationen in einer Teilsequenz häufiger als die letzten Assoziationen benutzt werden und somit auch häufiger verändert werden.

Die Entscheidung, wann die Sequenz, die die Best-Matching-Teilsequenz enthält, erweitert (ihre Länge vergrößert) werden soll, hängt davon ab, wie gut die Sequenz die nächste Situation vorhersagen kann und wie lang die Best-Matching-Teilsequenz ist. Wenn im vorigen Zeitschritt der Unterschied zwischen der Länge der Best-Matching-Teilsequenz und der Länge ihrer Sequenz gleich 1 war (d.h. die Best-Matching-Teilsequenz war die längste echte Teilsequenz, die gebildet werden kann) und die Sequenz eine gute Vorhersage für die aktuelle Situation bietet (d.h. der Abstand zwischen dem Gewichtsvektor der letzten Assoziation der Sequenz und dem aktuellen Input-Vektor ist kleiner als ein bestimmter Schwellwert), dann wird die Sequenz erweitert (länger gemacht), indem ihre letzte Assoziation dupliziert wird (siehe Abbildung 8.4(C)). Auf dieser Weise wird diese Sequenz, wenn sie das nächste Mal in ähnlichen Situationen selektiert wird, länger sein, und somit in der Lage, eine längere Situationsfolge vorherzusagen. Der Schwellwert für die Entscheidung über die Güte der Vorhersage kann wiederum mit Hilfe des oben beschriebenen adaptiven Schwellwerteinstellungsmechanismus (siehe Gleichung (8.6)) automatisch bestimmt werden.

Zusammenfassung

Wir haben ein Kohonen Netzwerk erweitert, indem wir es dynamisch machten (variable Anzahl von Zellen) und indem wir mit jeder Zelle einen Output-Vektor assoziierten. Der erhaltene Netzwerkalgorithmus ist in der Lage, über Sequenzen von Situationen zu generalisieren, d.h. er ist in der Lage, mit Hilfe der Interaktionshistorie (Historie der Interaktion zwischen System und Umgebung) zu lernen, den Input-Raum (alle möglichen Sequenzen von Situationen) inkrementell in mehrere Regionen zu partitionieren. Selbst wenn die Wahrscheinlichkeitsverteilung der Input-Parameter sich ändert, ist der Algorithmus in der Lage, diese Partionierung (Kartenbildung) selbständig zu reorganisieren,

indem er existierende Regionen verfeinert oder neue Regionen kreiert. Der Gesamtalgorithmus kann wie folgt zusammengefaßt werden:

Wiederhole in jedem Zeitschritt:

- Erweitere die Interaktionshistorie durch die aktuellen Input-Parameter und die entsprechenden Kontrollparameter.
- **If** die im letzten Zeitschritt ausgewählte Sequenz (d.h. die Sequenz, die die Best-Matching-Teilsequenz enthält) konnte eine gute Vorhersage für die aktuellen Input-Parameter liefern und ihre Best-Matching-Teilsequenz ist die längste echte Teilsequenz, die gebildet werden kann
  then erweitere diese Sequenz.
- Wähle die Sequenz, die im aktuellen Zeitschritt die Best-Matching-Teilsequenz enthält.
- **If** der Abstand zwischen der Best-Matching-Teilsequenz und der aktuellen Interaktionshistorie ist kleiner als der Entscheidungsschwellwert
  then aktualisiere die entsprechende Sequenz (verschiebe die Sequenz in Richtung auf Interaktionshistorie)
  else kreiere eine neue Sequenz.

8.3 Integration von Reinforcement Lernen

Bis jetzt wurde vorausgesetzt, daß es eine Komponente gibt, die in der Lage ist, zumindest für eine repräsentative Menge von Situationen (Trainingset), die passenden Kontrollparameter (Koordinationsparameter) zu liefern. Die Aufgabe des Lernprozesses bestand darin, eine Karte zu bilden, die Generalisierung ermöglicht. Das System bekommt dadurch die Fähigkeit, sich in jeder beliebigen Situation an die passenden Kontrollparameter erinnern zu können. Das System wird außerdem in die Lage versetzt, die Zukunft (z.B. wie die nächste Situation aussehen könnte) mit Hilfe der Sequenzen von Assoziationen, die es in der Vergangenheit bilden konnte, vorhersagen zu können.

Was ist aber, wenn nicht genügend Wissen vorhanden ist, um eine Komponente, die die Kontrollparameter liefert, zu modellieren? Wie kann das System ohne Überwachung

arbeiten, d.h. wie kann das System lernen, die Kontrollparameter selbständig zu berechnen?

Reinforcement Lernen stellt einen geeigneten Rahmen für die Lösung dieses Problems dar. Wie schon in den vorhergehenden Kapiteln erwähnt wurde, zielt Reinforcement Lernen darauf ab, ein System mit Hilfe von Rewards zu adaptieren. Grundsätzlich bekommt der Agent, wenn er eine Aktion in einem bestimmten Zustand ausführt, ein Reward-Signal. Wenn das Reward-Signal positiv ist, dann wird die Assoziation zwischen der Aktion und dem Zustand verstärkt, um die Wahrscheinlichkeit, daß in ähnlichen Zuständen in der Zukunft die gleiche Aktion ausgeführt wird, zu erhöhen. Wenn im Gegensatz das Reward-Signal negativ ist, dann wird diese Assoziation abgeschwächt, um die Wahrscheinlichkeit, daß der gleiche Fehler in der Zukunft wieder gemacht wird, zu verringern. Im folgenden wird gezeigt, wie durch die Kombination des oben beschriebenen Netzwerkalgorithmus mit Reinforcement Lernen die Kontrollparameter automatisch bestimmt werden können.

Um Kontrollparameter, die nützlich sind, erlernen zu können, soll die Update-Prozedur (siehe Gleichung 8.9) die ausgewählte Sequenz nicht wie bisher blind aktualisieren (d.h. immer versuchen, diese Sequenz mit der aktuellen Interaktionshistorie ähnlicher zu machen), sondern versuchen, das Reward-Feedback so zu berücksichtigen, daß nur Kontrollparameter, die gute Rewards produzieren, in die Sequenz übernommen werden. Dies kann erreicht werden, indem man den nach der Anwendung der ausgewählten Sequenz (eine Sequenz wird angewendet, indem die Kontrollparameter, die diese Sequenz zum Umgang mit der aktuellen Situation empfiehlt, zur Steuerung des Agenten benutzt werden. Siehe Abbildung 8.5(A)) erhaltenen Reward mit einem Reward-Schwellwert vergleicht und die Update-Prozedur nur dann ausführt, wenn der Reward größer als der Schwellwert ist. Für eine Sequenz S_k wird der entsprechende Reward-Schwellwert RT_k mit Hilfe der Rewards, die nach Anwendungen von S_k in der Vergangenheit erzeugt wurden, ermittelt (Mittelwertbildung). Auf diese Weise wird eine Sequenz nur dann in Richtung auf die aktuelle Interaktionshistorie verschoben, wenn die Interaktionshistorie bessere Assoziationen als diejenigen, die in der Sequenz codiert sind, enthält. Zur automatischen Aktualisierung des Reward-Schwellwerts RT_k wird wiederum der oben

beschriebene Schwellwerteinstellungsmechanismus benutzt:

$$RT_k^{new} = RT_k^{old} + \epsilon(\mu_{R_k} + \gamma\sigma_{R_k} - RT_k^{old}). \tag{8.10}$$

Dabei ist ϵ eine Lernkonstante und γ eine Integer-Konstante. Mit γ kann man die Höhe des Reward-Schwellwerts variieren und somit die Wahrscheinlichkeit einer Update-Operation beeinflussen (erhöhen oder verringern). μ_{R_k} und $\sigma^2_{R_k}$ sind der Mittelwert und die Varianz des Rewards R_k, der bei früheren Anwendungen der Sequenz S_k erzeugt wurde.

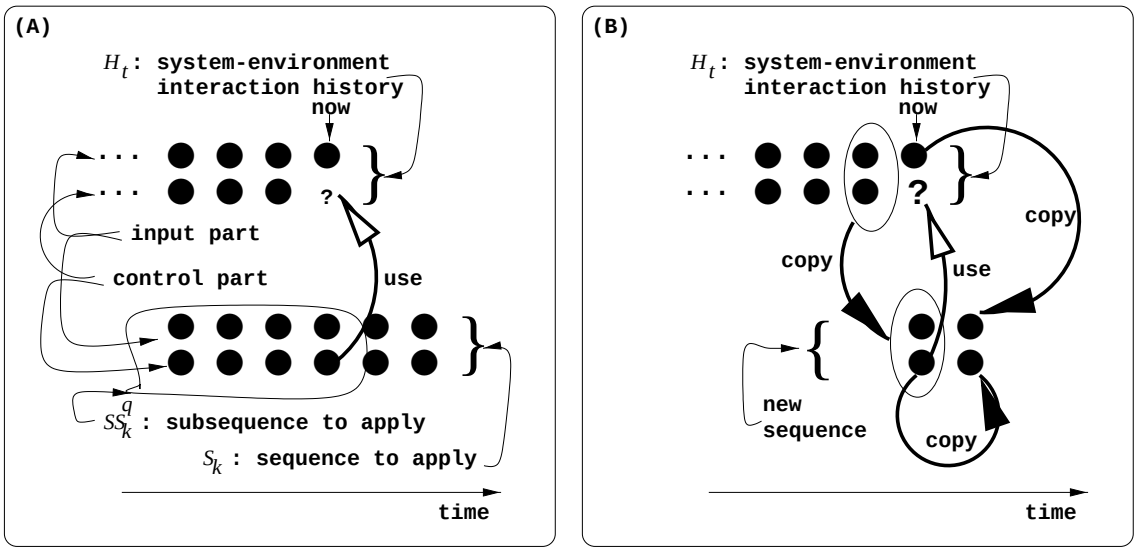


Abbildung 8.5: *(A) Anwendung einer ausgewählten Sequenz. Eine ausgewählte Sequenz wird angewendet, indem die Kontrollparameter, die in der letzten Assoziation der entsprechenden Best-Matching-Teilsequenz enthalten sind, zur Steuerung des Agenten in der aktuellen Situation benutzt werden. (B) Erzeugung und Anwendung einer neuen Sequenz.*

Da die Bildung von Sequenzen, die unnütze Kontrollparameter (d.h. Kontrollparameter, die schlechte Rewards verursachen) enthalten, unerwünscht ist, muß die Entscheidungsregel für die Erweiterung einer Sequenz auch eingeschränkt werden. Ähnlich wie bei der Update-Prozedur wird eine Sequenz nur dann erweitert, wenn ihre Anwendung zu einem Reward führt, der größer als der Reward-Schwellwert ist.

Wenn, auf der anderen Seite, der erhaltene Reward nicht gut genug ist, d.h., kleiner als der Reward-Schwellwert ist, dann muß das System den Lösungsraum explorieren (erforschen) und mit anderen Kontrollparametern experimentieren. Wenn kein zusätzliches Wissen über die Richtung, in der die Kontrollparameter verändert werden sollen, existiert, dann können während der Exploration zufällige Werte benutzt werden.

Der Gesamtalgorithmus zum Erlernen von Kontrollparametern (Koordinationsparametern) kann jetzt wie folgt zusammengefaßt werden:

Wiederhole in jedem Zeitschritt:

- **If** der erhaltene Reward ist größer als der Reward-Schwellwert der im vorigen Zeitschritt angewandten Sequenz
  then
  - verschiebe die im vorigen Zeitschritt angewandte Sequenz in Richtung auf die Interaktionshistorie (d.h., führe die Update-Prozedur aus).
  - **if** die im vorigen Zeitschritt angewandte Sequenz konnte die aktuelle Situation gut vorhersagen und sie ist nicht lang genug (d.h., die Länge dieser Sequenz und die Länge der entsprechenden Best-Matching-Teilsequenz unterscheiden sich nur um 1)
    then erweitere diese Sequenz (d.h., verlängere sie).

  else
  - exploriere den Lösungsraum, um einen neuen Vektor von Kontrollparametern zu bestimmen.
  - aktualisiere die im vorigen Zeitschritt angewandte Sequenz mit Hilfe dieses neuen Vektors (d.h., ersetze den Kontrollteil der letzten Assoziation der angewandten Teilsequenz durch diesen neuen Vektor).
- erweitere die Interaktionshistorie durch den aktuellen Vektor von Input-Parametern (die entsprechenden Kontrollparameter werden später bestimmt).
- wähle die Sequenz, die im aktuellen Zeitschritt die Best-Matching-Teilsequenz enthält.
- **If** der Abstand zwischen der Best-Matching-Teilsequenz und der aktuellen Interaktionshistorie ist kleiner als der Entscheidungsschwellwert

then wende die gewählte Sequenz an (siehe Abbildung 8.5(A)).
else erzeuge eine neue Sequenz und wende sie an (siehe Abbildung 8.5(B)).

- erweitere die Interaktionshistorie durch den entsprechenden Vektor von Kontrollparametern.

Der obige Lernalgorithmus integriert Reinforcement Lernen in die Prozedur zur Partitionierung des Input-Raums in Regionen. Die Aufgabe des Lernprozesses besteht darin, Sequenzen zu erzeugen und diese so zu verändern, daß sie zu stabilen Assoziationen zwischen Input- und Kontrollparametern konvergieren. Stabile Assoziationen stellen Regelmäßigkeiten dar, die das System aufgrund seiner Erfahrung identifizieren kann. Sie bilden die Grundlage für vernünftiges Agieren in zukünftigen Situationen.

8.4 Simulationsergebnisse

In diesem Abschnitt wird der obige Lernalgorithmus mit Hilfe der simulierten, dynamischen Umwelt von Abschnitt 7.5 getestet. Es soll das komplexe Ziel ,,*Navigation*" durch die Koordination der weniger komplexen Ziele ,,*Survival*" und ,,*Goal-Directedness*" gelöst werden (siehe Abbildung 8.6).

,,Navigation" ist definiert als die Aufgabe des Findens eines Pfades, entlang dem der Agent sich von einem Startpunkt zu einem Zielpunkt sicher bewegen kann (siehe Abbildung 8.7). Ein Spiel endet, wenn der Agent den Zielpunkt erreicht hat oder wenn er stirbt. Der Agent stirbt, wenn er mit einem Feind oder mit einem Hindernis kollidiert oder wenn er keine Energie mehr hat.

,,Goal-Directedness" soll dafür sorgen, daß der Agent sich in der dynamischen Umgebung in Richtung des Zielpunkts bewegt. Für diesen Zweck wurde der Agent mit dem zusätzlichen Sensor S_G ausgerüstet. Der Sensor S_G liefert in jedem Zeitschritt den Winkel zwischen der Vorwärtsrichtung und der Linie, die die aktuelle Position des Agenten mit dem Zielpunkt verbindet. Der Winkel ist ein Wert in $[-\pi, \pi]$. Er ist positiv wenn der Zielpunkt sich auf der linken Seite des Agenten befindet und negativ andernfalls. Ein Entwurf für das Ziel ,,Goal-Directedness" im Sinne der Architektur von Kapitel 5 ist in Abbildung 8.8 dargestellt. Die Prioritäten der Basis-Verhaltensmuster bezüglich des

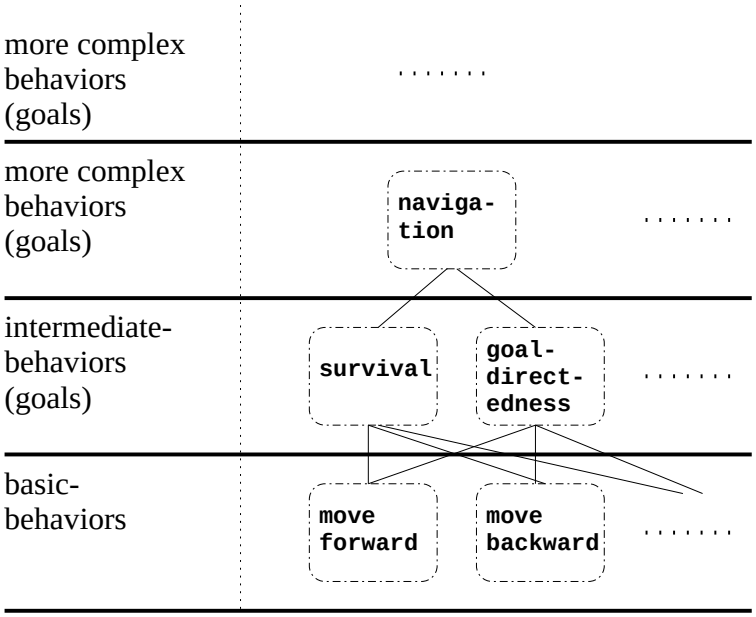


Abbildung 8.6: *Lösung der komplexen Aufgabe (Ziel) ,,Navigation".*

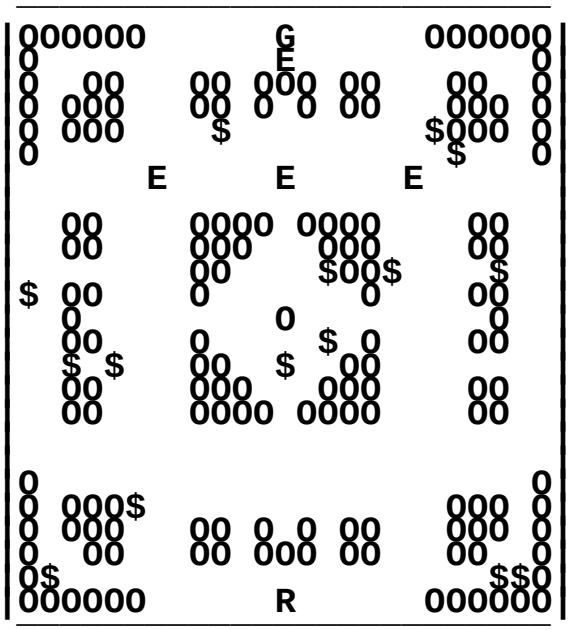


Abbildung 8.7: *,,Navigation": Einen Pfad für die sichere Bewegung von einem Startpunkt zum einem Zielpunkt (G) finden.*

Ziels ,,Goal-Directedness" werden in jedem Zeitschritt gemäß der Formel in Gleichung 5.1 berechnet.

Behaviors \ Sensors	S_F	S_B	S_L	S_R	S_G
move forward	g_0	g_0	g_0	g_0	g_1
move backward	g_0	g_0	g_0	g_0	g_2
move left	g_0	g_0	g_0	g_0	g_3
move right	g_0	g_0	g_0	g_0	g_4

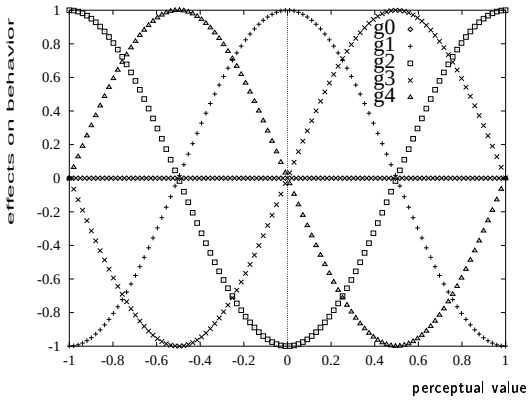


Abbildung 8.8: *Entwurf für das Ziel ,,Goal-Directedness": Abhängigkeit zwischen den Basis-Verhaltensmustern und den Sensoren modelliert durch die Funktionen* $g_0(x) = 0$, $g_1(x) = cos(\pi x)$, $g_2(x) = -cos(\pi x)$, $g_3(x) = sin(\pi x)$, *und* $g_4(x) = -sin(\pi x)$.

,,Survival" bedeutet wie bisher, daß der Agent ständig versuchen soll, Nahrung zu bekommen, damit er Energie sammelt und sich somit bewegen kann. Er muß auch gleichzeitig Hindernisse vermeiden und sich vor den angreifenden Feinden retten. Ein Entwurf für das Ziel ,,Survival" im Sinne der Architektur von Kapitel 5 ist in Abbildung 8.9 dargestellt. Die Prioritäten der Basis-Verhaltensmuster bezüglich des Ziels ,,Survival" werden in jedem Zeitschritt gemäß der Formel in Gleichung 5.1 berechnet.

Um das Ziel ,,Navigation" zu lösen, muß das *lernende System* in jedem Zeitschritt den Koordinationsvektor $\Omega(t) = \begin{pmatrix} \Omega_1(t) \\ \Omega_2(t) \end{pmatrix}$ berechnen. Im folgenden wird das lernende System mit einem *statischen System* und mit einem *Random-System* verglichen. Das statische System benutzt den konstanten Koordinationspametervektor $\Omega(t) = \begin{pmatrix} \Omega_1(t) \\ \Omega_2(t) \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$. Diese Parameter wurden von Hand ermittelt und liefern recht gute Ergebnisse. Die Leistung des statischen Systems kann man daher als Zielleistung ansehen, die vom lernenden System erreicht werden soll. Das Random-System benutzt in jedem Zeitschritt zufällige Koordinationsparameter und gilt als initiale Konfiguration für das

Sensors / Behaviors	S_F	S_B	S_L	S_R	S_G
move forward	f_2	f_1	f_1	f_1	f_0
move backward	f_1	f_2	f_1	f_1	f_0
move left	f_1	f_1	f_2	f_1	f_0
move right	f_1	f_1	f_1	f_2	f_0

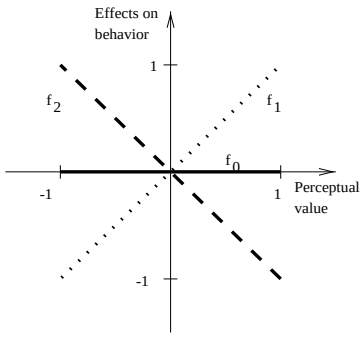


Abbildung 8.9: *Entwurf für das Ziel „Survival": Abhängigkeit zwischen den Basis-Verhaltensmustern und den Sensoren modelliert durch die Funktionen* $f_0(x) = 0$, $f_1(x) = x$ *und* $f_2(x) = -x$.

lernende System (am Anfang, wenn noch nicht genügend Erfahrung gesammelt wurde, werden auch beim lernenden System die Koordinationsparameter zufällig gewählt). Im folgenden wird auch gezeigt, wie die Wahl bestimmter Parameter des Lernalgorithmus sich auf die Leistung des Agenten auswirkt. Zur Evaluierung der Navigationsleistung der verschiedenen Systeme, werden folgen Evaluierungsgrößen benutzt:

- Erfolgswahrscheinlichkeit (*success probability*): Die Wahrscheinlichkeit, daß ein Spiel mit Erfolg abgeschlossen wird, d.h., der Agent einen Pfad zum Zielpunkt findet (den Zielpunkt erreicht bevor er stirbt). Größere Werte zeigen eine bessere Leistung an.

- Durchschnittswert von $\frac{Anzahl\ der\ Schritte}{Startabstand}$ pro nicht erfolgreiches Spiel (*average* $\frac{number\ of\ steps}{start\ distance}$ *per unsuccessful play*): Mittlerer Wert des Quotienten aus der Anzahl der durch den Agenten ausgeführten Schritte (Dauer des Spiels) und dem Abstand zwischen Startpunkt und Zielpunkt berechnet über alle nicht erfolgreichen Spiele. Größere Werte zeigen eine bessere Leistung an.

- Durchschnittswert von $\frac{Endabstand}{Startabstand}$ pro Spiel (*average* $\frac{final\ distance}{start\ distance}$ *per play*): Mittlerer Wert des Quotienten aus dem verbleibenden Abstand zum Zielpunkt am Ende eines Spiels und dem Abstand zwischen Startpunkt und Zielpunkt berechnet über alle Spiele. Kleinere Werte zeigen eine bessere Leistung an.

Leistung des lernenden Systems

In einer ersten Studie wurde die Leistung des lernenden Systems durch Vergleich mit dem statischen und dem Random-System evaluiert. Das vom lernenden System benutzte dynamische, selbstorganisierende Netzwerk durfte bis zu 50 Sequenzen bilden. Jede Sequenz durfte maximal 50 Zellen enthalten. Das Reward-Signal R, welches das Ziel ,,Navigation" definiert, bestand aus der Kombination der beiden Signale R_1 und R_2:

$$R = \rho R_1 + (1-\rho)R_2 \tag{8.11}$$

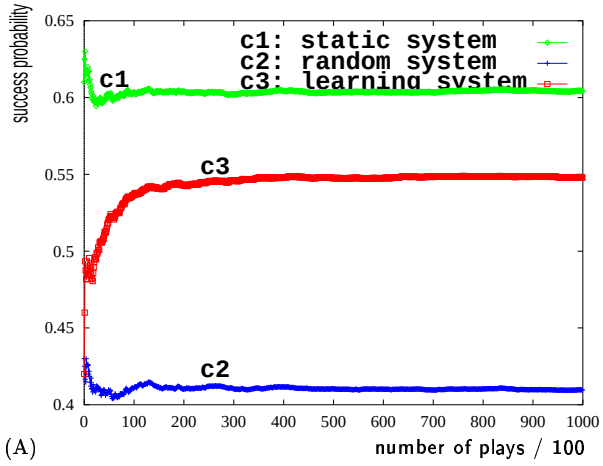
mit

$$R_1 = \begin{cases} -5 & \text{wenn Agent stirbt} \\ 0 & \text{sonst} \end{cases} \quad \text{und} \quad R_2 = \begin{cases} 5 & \text{wenn Zielpunkt erreicht} \\ 0 & \text{sonst} \end{cases}$$

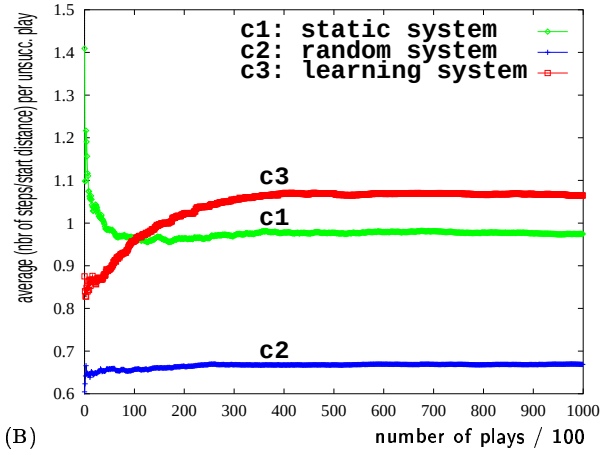
ρ ist ein Parameter, der dazu benutzt werden kann, um den relativen Einfluß von R_1 und R_2 auf R verändern zu können (in diesem Experiment wurde $\rho = 0.5$ gewählt). Die anderen Parameter des Lernalgorithmus wurden wie folgt gewählt: α (benutzt in Gleichung (8.6)): 2, β (benutzt in Gleichung (8.6)): 0.9, ϵ (benutzt in Gleichung (8.10)): 0.9, γ (benutzt in Gleichung (8.10)): 0, η (benutzt in Gleichung (8.9)): Fallend von 1 auf 0.000001. Abbildung 8.10 zeigt wie das lernende System seine Leistung bezüglich aller drei Evaluierungsgrößen inkrementell verbessert. Trotz der deutlich großen Leistungsverbesserung im Vergleich zum Random-System (initiale Konfiguration des lernenden Systems) konnte das lernende System in diesem Fall keine bessere Erfolgswahrscheinlichkeit als das statische System erreichen. Dies liegt unter anderem daran, daß das neuronale Netzwerk nur eine kleine Menge von Sequenzen bilden durfte. In einem zweiten Experiment wurde der Input-Raum eingeschränkt, indem feste Positionen für den Startpunkt und für den Zielpunkt gewählt wurden (die gleichen Positionen wie in Abbildung 8.7). Das lernende System konnte in diesem Fall sogar eine bessere Erfolgswahrscheinlichkeit als das statische System erreichen (siehe Abbildung 8.11).

Wirkung des Reward-Signals

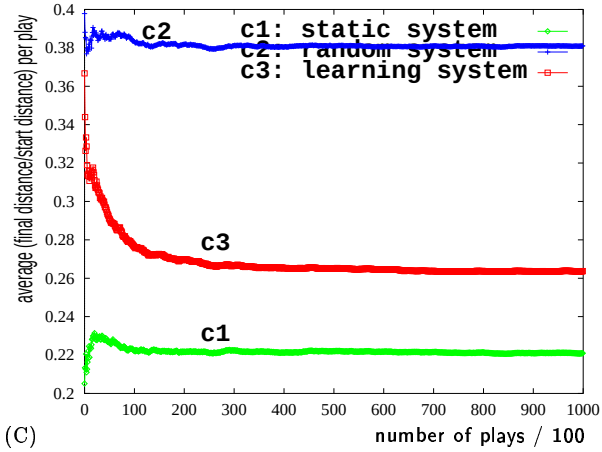
Das Reward-Signal definiert die Aufgabe, die das System erfüllen soll (hier die Lösung des Ziels ,,Navigation") und muß daher sorgfältig gewählt werden. Ungeeignete Reward-Signale können dazu führen, daß der Lernalgorithmus überhaupt nicht konvergiert oder



(A)



(B)



(C)

Abbildung 8.10: *Vergleich der Leistung des lernenden Systems mit der eines statischen Systems und mit der eines Random-Systems: c1: statisches System; c2: Random-System; c3: lernendes System.*

daß er nur unzufriedenstellende Ergebnisse liefert. Abbildung 8.12 zeigt die Leistung des lernenden Systems bei verschiedenen Reward-Signalen, die durch Variierung des Parameters ρ in Gleichung 8.11 erzeugt wurden. Wenn $\rho = 0\%$ (d.h. Sterben wird nicht im Reward-Signal berücksichtigt), dann verhält sich das lernende System genau wie das Random-System und der Lernalgorithmus konvergiert gar nicht. Der Grund dafür ist, daß das System nicht weiß, daß Sterben schlecht ist, und daher nicht versuchen wird, es während des Lernens zu vermeiden. Die Lernaufgabe wird dadurch sehr erschwert. In diesem Experiment konnte der Lernalgorithmus nur sehr kurze Sequenzen bilden. Die Leistung des Systems erhöht sich mit steigendem ρ. Wenn $\rho = 100\%$ (d.h. nur Sterben beeinflußt das Reward-Signal), dann steigt die Erfolgswahrscheinlichkeit zwar am Anfang sehr schnell, wird aber nach mehreren Spielen etwas schlechter als die

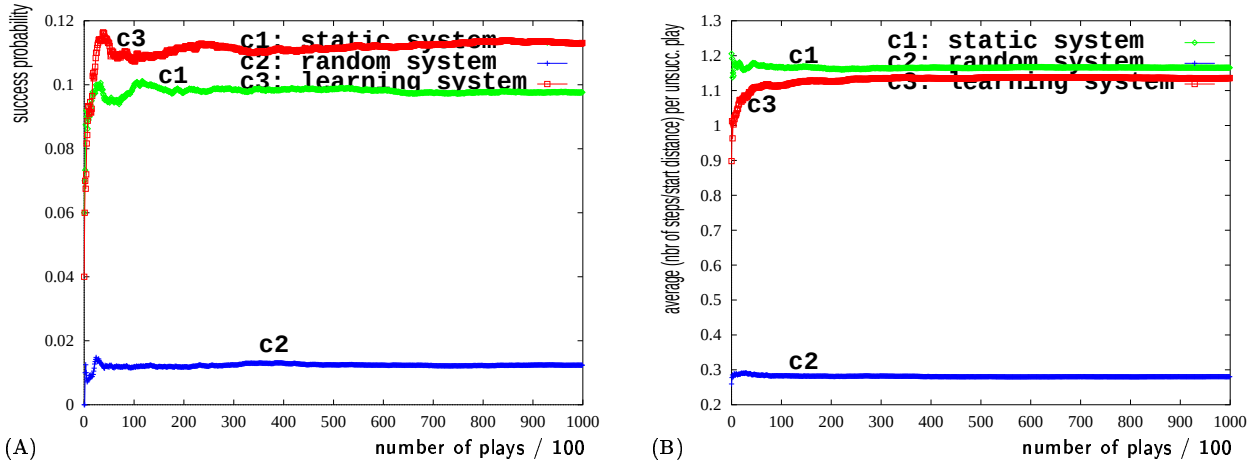


Abbildung 8.11: *Vergleich der Leistung des lernenden Systems mit der eines statischen Systems und mit der eines Random-Systems: c1: statisches System; c2: Random-System; c3: lernendes System.*

Erfolgswahrscheinlichkeit, die mit $\rho = 60\%$ erreicht werden konnte und die in diesem Experiment die beste war. Dies bedeutet, daß für die Wahl des Reward-Signals, nicht nur das Sterben, sondern auch das Erreichen der Zielposition wichtig ist. Die Parametereinstellungen für den Lernalgorithmus waren die gleichen wie im vorigen Experiment. Der Startpunkt und der Zielpunkt wurden am Anfang jedes Spiels zufällig gewählt.

Wirkung des Parameters η

In einem weiteren Experiment wurde die Wirkung des Parameters η (update gain) von Gleichung (8.9) auf die Leistungsfähigkeit des Lernalgorithmus untersucht. Das lernende System wurde mit den Werten $\eta = 0.9$, $\eta = 0.5$, $\eta = 0.1$, und η: Fallend von 1

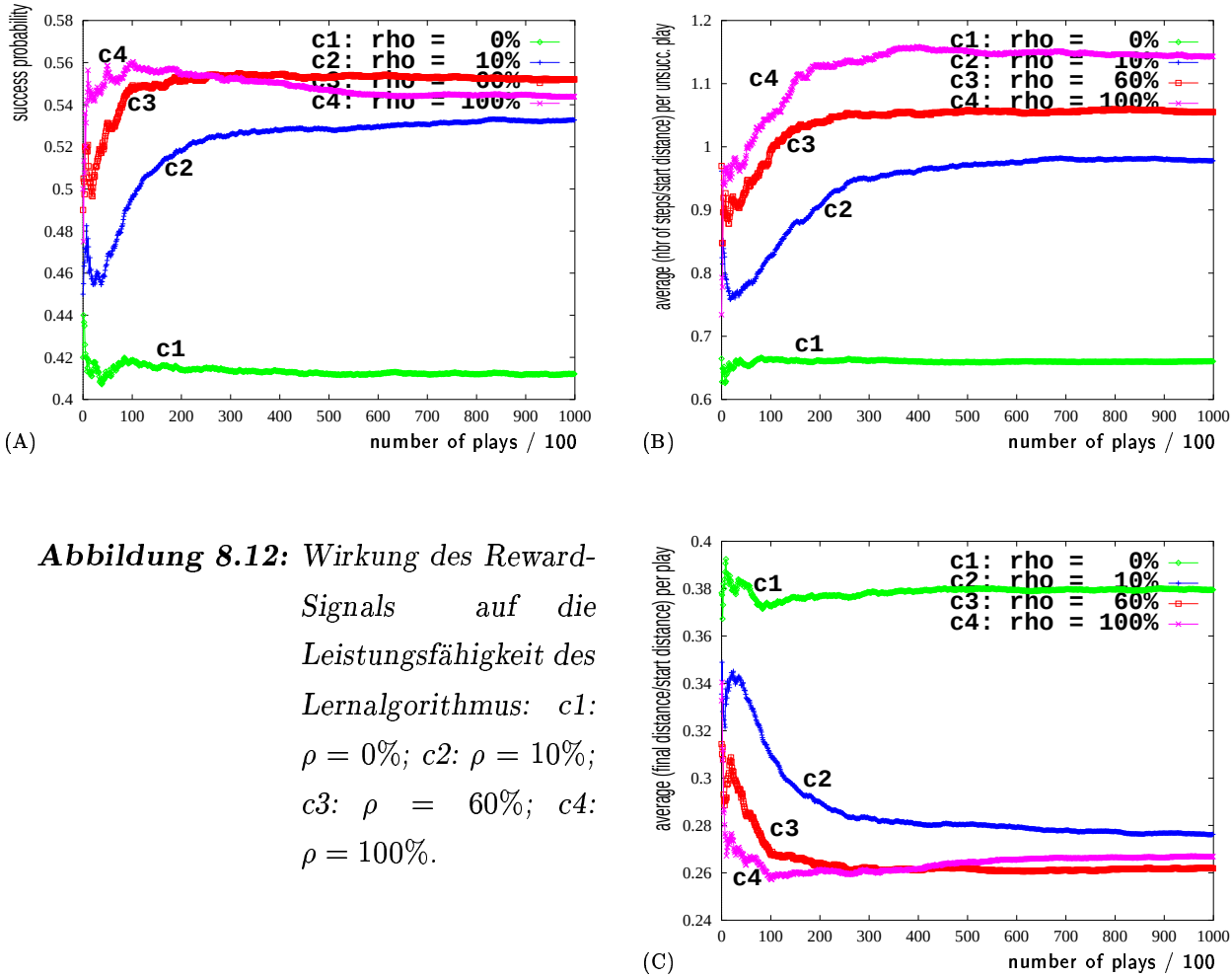
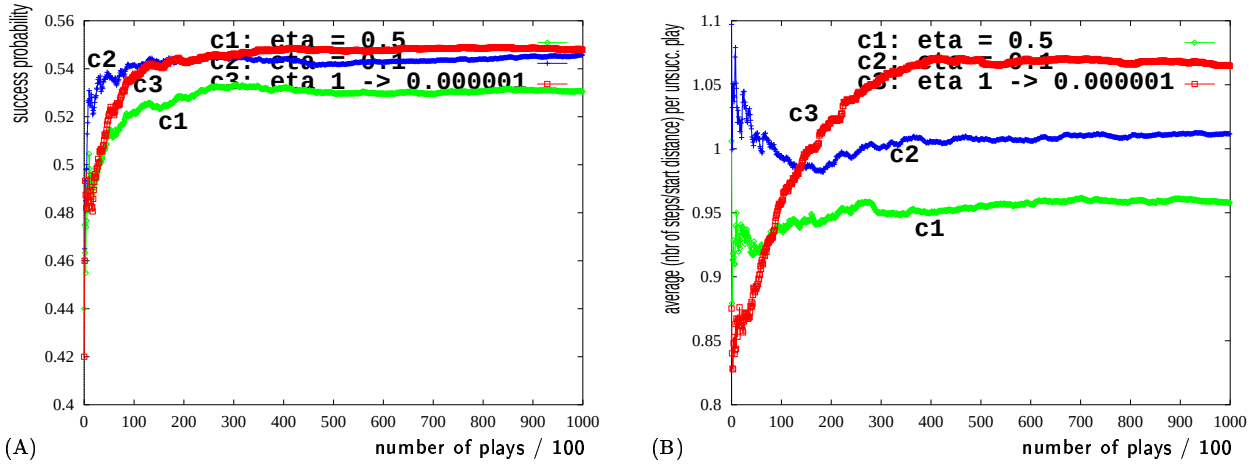


Abbildung 8.12: *Wirkung des Reward-Signals auf die Leistungsfähigkeit des Lernalgorithmus: c1:* $\rho = 0\%$*; c2:* $\rho = 10\%$*; c3:* $\rho = 60\%$*; c4:* $\rho = 100\%$.

auf 0.000001 getestet. Dieses Experiment hat gezeigt, daß je kleiner η ist, desto besser sind die Ergebnisse. Die besten Ergebnissen wurden allerdings mit einem mit der Zeit fallenden η erreicht (siehe Abbildung 8.13). Die anderen Parametereinstellungen für den Lernalgorithmus waren die gleichen wie im vorigen Experiment. Der Startpunkt und der Zielpunkt wurden am Anfang jedes Spiels zufällig gewählt.

Wirkung des Parameters γ

In einem anderen Experiment wurde die Wirkung des Parameters γ von Gleichung (8.10) auf die Leistungsfähigkeit des Lernalgorithmus untersucht. Das lernende System wurde mit den Werten $\gamma = -7$, $\gamma = -2$, $\gamma = 0$, und $\gamma = 1$ getestet. Es ist zu erwarten, daß zu



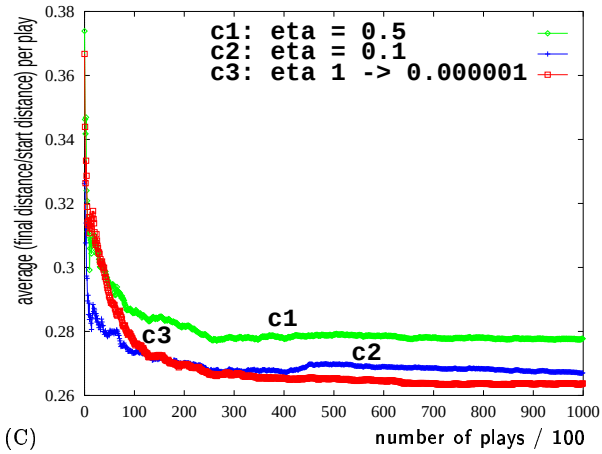
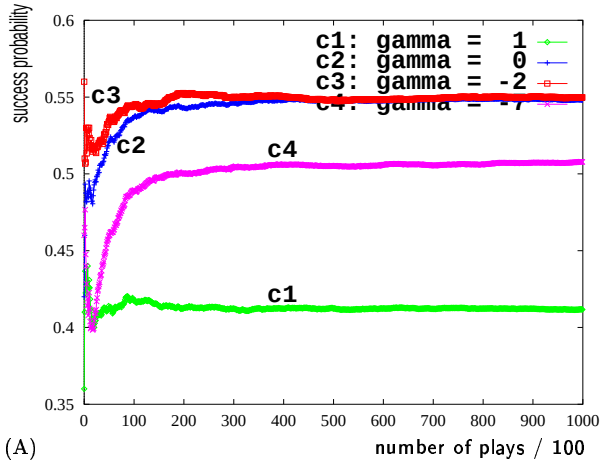
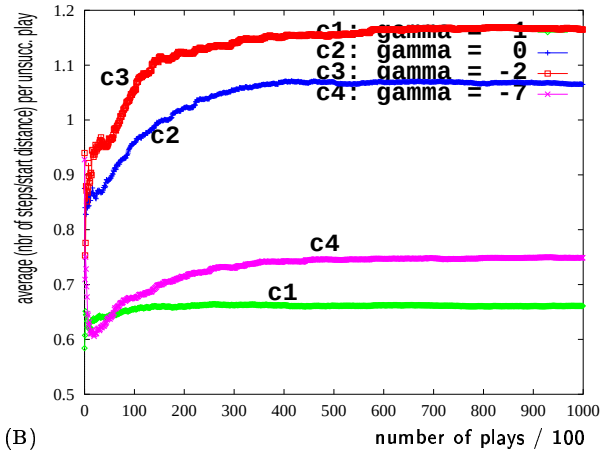


Abbildung 8.13: *Wirkung des Parameters η von Gleichung (8.9) auf die Leistungsfähigkeit des Lernalgorithmus: c1: $\eta = 0.5$; c2: $\eta = 0.1$; c3: η fallend von 1 auf* 0.000001.

große Werte von γ schlechte Ergebnisse liefern, weil sie zu hohen Reward-Schwellwerten führen und dies seinerseits die Anzahl der durchgeführten Update-Operationen reduziert und folglich die Bildung nützlicher Sequenzen verhindert. Genau dies ist auch eingetreten. Mit $\gamma = 1$ hat der Lernalgorithmus überhaupt nicht konvergiert. Seine Leistung war vergleichbar mit der des Random-Systems (siehe Abbildung 8.14). Wenn andererseits γ zu klein ist, dann wird jedes Reward-Signal eine Update-Operation verursachen, was zur Bildung von zufälligen und nicht nützlichen Sequenzen führt. Abbildung 8.14 zeigt die verhältnismäßig schlechte Leistung, die mit $\gamma = -7$ erreicht wurde. Die beste Leistung wurde in diesem Experiment mit $\gamma = -2$ erreicht. Die anderen Parametereinstellungen für den Lernalgorithmus waren die gleichen wie im vorigen Experiment. Der Startpunkt und der Zielpunkt wurden am Anfang jedes Spiels zufällig gewählt.

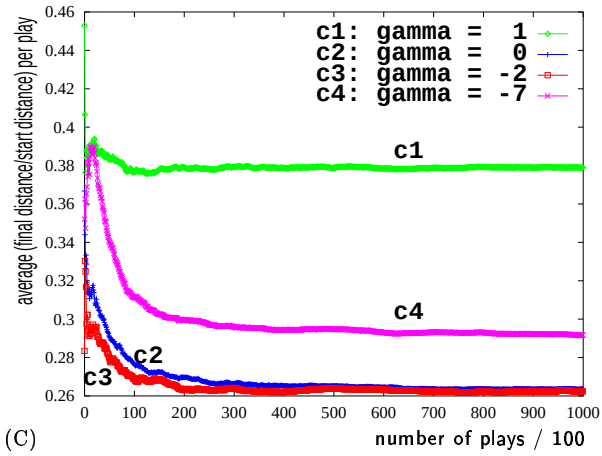


(A)



(B)

Abbildung 8.14: *Wirkung des Parameters γ von Gleichung (8.10) auf die Leistungsfähigkeit des Lernalgorithmus: c1: $\gamma = 1$; c2: $\gamma = 0$; c3: $\gamma = -2$; c4: $\gamma = -7$.*



(C)

Kapitel 9

ZUSAMMENFASSUNG UND AUSBLICK

9.1 Rückblick

Autonome mobile Systeme sind physische Agenten, die sich in dynamischen und unvorhersagbaren Umgebungen befinden. Der klassische Ansatz zur Realisierung von Autonomie basiert auf logischen Grundlagen. Die zentrale These dabei ist, daß intelligentes Verhalten durch eine Schlußfolgerungskomponente, die auf einem symbolischen Weltmodell operiert, realisiert werden kann. Man spricht in diesem Zusammenhang auch von Planung. Obwohl diese Systeme innerhalb einiger Anwendungsgebiete durchaus beachtenswerte Leistungen erzielen konnten, haben ihre Anwendungen in autonomen Systemen, die in komplexen und dynamischen realen Umgebungen agieren, nur unzufriedenstellende Ergebnisse erbracht. Die wenigen Systeme, die konstruiert wurden, weisen viele Unzulänglichkeiten wie mangelnde Robustheit und lange Antwortzeiten auf.

Als Alternative zu den klassischen Systemen wurden in den letzten Jahren verhaltensbasierte Systeme vorgeschlagen. Das Hauptaugenmerk in diesen Systemen liegt auf der engen Kopplung zwischen Perzeption (perception) und Aktion (action). Dies geschieht mit Hilfe einer Anzahl von speziellen reaktiven Prozessen, die meistens schnell und einfach sind, und die ein Minimum an expliziter Zustandsinformation enthalten. Das Ergebnis ist im allgemeinen eine hoch spezialisierte und angepaßte Lösung zu einer bestimmten Agent-Umwelt Situation. Diese Architekturen verlangen aber, daß der Designer des Systems in der Lage ist, die nötigen Prozesse zu entwerfen und für die richtigen Verbindungen zwischen ihnen zu sorgen. Mit steigender Komplexität heutiger Aufgaben wird es aber immer schwieriger, solchen Anforderungen zu genügen. Dieser Ansatz weist

außerdem, wegen der Festverdrahtung der Kontrollstruktur, einen großen Mangel an Flexibilität auf. Die Fähigkeiten des Agenten beschränken sich auf die Verhaltensmuster, die der Designer implementiert hat.

Im Rahmen dieser Arbeit wurde versucht, eine neue Steuerungsarchitektur für autonome mobile Systeme zu entwickeln, die die Vorteile beider oben erwähnten Ansätze nutzen soll.

Ein Modell, das die wichtigsten Eigenschaften autonomer Systeme, nämlich Reaktivität, Zielorientierung und Adaptivität berücksichtigt, wurde entworfen. Die Grundlage für dieses Modell ist eine Menge von konkurrierenden Prozessen, auch Verhaltensmuster oder Verhaltensmodule genannt, und eine auf Prioritäten basierende Arbitrierungsstrategie. Jedes Verhaltensmuster hat zu jedem Zeitpunkt und bezüglich jedes Ziels (goal: Die zu erledigende Aufgabe oder Task) eine bestimmte Priorität. Die Priorität spiegelt die Wichtigkeit des Verhaltensmusters für den Umgang mit der aktuellen Umweltsituation im Kontext des entsprechenden Ziels wider und wird dazu benutzt, das Verhaltensmuster zu bestimmen, das die Kontrolle im nächsten Kontrollzyklus übernehmen wird.

Die erste Frage, die sich daraus ergibt, ist: Wie können diese Prioritäten ermittelt werden? Ausgangspunkt ist das Anfangswissen des Designers über das System(z.B. Wissen über die Wirkung der Sensordaten auf die Verhaltensmuster) sowie eine Menge von Sensoren, die zu jedem Zeitpunkt Informationen über Umwelt und Ziel liefern. Eine der subtilen Eigenschaften der Sensorwerte ist, daß sie oft ungenau und manchmal auch falsch sind. Das Wissen des Designers ist meistens unvollständig.

Um ein technisches System zu bauen, das mit diesen Problemen fertig werden kann, müssen auch Konzepte zur Behandlung von Ungenauigkeit und Unvollständigkeit vorhanden sein. Ein solches Konzept ist z.B. Adaptivität. Das System ist adaptiv, wenn es in der Lage ist, sich selbst anzupassen und sein Verhalten zu verbessern. Adaptivität kann mit Hilfe von Techniken des maschinellen Lernens erreicht werden. Neuronale Netze sowie Methoden zum Lernen aus Erfahrung (Reinforcement Lernen) sind sehr nützliche Werkzeuge in diesem Zusammenhang. Neuronale Netze sind in der Lage on-line zu lernen und haben gute Generalisierungsfähigkeiten. Ungenaue und unvollständige Sensordaten können weiterhin eine Ausgabe erzeugen, die ursprünglich mit vollständigeren Daten as-

soziiert wurde. Reinforcement Lernen stellt einen Mechanismus dar, in dem der Designer Anforderungen und Zielvorgaben an das System allein durch Bewertung (,,Belohnung" (reward) und ,,Bestrafung" (punishment)) seiner Leistungsfähigkeit hinsichtlich der zu lösenden Aufgabe artikulieren kann. Einmal durchgeführte Aktionen werden somit als ,,Erfahrungen" im weiteren Betrieb nutzbringend eingesetzt.

In diesem Modell werden die Prioritäten mit Hilfe dieser Methoden gelernt. Zuerst wird versucht eine selbstorganisierende Karte (self-organizing map) zu konstruieren und gleichzeitig das Anfangswissen des Designers in diese Struktur zu integrieren. Damit wird das Problem der ungenauen und falschen Sensordaten weitgehend gelöst und gleichzeitig die Voraussetzung für die Anwendung von Reinforcement Lernen, nämlich die Generalisierung über Zustände, geschaffen. Wenn dies geschehen ist, wird Reinforcement Lernen angewandt, um die anfangs nicht optimale Steuerung zu verbessern und die Unvollständigkeit soweit wie möglich zu beheben. Dies geschieht mit Hilfe einer neuen Methode, die auf der Kombination von Exploration und Algorithmen der Dynamischen Programmierung beruht. Simulationsergebnisse haben die besondere Nützlichkeit dieser Vorgehensweise gezeigt. Mit einigen Grundfähigkeiten ausgestattet ist somit das System in der Lage, seine Funktionalität im Einsatz erfahrungsgestützt zu erweitern und zu verbessern und auch in unvorhersehbaren Situationen sinnvolles Verhalten zu zeigen.

Da nicht jedes beliebig komplexe Verhalten auf diese Weise leicht zu realisieren ist, stellt sich die Frage, wie der Agent eine komplexe Aufgabe schrittweise durch Koordination von bereits erlernten Verhaltensweisen erledigen kann. Zur Lösung dieses Problems wurde wiederum das Lernen aus Erfahrung benutzt. Bei der neuentwickelten Koordinationsmethode wird ein dynamisches neuronales Netz benutzt, um die Menge von Sequenzen von Parametervektorpaaren (ein Paar besteht aus einem Eingabeparametervektor und dem dazugehörigen Kontrollparametervektor) zu partitionieren. Der Agent versucht dabei, sich an erfolgreiche Sequenzen zu erinnern. In jedem Schritt wird die Sequenz, die am meisten Erfolg verspricht, benutzt. Abhängig vom Erfolg und der Vorhersagekraft der benutzten Sequenz wird dann die Sequenzmenge angepaßt. Die Zweckmäßigkeit dieser Methode wurde wiederum mit Hilfe von Simulationen erprobt.

Um ,,intelligente", lernfähige Systeme zu schaffen, die autonom in einer dynamischen Umwelt agieren können, müssen geeignete Vorstrukturierungen (a-priori-Wissen) einer-

seits und Lernalgorithmen andererseits in Lernarchitekturen vereinigt werden. Um dies zu leisten, ist es erforderlich, zu untersuchen, welche Arten von Information der menschliche Experte am einfachsten beitragen kann und welche Architekturen es am wirkungsvollsten erlauben, dieses Wissen mit Lernalgorithmen zu verknüpfen. Erste erfolgreiche Schritte in diese Richtung wurden in der vorliegenden Arbeit durch die Verbindung von verhaltensbasierten Ansätzen mit Techniken des maschinellen Lernens unternommen.

In den Experimenten, die in der vorliegenden Arbeit beschrieben sind, wurden simulierte Agenten benutzt. Obwohl es auf den ersten Blick scheint, daß für die Untersuchung von Robotern, die in realen Umgebungen agieren, Simulation ungeeignet ist, haben sich simulierte Umgebungen beim Testen von Designoptionen als sehr nützlich erwiesen. Die Ergebnisse der Simulation scheinen auch robust genug zu sein, um auf reale Umgebungen ohne große Schwierigkeiten übertragen zu werden (vorausgesetzt, daß die sensorischen und motorischen Fähigkeiten der Roboter denen ihrer simulierten Entsprechungen ähnlich sind). Außerdem kann das Trainieren in einer simulierten Umgebung und die anschließende Übertragung des resultierenden Kontrollers auf einen realen Roboter eine interessante, alternative Designmethode sein, weil die Verwendung von realen Robotern sehr zeitaufwendig ist . Wenn diese Übertragung sich dennoch als schwierig oder unnütz erweist, dann sollte es mindestens in einigen Fällen möglich sein, Verhaltensmodule, die in simulierten Umgebungen erlernt wurden, als Ausgangspunkt für das Trainieren von realen Robotern zu benutzen.

9.2 Verbindung zu anderen Arbeiten

Die vorliegende Arbeit ist mit einer Reihe von neueren Forschungsarbeiten verwandt, die sich auf die Realisierung von kompletten, mit der physischen Welt eng verbundenen, künstlichen Agenten konzentrieren. Diese Systeme werden oft ,,eingebettete" (embedded) oder ,,situierte" (situated) Agenten genannt. Beispiele für diesen Trend stellen die in [1],[76], [23],[24], und [172] beschriebenen Arbeiten dar. Obwohl es wichtige Unterschiede zwischen den verschiedenen Ansätzen gibt, scheinen einige gemeinsame Punkte sich gut etabliert zu haben. Eine erste grundlegende Anforderung ist, daß Agenten in der Lage sein müssen, ihre Aktivität in der realen Welt und in Realzeit auszuüben. Ein anderer wichtiger Punkt ist, daß adaptives Verhalten nur aus der engen Kopplung des

Agenten mit seiner Umgebung entstehen kann. Die vorliegende Arbeit hat sich darauf konzentriert, solch eine Kopplung durch Benutzung von Selbstorganisation und Reinforcement Lernen zu erzielen. Die Kontrollstruktur wird während der Interaktion des Agenten mit seiner Umwelt dynamisch entwickelt.

Selbstorganisation ist eine Art von Verhalten, das dynamische Systeme aufweisen. Beispiele für dieses Verhalten sind in vielen Bereichen der Wissenschaft reichlich vorhanden (siehe z.B. [72]). Selbstorganisation verweist auf die Fähigkeit eines Systems, das unüberwachtes Lernen benutzt, spezifische Detektoren für verschiedene Signalmuster zu entwickeln. Unüberwachtes Lernen (z.B. [81]) unterscheidet sich von überwachtem Lernen (z.B. [137]) in der Art der Information, die dem System zugeführt wird. Bei überwachtem Lernen muß ein ,,Lehrer" die Klasse, zu der jedes Trainingsbeispiel gehört, liefern. Bei unüberwachtem Lernen geht es hingegen darum, Regularitäten in den Trainingsbeispielen zu finden. Selbstorganisation in lernenden, adaptiven Kontrollern ist nützlich, weil es unwahrscheinlich ist, daß der Designer detailliertes Wissen darüber hat, wie die Umgebung mittels der Sensoren des Agenten charakterisiert werden kann und folglich es nicht einfach ist, die passenden Kontrollparameterwerte zu finden, die der Kontroller benutzen soll, wenn er spezifischen Umgebungen gegenübersteht. Ein System, das in der Lage ist, Umgebungssituationen autonom zu charakterisieren, kann nützliche Kontrollparameter mit den Situationen assoziieren und den Kontroller erfolgreich adaptieren.

Reinforcement Lernen (siehe z.B. [157]) wurde in den letzten Jahren in vielen verschiedenen algorithmischen Rahmen studiert. In [159] zum Beispiel haben Sutton, Barto und Williams die Benutzung von Reinforcement Lernen für die direkte adaptive Steuerung vorgeschlagen. Mahadevan and Connell ([89]) haben eine ,,Subsumption Architecture" ([19]) implementiert, in der die Verhaltensmodule mit Hilfe einer erweiterten Version des Q-Learning-Algorithmus ([168]) lernen. Methoden des Reinforcement Lernens kombinieren Methoden zur Anpassung von Aktionsauswahl mit Methoden zur Einschätzung der langfristigen Konsequenzen von Aktionen. Die Grundidee in Algorithmen des Reinforcement Lernens wie Q-Learning ist, eine Funktion $Q(.,.)$ von Zuständen und Aktionen abzuschätzen. Dabei ist $Q(x,a)$ der Erwartungswert der diskontierten Summe der zukünftigen Rewards, die dadurch erzeugt werden, daß Aktion a in Zustand s ausgeführt wird und danach nur optimale Aktionen ausgewählt werden.

9.3 Einschränkungen

Mit Hilfe der in dieser Arbeit vorgestellten Lernmethoden ist es möglich, Agenten zu entwerfen, die selbständig ihre Robustheit verbessern können. Die Agenten sind auch in der Lage, selbständig komplexe Tasks zu erlernen und bereits vorhandene Kompetenzen zu koordinieren, um komplexere Ziele zu lösen. Es wurde allerdings vorausgesetzt, daß eine konsistente und vollständige Menge von Basis-Verhaltensmustern bereits existiert, und daß es außerdem möglich ist, für die zu lösende Aufgabe eine Komplexitätshierarchie (Goal-Hierarchie) zu definieren. Für allgemeine Problemstellungen ist es aber schwierig, die Konsistenz und die Vollständigkeit der Mengen von Basis-Verhaltensmustern zu garantieren. Es ist auch schwierig, passende Komplexitätshierarchien zu definieren.

Eine weitere Einschränkung besteht in den vielen Parametern, die bei der Anwendung der Lernalgorithmen vom Designer ,, von Hand" passend eingestellt werden müssen. Für manche Parameter können allgemeine Einstellungsregeln empirisch gefunden werden. Viele andere Parameter sind aber von den Einzelheiten der Aufgabenstellung abhängig und lassen sich nur sehr schwer im voraus bestimmen.

Ein weiteres Problem besteht, wie in allen nicht analytischen Lösungen, in der Schwierigkeit, die Korrektheit der Lösung zu beweisen. Die detaillierten Problemlösungsaktionen des Agenten können nur zur Laufzeit bestimmt werden (Emergence). Einzelnes Verhalten wird durch eine komplexe Wechselwirkung zwischen Agent und Umgebung geregelt. Die genaue ,,Trajektorie", der der Agent folgen wird, kann nur entdeckt werden, indem man den Agenten in seiner Umgebung laufen läßt. Das Verhalten des Agenten kann somit zur Designzeit nicht eindeutig festgestellt werden. Dies führt zu Einschränkungen bei der Anwendung in sicherheitskritischen Gebieten, in denen es wichtig ist, daß das System seine Spezifikation erfüllt.

All diese Probleme und Einschränkungen erfordern weitere Forschungsarbeit. Eine wichtige Forschungsrichtung wäre es, den Automatisierungsprozeß voranzutreiben, um die Anzahl der Designentscheidungen, die vom Designer getroffen werden müssen, zu reduzieren. Man könnte zum Beispiel versuchen, aus der allgemeinen Aufgabenstellung automatisch eine Komplexitätshierarchie zu erzeugen. Diese Art von High-Level-Planung würde die Autonomie des Agenten entscheidend verbessern und zur Lösung einiger der oben erwähnten Probleme beitragen (z.B. Korrektheitsgarantien). Man könnte auch ver-

suchen, mehr automatische Parametereinstellungsmechanismen in die Lernalgorithmen zu integrieren. Das Testen der Lernarchitektur in realistischeren Umgebungen sollte die Aspekte der Interaktion zwischen Agent und Umgebung deutlicher machen und Anlaß zu Verbesserungen geben. Die Anwendbarkeit der Architektur für andere Arten von Agenten, z.B. Software-Agent, Multi-Agenten-Systeme, usw., könnte auch untersucht werden.

Insgesamt glaube ich, daß die vorliegende Arbeit die Wichtigkeit von Lernen für das Erreichen eines zufriedenstellenden Adaptierungslevels zwischen einem künstlichen Agenten und seiner Umgebung zeigt. Es ist aber klar, daß noch viel Forschungsarbeit notwendig ist, um verstehen zu können, ob mit dem hier beschriebenen Ansatz eine mit dem adaptiven Verhalten eines lebenden Systems vergleichbare Komplexitätsstufe erreicht werden kann.

9.4 Die Zukunft

Robotik-Agenten gehören zur allgemeinen Klasse von intelligenten Agenten, die ein neues Paradigma für die Entwicklung von Softwareanwendungen darstellen. Intelligente Agenten sind gegenwärtig Gegenstand von intensivem Interesse seitens vieler Unterbereiche der Informatik (siehe z.B. [73]). Agenten werden in einer zunehmend breiten Vielzahl von Anwendungen benutzt, die von vergleichsweise kleinen Systemen wie Email-Filter zu großen, offenen, komplexen und kritischen Systemen wie Flugverkehrssteuerung reichen. Obwohl es auf den ersten Blick erscheint, daß solche extrem verschiedenen Arten von Systemen nur wenig gemeinsam haben könnten, ist der Begriff *Agent* eine Schlüsselabstraktion und ein grundsätzliches und wichtiges neues Werkzeug für die Konstruktion solcher Systeme. Grund dafür ist, daß die Metapher von autonomen Ploblemlösungsinstanzen eine intuitive und natürliche Art für die Konzeptualisierung mehrerer Probleme darstellt.

Im Moment herrscht noch eine rege Diskussion darüber, welche Eigenschaften Agenten im einzelnen aufweisen sollen, welche Anforderungen an sie gestellt werden müssen und wie eine allgemeingültige Definition des Begriffs ,,intelligenter Agent" lauten könnte. Robuste und effiziente Systeme erfordern aber auf jeden Fall methodische und rigorose

Designmethoden. Es gibt mehrere generelle Ansätze zum Design von Agenten, von denen der Ansatz der Selbstadaptation am geeignetsten erscheint. Dabei wird davon ausgegangen, daß ein Agent aus der Beobachtung seiner Umwelt - insbesondere der Beobachtung seines Benutzers - lernt und schrittweise immer nützlicher wird.

Die vielversprechende Idee des Agenten, und insbesondere die des lernenden Agenten, eröffnet einen weiten Horizont neuer, intelligenterer und vor allem flexiblerer Möglichkeiten, mit dem Informationsangebot der Zukunft umzugehen. Durch sie vollzieht die Wissenschaft des Informationsmanagements einen weiteren Schritt hin zur Verselbständigung von Software und Computern. Damit geschieht hier eine Entwicklung parallel zum Fortschritt der Programmiersprachen, die, angefangen von Lochkarten, hin zu immer ,,höheren" Befehlen strebten und noch immer streben. Welchem progressiv Denkenden leuchten nicht die Augen bei dem Gedanken, endlich mit einer Maschine in der ,,höchsten Sprache", der menschlichen Lautsprache kommunizieren zu können !?

Doch bis zur Verwirklichung dieser euphorischen Vision - so sie denn überhaupt jemals erreicht wird - ist es noch ein weiter Weg und eine große Anforderung an die Informatik, diesen Weg begehbar zu machen.

Anhang A

ENGLISH SUMMARY

Designing adaptive learning Robots

Abstract: *This chapter gives a short summary of the dissertation in English. More details can be found in the given literature. The dissertation deals with a self-improving reactive control system for autonomous agents. It relies on the emergence of more global behavior from the interaction of smaller behavioral units. To simplify and automate the design process techniques of learning and adaptivity are introduced at three stages: first, improving the robustness of the system in order to deal with noisy, inaccurate, or inconsistent sensor data, second, improving the performance of the agent in the context of different goals (behaviors), and third, extending the capabilities of the agent by coordinating behaviors it is already able to deal with to solve more general and complex tasks.*

A.1 Introduction

The autonomy of an agent is defined as the ability to operate independently in a dynamically changing and complex environment. The classical AI approach towards autonomous agents is based on logical foundations, as exemplified by Shakey [116]. More recently, an alternative approach, known as the bottom-up or behavior-based approach, has been explored by several researchers, e.g. Brooks [19]. Some important limitations of this new approach are its lack of goal-directedness (it provides no guarantee to fulfill the goal) and flexibility (the control is entirely wired and the robot is limited to the behaviors implemented by its designer). In recent years much effort has been invested to improve the original proposal. Representative examples of these new approaches include the work

described in [7, 42, 48] concerned with hybridizing reactive and classical systems and in [89, 103] concerned with introducing effective learning capabilities.

Our work is focused on improving the above mentioned shortcomings of reactive systems by combining run-time arbitration, goals and learning. The approach we propose for designing autonomous agents relies on the emergence of more global behavior from the interaction of smaller behavioral units. The basic idea consists of achieving goals by switching basic-behaviors on and off. To do this, priorities for the basic-behaviors are computed at each time-step using the current sensor data and the knowledge of the designer. Because it may be very difficult, if not impossible, for a human designer to incorporate enough world knowledge into an agent from the very beginning, machine learning techniques may play a central role in the development of intelligent autonomous agents. We introduced learning and adaptivity techniques at three different stages of the design process: first, improving the robustness of the system in order to deal with noisy, inaccurate, or inconsistent sensor data. This is achieved by using a self-organizing map and by integrating the knowledge of the designer into this map. Second, improving the performance of the agent with regard to the individual goals (behaviors) separately. This is achieved by using exploration algorithms combined with dynamic programming techniques. And third, coordinating the individual goals to solve more general and complex tasks and get an optimal overall behavior of the system. This is achieved by combining a dynamic self-organizing network with reinforcement learning.

In this chapter we give a brief summary of the control architecture. Section A.2 describes the design methodology and the basic idea (see [56] for more details). Section A.3 gives an overview of the part concerning the improvement of the robustness of the system (see [59, 61] for more details). Section A.4 describes the part concerned with improving the performance of the system in the context of different goals (see [62] for more details). Section A.5 describes the method for coordinating goals (see [64] for more details).

A.2 Control architecture

We investigate a methodology using concurrent processes (behavior modules, in the following referred to as behaviors) and a priority-based arbitration scheme. The behaviors

are organized in different levels (see figure A.1).

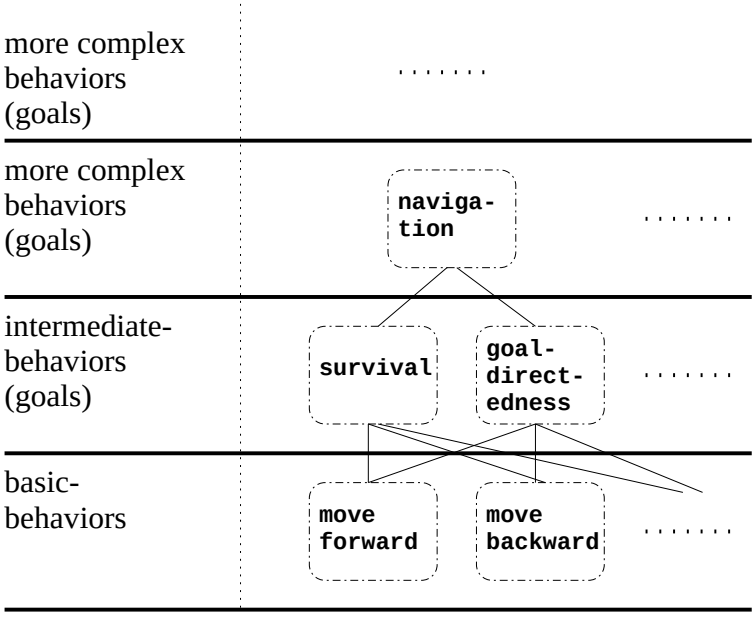


Fig. A.1: *Organization of the behaviors (goals) in different complexity levels.*

The lowest level consists of a set of basic-behaviors which define the basic abilities of the agent. They interact in order to fulfill the tasks the agent should deal with. They can be implemented using any appropriate method.

The next higher level consists of a set of intermediate-behaviors or goals that can be specified by switching basic-behaviors on and off. To achieve a certain goal, each of the basic-behaviors is assigned at each time-step a priority which is a value that reflects how good is the basic-behavior for fulfilling the required goal. To compute the priorities of the basic-behaviors for a given goal, the designer is asked first to use all its knowledge about the system to model the dependency of the basic-behaviors on the sensors. The dependency of a basic-behavior on the sensors is specified by several functions that state the effects of perceptual values on the activity of the basic-behavior (with respect to the given goal). The priority of the basic-behavior is then calculated by summing up all these effects.

All subsequent levels consist of more and more complex goals (behaviors). The higher is the level the higher is its competence. Complex goals are achieved by coordinating less complex goals. Coordination consists of computing at each time-step for each

basic-behavior its priority with regard to the complex goal. The priority is obtained by combining the priorities with regard to the individual less complex goals.

An important application example is autonomous robotic navigation , i.e., the task to find a path from a source point to a destination point. The complex behavior "navigation" can, for example, be achieved by coordinating the two intermediate-behaviors G_1 = "survival" and G_2 = "goal-directedness" (see figure A.1). "survival" takes care that agent survives, i.e., to have always enough energy, to avoid obstacles, ... "goal-directedness" ensures that the agent moves towards its destination. If we suppose that there is a set of basic-behaviors $\{B_1, B_2, \ldots\}$ that allow movement, their priorities with regard to "survival"(G_1) and "goal-directedness" (G_2) can be calculated at each time-step as follows:

$$P_{G_1}(B_i,t) = \frac{1}{2}\left[1 + \frac{1}{n}\sum_{j=1}^{n} f_{i,j}(s_j(t))\right],\ P_{G_2}(B_i,t) = \frac{1}{2}\left[1 + \frac{1}{n}\sum_{j=1}^{n} g_{i,j}(s_j(t))\right]$$

where $f_{i,j}(s_j(t))$ denotes the effect of perceptual value s_j on basic-behavior B_i at time t with regard to the goal "survival" and $g_{i,j}(s_j(t))$ that with regard to the goal "goal-directedness" (see figure A.2). n is the number of perceptual values. The priority of each basic-behavior with regard to the complex goal "navigation" can be obtained by combining its priority with regard to "survival" and its priority with regard to "goal-directedness".

A.3 Improving the robustness of the system

The component of a behavior dedicated to the computation of its priority with respect to a certain goal (the priority-component) may be thought of as the formulation of an input-output mapping, the input being some pattern of raw sensor data and the output being a value expressing the impact of the current environmental situation on the activation of the behavior with regard to that goal. This mapping, initially specified by the designer, may be adjusted and tuned using a neural network. Artificial neural networks are able to learn on-line and also have useful generalization characteristics. Noisy and incomplete input patterns can still trigger a specific response that was originally associated with more complete data.

The network algorithm that we use consists of the combination of two parts. The first part is a Kohonen self-organizing feature map [81] that is common for all behaviors. The

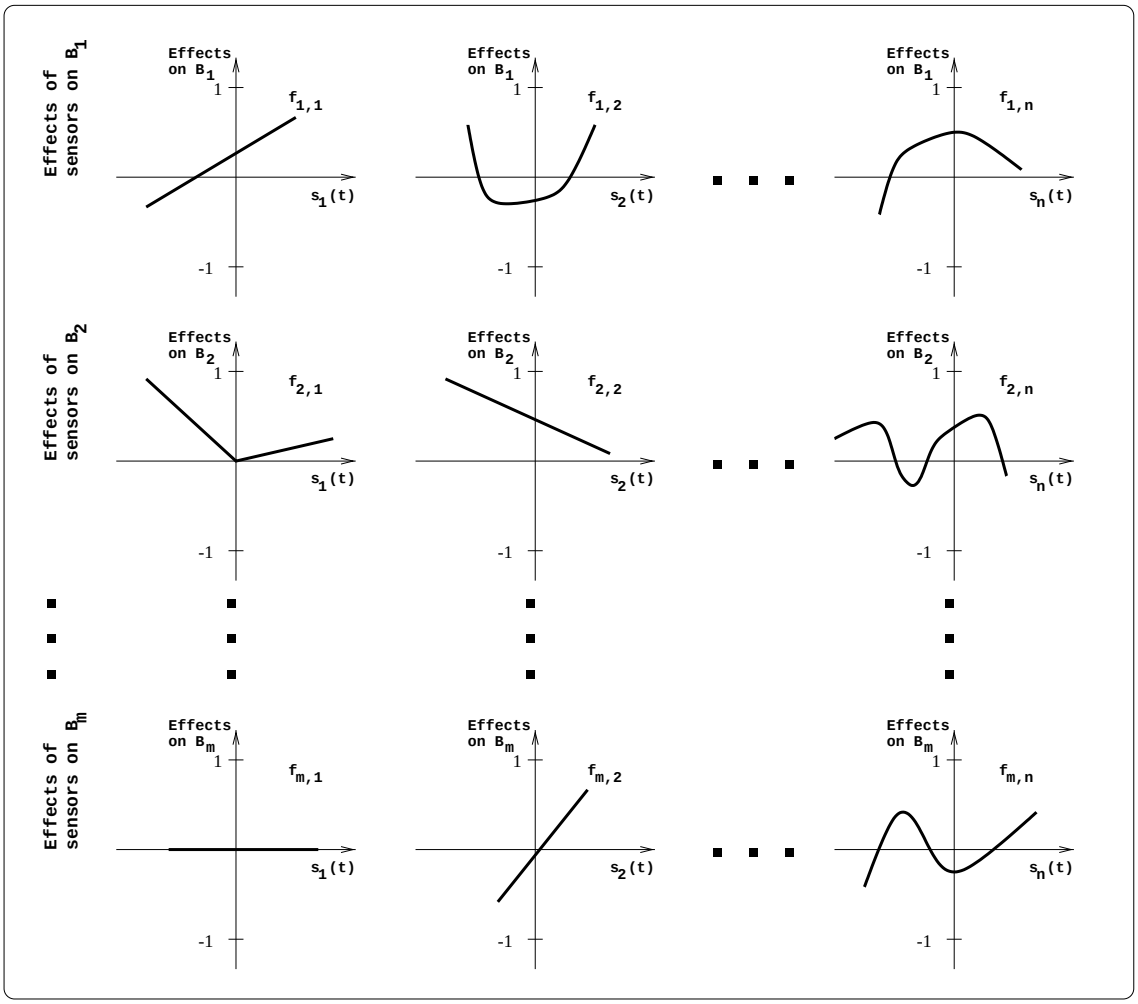


Fig. A.2: *Designing a goal (intermediate-behavior) by specifying the dependency between sensors and basic-behaviors. To achieve a specific goal basic-behaviors are selected according to priorities that are computed at each time-step by summing up the effects of all perceptual values on each basic-behavior.* $s_1(t)$ *is the perceptual value delivered by sensor number 1,* $s_2(t)$ *that by sensor number 2 and so on.* $B_1, B_2, \ldots, B_m$ *are the basic-behaviors.*

second part is distributed over all behaviors. Figure A.3 shows the integration of the association network in the original model for computing priorities. Two types of learning are being simultaneously applied: unsupervised learning concerned with building the self-organizing map and supervised learning in each behavior concerned with integrating

the knowledge of the designer (priority–component) into the system by associating each region of the map (each neural unit) with priority values for the different behaviors(see [59, 61] for details).

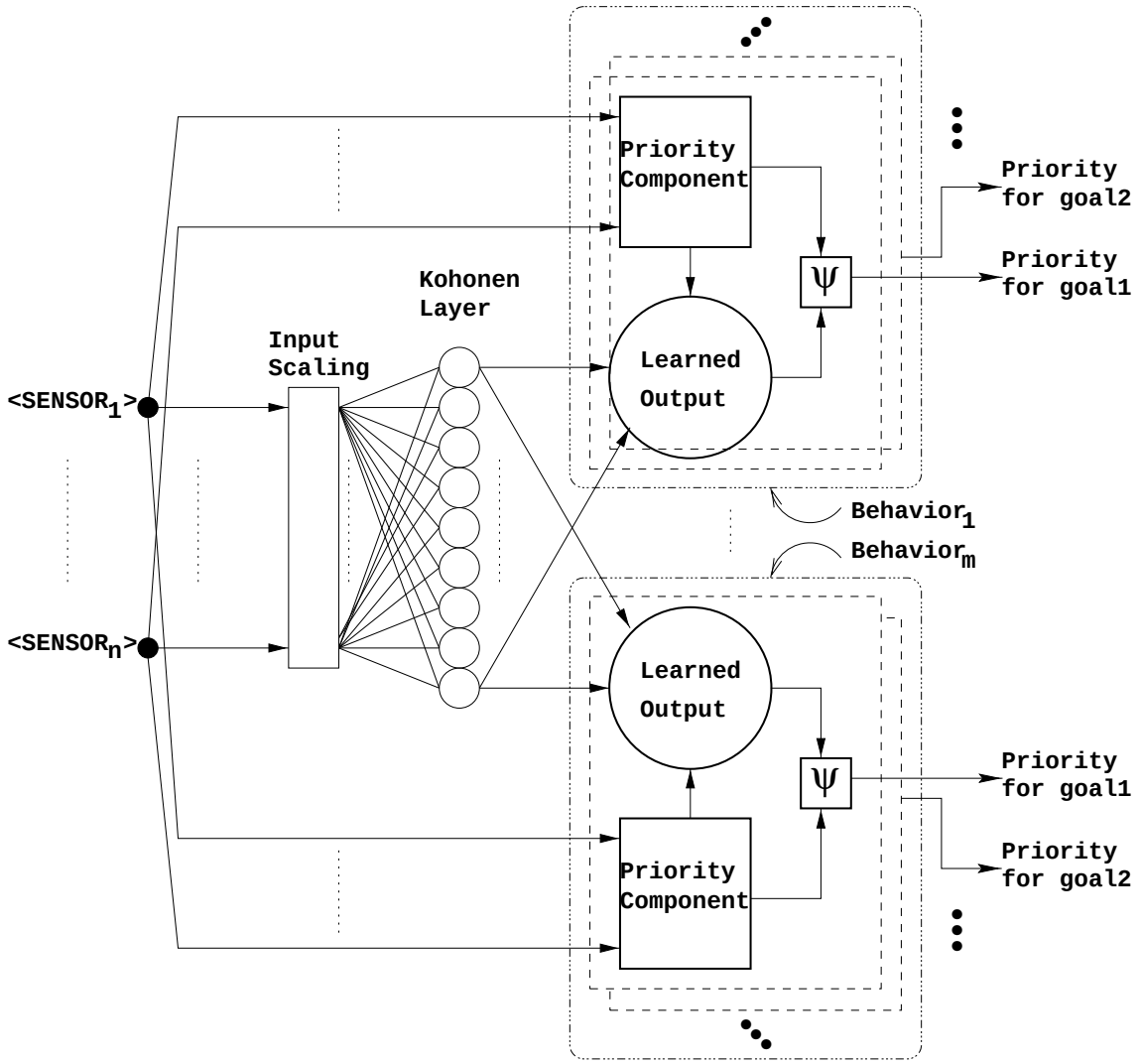


Fig. A.3: *Addition of an association network to the original model for computing priorities. Input values (sensor data) are routed both to the Kohonen layer and to the priority–component of each behavior. The output of a priority–component is used to train the corresponding network part in order to integrate the available knowledge into the neural control system.*

This neural control structure has many desirable characteristics such as: i) robustness,

i.e., the system is able to deal with noisy and incomplete sensor data, ii) ability to be improved using reinforcement learning (the self-organizing map solves the problem of generalization over states which is important in reinforcement learning), iii) possibility of parallel computations, i.e., each behavior computes its own mappings.

Figure A.4 shows for example how the robustness of an agent acting in a simulated dynamic environment is improved using such a neural structure. The dynamic environment involves the agent, enemies, food, and obstacles. The enemies chase the agent, food provides the agent with additional energy, and moving costs the agent energy. At each time-step the agent has four basic-behaviors to choose from: "move forward", "move backward", "move left", and "move right". The agent is allowed to see only the area surrounding it. It has four sensors that are used to sense objects in the directions forward, backward, left and right. Each of the sensors delivers a real perceptual value which depends on the kind of the detected object and its distance from the agent. A play ends when the agent collides with an enemy or an obstacle or runs out of energy. The agent should try to survive as long as possible.

A.4 Improving the performance of the system with regard to specific goals

The computations presented above assume that the designer of the agent is able to model the dependency of behaviors on sensors in the context of all goals and specify the necessary functions correctly. Unfortunately, this is usually a difficult task. The designer has usually only little knowledge about the system. Even when the effects of sensors on behaviors are known, this knowledge is often only immediate, i.e. the delayed consequences of actions are unknown.

Reinforcement learning (RL) provides an appropriate framework for solving these problems. It aims to adapt an agent to an unknown environment according to rewards. Many RL methods exist that can handle delayed reward and uncertainty, e.g., Q-learning. The goals these systems learns, however, are implicit, i.e., what is learned in the context of one goal, cannot be applied in the context of another goal (e.g., all of the Q-values are goal dependent). Furthermore, most of these systems converge slowly.

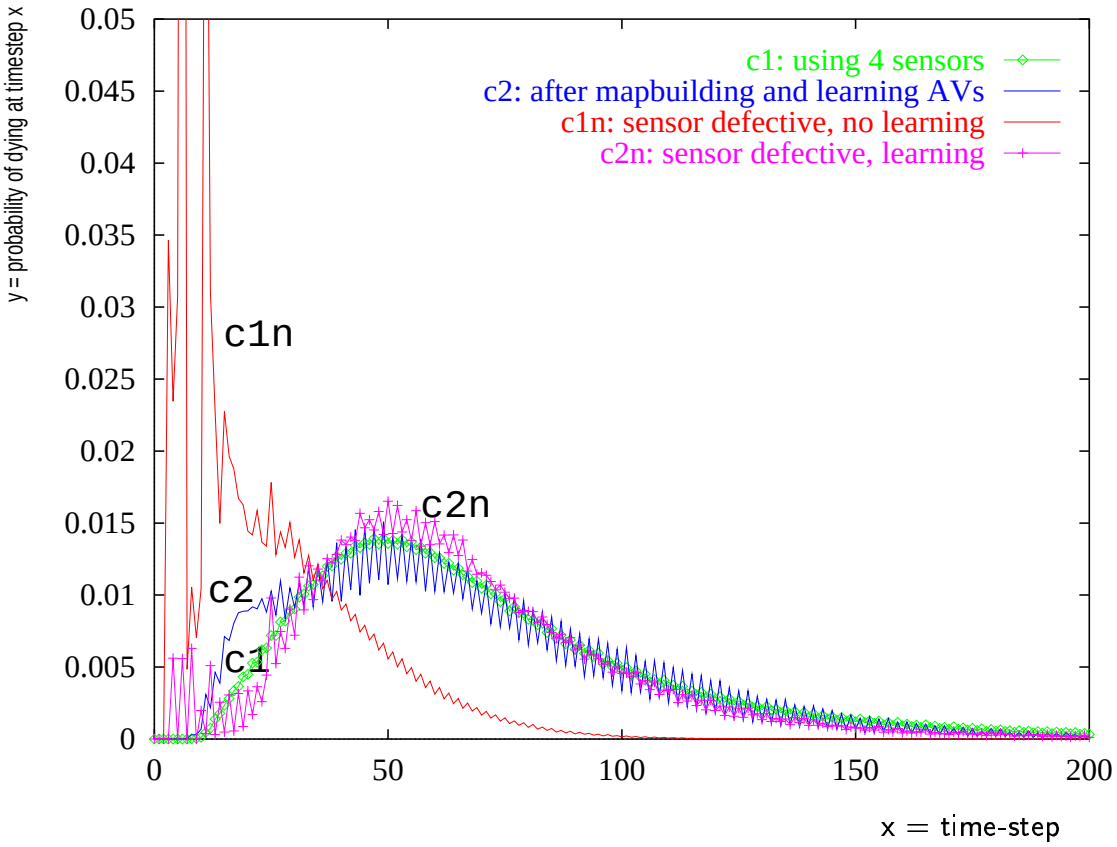


Fig. A.4: *Improving the robustness of the agent in the context of the goal "survival". c1: Performance of the agent when the dependencies between sensors and basic-behaviors (priority-components) are specified by the designer. c2: The performance of the agent after map-building and integrating the knowledge of the designer into the neural structure is almost the same as that of the teacher (the priority-components are used during learning as a teacher). c1n: The agent performs very bad when a sensor is defective and no neural structure is used. c2n: With the neural structure there is only a small performance deviation when a sensor is defective.*

The method we use achieves the purpose of RL and at the same time deals with many of these problems. It consists of identifying the environment using exploration and then determining an optimal policy using dynamic programming.

The overall algorithm for improving the policy of the agent, i.e., improving the priorities of the different behaviors in the context of the different goals after having a first appro-

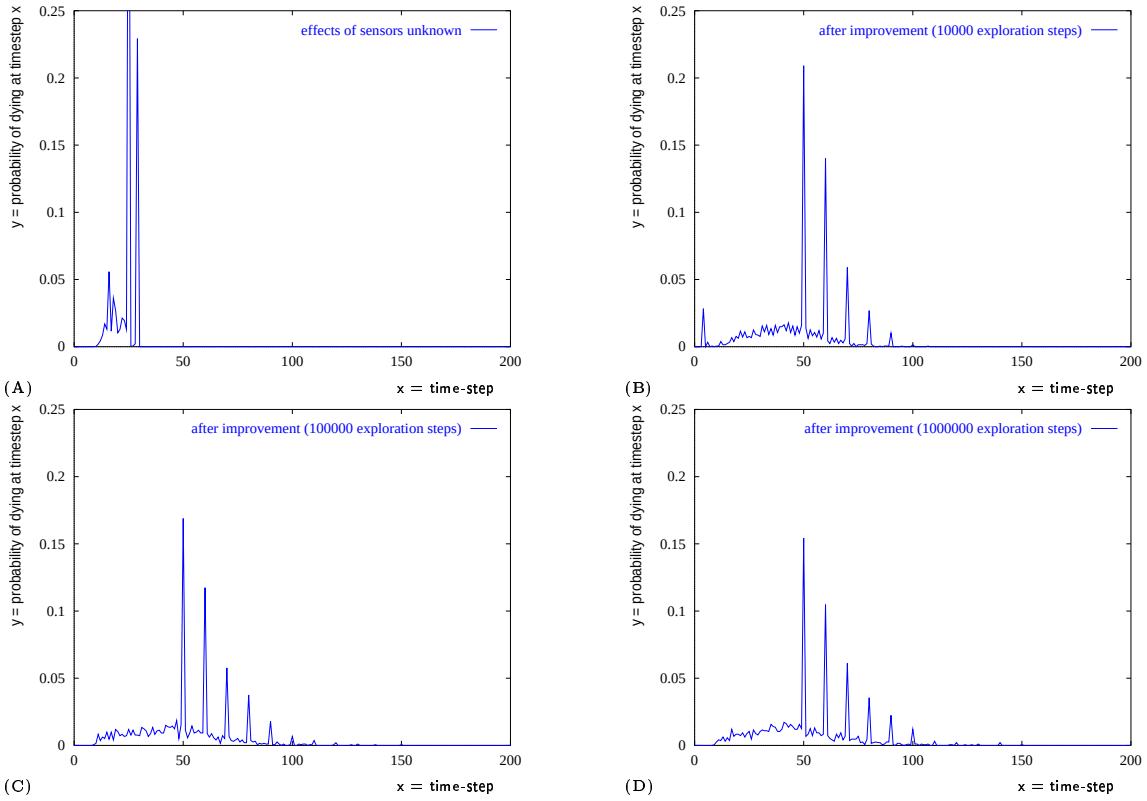
ximation (obtained after map-building and integrating the available knowledge into the neural control structure (see section A.3)) of the priorities is as follows:

- Determine an approximation of the dynamics of the environment using exploration.
- For each goal (e.g., "survival", "goal-directedness", ...):
  - Evaluate the already available policy for the goal using policy evaluation. (The available policy is given by the priorities of the different behaviors with respect to the goal).
  - Use the obtained value function and value iteration to determine an optimal policy for the goal.
  - Transform the obtained policy back into priorities for the different behaviors.

The proposed improvement method was shown to be very effective especially when the designer has only little knowledge about the system. Figure A.5 shows for example how the agent acting in the simulated environment described in the previous section (with slightly changed parameter settings) is able to learn the goal "survival" starting from a control structure for which only the effects of one of the four sensors are known. The method has also the advantage to be able to learn in the context of many different goals simultaneously. Another advantage is the principled exploration method which seems to make the learning system faster then other RL paradigms. See [62] for details.

A.5 Extending the capabilities of the agent

Extending the capabilities of an agent is done by combining behaviors it is already able to cope with to produce more complex emergent ones. The method consists of a combination of map-building using a dynamic self-organizing feature map with output and reinforcement learning. The main role of the map-building strategy is to remember previous successful control parameters (coordination parameters) and use them when the agent faces similar circumstances. The main role of the reinforcement learning strategy is to reinforce associations (mappings between situations and control parameters) that tend to produce useful results.



	(A)	(B)	(C)	(D)
I0 = [0, 50]	0.999758	0.618705	0.549949	0.553702
I1 =] 50, 60]	0.000000	0.226379	0.207003	0.187174
I2 =] 60, 70]	0.000000	0.097842	0.111226	0.115224
I3 =] 70, 80]	0.000000	0.039808	0.071061	0.069864
I4 =] 80, 90]	0.000000	0.014388	0.033986	0.039625
I5 =] 90,100]	0.000000	0.001918	0.012873	0.017727
I6 =]100,110]	0.000000	0.000480	0.008239	0.008342
I7 =]110,120]	0.000000	0.000000	0.002575	0.003128
I8 =]120,130]	0.000000	0.000000	0.001545	0.002086
I9 =]130,140]	0.000000	0.000000	0.000515	0.002086
I10 =]140,150]	0.000000	0.000000	0.000515	0.000521
I11 =]150,160]	0.000000	0.000000	0.000000	0.000000
I12 =]160,170]	0.000000	0.000000	0.000000	0.000000
I13 =]170,180]	0.000000	0.000000	0.000000	0.000000
I14 =]180,190]	0.000000	0.000000	0.000000	0.000000
I15 =]190,200]	0.000000	0.000000	0.000000	0.000000
I16 =]200,...[	0.000000	0.000000	0.000000	0.000000
Performance	0.000000	0.611511	0.932544	0.986966

Fig. A.5: *Learning the goal "survival" starting from a control structure for which the effects of many sensors are unknown. The table summarizes the probabilities of dying in the intervals* $I_0, .., I_{16}$ *for the cases (A), (B), (C), and (D).*

The basic learning scheme consists of the following steps: first, using the feature map to generate control parameters for the current situation, second, observe the agent operating with these parameters, and third, update the feature map according to the outcome.

We extended a Kohonen network by allowing a variable number of cells and associating an output vector with each cell. The network algorithm obtained was capable of generalization over sequences of situations (input parameters and corresponding output parameters), i.e., it learned to partition the input space (space of all possible sequences of situations) incrementally into several regions using the history of the system-environment interaction. Combining this network algorithm with reinforcement learning allowed the system to determine the appropriate control parameters automatically.

Reinforcement learning was integrated as follows (see [64] for more details): the update procedure does not blindly update the selected sequence to be more similar to the current interaction history; instead it takes into account the reward feedback in such a way that control parameters that produce beneficial rewards are remembered while control parameters that produce bad rewards are modified.

This behavior coordination method was evaluated through many simulation studies and was shown to be very effective (see [64]). Figure A.6 shows for example how an agent (an extended version of the agent used in the previous sections) that learns to coordinate the goals "survival" and "goal-directedness" to achieve the more complex goal "navigation" (see figure A.1) incrementally increases its performance. The learning system (the system that uses this coordination method) was compared with a random system that changes the coordination parameters randomly at each time-step (initial configuration of learning system) and with a static system that uses fixed coordination parameters which were determined manually and found to be quite effective (a desirable configuration for the learning system).

A.6 Related work

Our work situates itself in the recent line of research that concentrates on the realization of complete artificial agents strongly coupled with the physical world and usually called "embedded" or "situated" agents. Examples of this trend include [1],[76], [23],[24], and

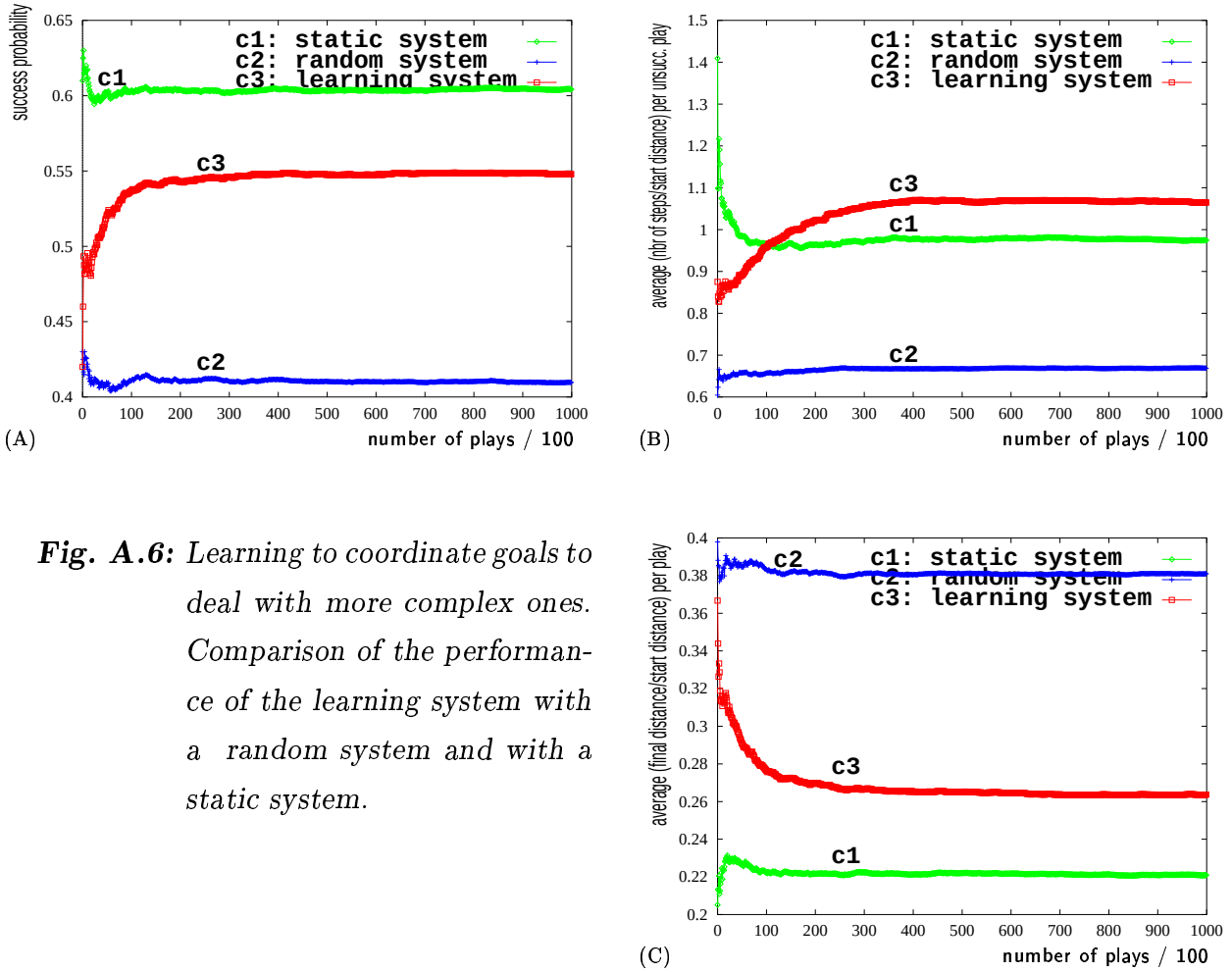


Fig. A.6: *Learning to coordinate goals to deal with more complex ones. Comparison of the performance of the learning system with a random system and with a static system.*

[172].

While there are important differences among the various approaches, some common points seem to be well established. A first, fundamental requirement is that agents must be able to carry on their activity in the real world and in real time. Another important point is that adaptive behavior cannot be considered as a product of an agent in isolation from the world, but can only emerge from a strong coupling of the agent and its environment.

Our research has concentrated on a particular way to obtain such a coupling: the use of self-organization and reinforcement learning to dynamically develop a robot controller through interaction of the robot with the world.

Self-organization is a type of behavior exhibited by dynamical systems, and examples

of this behavior are abundant in many areas of science (see, e.g., [72]). Self-organization refers to the ability of an unsupervised learning system to develop specific detectors of different signal patterns. Unsupervised learning (e.g. [81]) tasks differ from supervised learning (e.g. [137]) tasks in the type of information supplied to the system. In supervised learning, a "teacher" must supply the class to which each training example belongs, whereas in unsupervised learning the goal is to look for regularities in the training examples. Self-organization in learning adaptive controllers is useful because it is unlikely that designers would have detailed knowledge of how to characterize the environment in terms of the agent's sensors. Thus, it is not easy to select the appropriate control parameters values the controller must use when facing specific environments. A system that is able to characterize environmental situations autonomously can associate useful control parameters to environmental situations and adapt the controller successfully.

Reinforcement learning (see, e.g., [157]) has recently been studied in many different algorithmic frameworks. In [159], for example, Sutton, Barto and Williams suggest the use of reinforcement learning for direct adaptive control. Mahadevan and Connell ([89]) have implemented a subsumption architecture ([19]) in which the behavioral modules learn by extended versions of the Q-learning algorithm ([168]). Reinforcement learning methods combine methods for adjusting action-selections with methods for estimating the long-term consequences of actions. The basic idea in reinforcement learning algorithms such as Q-learning ([168]) is to estimate a real-valued function, $Q(.,.)$, of states and actions, where $Q(x,a)$ is the expected discounted sum of future rewards for performing action a in state x and performing optimally thereafter.

A.7 Conclusions

This dissertation has introduced a new Method for the design of intelligent behavior in a behavior-based architecture using priorities and learning techniques. The method relies on the emergence of more global behavior from the interaction of smaller behavioral units.

Fairly complex interactions can be developed even with simple basic-behaviors. In particular, behavior patterns that appear to follow a sequential plan can be realized by

the agents when there is enough information in the environment to determine the right sequencing of actions. The addition of elements such as a memory of past perceptions and reactions improves the level of adaptation to the dynamics of the environment.

The agents are strongly coupled with their environment through their sensorimotor apparatus, and are endowed with an initial control structure that has the ability to adapt itself to the environment and to learn from experience. To develop an agent, both explicit design and machine learning have an important role. Although, no much initial knowledge is required (all dependency functions can be chosen arbitrarily, e.g., all equal to zero), it is sensible that the designer implements all its knowledge about the system into the initial control structure. This will avoid long learning periods and allow the system to become able to deal with its task in a satisfactory way early enough.

For our experiments, we used simulated agents. The proposed learning methods were shown to be very effective especially when the designer has only a little knowledge about the system. Simulation appears at the first moment to be inadequate for real world robots. However, simulated environments have proved very useful to test design options, and the results of simulations seem to be robust enough to carry over to the real world without major problems, provided the sensory and motor capacities of the robots are similar to those of their simulated counterparts. Furthermore, since the use of real robots is too time-consuming, training in a simulated environment and then transferring the resulting controller to a real robot can be a viable alternative. At least in some cases it is possible to use behavioral modules learned in simulated environments as a starting point for real robot training.

The results obtained are encouraging and suggest that the whole architecture is a promising approach to building complete autonomous learning systems. The architecture is not limited to robotic agents. It can also be applied to design goal-directed behavior for other kinds of agents such as software agents.

LITERATURVERZEICHNIS

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87.* Morgan Kaufmann, Los Altos, CA, 1987.

[2] J. Albus, H. McCain, and R. Lumia. Nasa/nbs standard reference model for telerobot control system architecture (nasrem). NBS Technical Note 1235, Robot Systems Division, National Bureau of Standards, 1987.

[3] J. A. Anderson. Neural models with cognitive implications. In D. LaBerge and S. J. Samuels, editors, *Basic Processes in Reading Perception and Comprehension Models*, pages 27–90, Hillsdale, NJ, 1977. Erlbaum.

[4] Michael A. Arbib. *Brains, Machines and Mathematics.* McGraw-Hill, New York, NY, 1964.

[5] Ronald C. Arkin. Motor schema-based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 264–271, Raleigh, NC, 1987.

[6] Ronald C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, August 1989.

[7] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[8] Ronald C. Arkin. Intelligent robotic systems. *IEEE Expert*, 10(2):6–8, April 1995. Guest Editor's Introduction.

[9] Ronald C. Arkin. Reactive robotic systems. In M. Arbib, editor, *Handbook of Brain Theory and Neural Networks*, pages 793–796. MIT Press, 1995.

[10] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proceedings of The IAPR/IEEE Workshop on Visual Behaviors-1994*, pages 112–118, 1994.

[11] W. Ross Ashby. *Design for a Brain.* Chapman and Hall, London, 1952.

[12] W. Ross Ashby. *An Introduction to Cybernetics.* Chapman and Hall, London, 1956.

[13] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[14] R. Bellman and R. Kalaba. Dynamic programming and feedback control. In *Proceedings of 1st IFAC*, pages 460–464, 1960.

[15] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[16] Huges Bersini. Animat's I. In Francisco J. Varela and Paul Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 456–465. MIT Press, December 1991.

[17] V. G. Boltyanski, R. V. Gamkrelidge, E. F. Mishchenko, and L. S. Pontryagin. The maximum principle in the theory of optimal processes of control. In *Proceedings of 1st IFAC*, pages 454–459, 1960.

[18] Ronald J. Brachman and Hector J. Levesque. *Readings in Knowledge Representation.* Morgan Kaufmann, Los Altos, CA, 1985.

[19] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, April 1986.

[20] R. A. Brooks. Autonomous mobile robots. In W. E. L. Grimson and R. S. Patil, editors, *AI in the 1980s and beyond*, pages 343–363, Cambridge, MA, 1987. MIT Press.

[21] R. A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2):253–262, Summer 1989.

[22] R. A. Brooks. The whole iguana. In M. Brady, editor, *Robotics Science*, pages 432–456, Cambridge, MA, 1989. MIT Press.

[23] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1-2):3–16, 1990. Special issue on designing autonomous agents.

[24] R. A. Brooks. Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 569–595, 1991.

[25] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, 1991.

[26] E. Carniak and D. McDermot. *Introduction to Artificial Intelligence*. Series in Computer Science. Addision-Wesley, 1985.

[27] D. Chapman. Planing for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[28] D. Chapman and P. Agre. Abstract reasoning as emergent from concrete activity. In M.P. Georgeff and A.L. Lansky, editors, *Proceedings of the Workshop on Reasoning About Actions and Plans*, pages 411–424, Timberline, Oregon, 1986. Morgan Kaufmann, Los Altos, CA.

[29] D. Chapman and L.P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of The International Joint Conference on Artificial Intelligence, IJCAI-91*, 1991.

[30] V. C. Chen and Y. Pao. Learning control with neural networks. In *IEEE International Conference on Robotics and Automation*, Washington D.C., 1989.

[31] D. T. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):71–108, 1993.

[32] P. R. Cohen and E. A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume I-III. Pitman Books, London, 1982.

[33] Holk Cruse. What mechanisms coordinate leg movement in walking arthropods. *Trends in Neurosciences*, 13(1):15–21, 1990.

[34] H. Curtis. *Biology*. Worth Publishers Inc., New York, second edition edition, 1975.

[35] P. Dayan and T. Sejnowski. Td(λ) converges with probability 1. *Machine Learning*, 14:295–301, 1994.

[36] V. Ruiz de Angulo and C. Torras. On-line learning with minimum degradation in feedforward networks. *IEEE Transactions on Neural Networks*, 6(3), May 1995.

[37] E. Dickmanns and A. Zapp. A curvature-based scheme for improving road vehicle guidance by computer vision. In *Proceedings of the SPIE Conference on Mobile Robots*, 1986.

[38] Rüdiger Dillmann. *Lernende Roboter - Aspekte maschinellen Lernens.* Sringer Verlag, 1988.

[39] Edward A. Feigenbaum and Julian Feldman, editors. *Computers and Thought.* McGraw-Hill, New York, NY, 1963.

[40] I. A. Ferguson. Towards an architecture for adaptive, rational, mobile agents. In E. Werner and Y. Demazeau, editors, *Proceedings of the Third European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91)*, pages 249–262, Amsterdam, 1992. Elsevier Science Publishers.

[41] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.

[42] R.J. Firby. Adaptive execution in complex dynamic worlds. Ph.D. Dissertation YALEU/CSD/RR#672, Yale University, Jan. 1989.

[43] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff et al, editor, *From Animals to Animats: Proceedings of the 3d International Conference on Simulation of Adaptive Behavior*, pages 421–430, Cambridge, MA, 1994. MIT Press.

[44] Dario Floreano. Engineering adaptive behavior. *Adaptive Behavior*, 5(3-4):407–420, Winter/Spring 1997.

[45] K. S. Fu. Learning control systems. In J. T. Ton and R. H. Wilcox, editors, *Computer and Information Sciences*, New York, 1964. Spartan Books.

[46] R. S. Fu. Learning control systems - review and outlook. *IEEE Transactions on Automatic Control*, AC-15, 1970.

[47] Douglas Futuyma. *Evolutionary Biology.* Sinauer, Sunderland, MA, 1986.

[48] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, pages 809–815, San Jose, CA, July 1992.

[49] M. R. Genesereth and N. Nilson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[50] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.

[51] Georges Giralt, Raja Chatila, and Marc Vaisset. An integrated navigation and motion control system for multisensory robots. In Brady and Paul, editors, *Robotics Research 1*, pages 191–214, Cambridge, MA, 1984. MIT Press.

[52] Y. Goto and A. Stentz. Mobile robot navigation: The CMU system. *IEEE Expert*, pages 44–54, Winter 1987.

[53] Karl Georg Götz and Hans Wenking. Visual control of locomotion in the walking fruitfly drosophilia. *Journal of Computational physiology*, 85:235–266, 1973.

[54] J. H. Graham and G. N. Saridis. Linguistic design structures for hierarchical systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12(3), 1982.

[55] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–64, 1987.

[56] M. S. Hamdi. A goal-oriented behavior-based control architecture for autonomous mobile robots allowing learning. In M. Kaiser, editor, *Proceedings of the Fourth European Workshop on Learning Robots*, Karlsruhe, Germany, December 1995.

[57] M. S. Hamdi. Local level arbitration of reactive behaviors and adaptivity to the environment. In V. Klingspor, editor, *Proceedings of the Fifth European Workshop on Learning Robots*, Bari, Italy, July 1996.

[58] M. S. Hamdi and K. Kaiser. Adaptable arbitration of behaviors: Some simulation results. In *Proceedings of The Second World Congress on Intelligent Manufacturing Processes and Systems*, Budapest, Hungary, June 1997.

[59] M. S. Hamdi and K. Kaiser. Adaptable local level arbitration of behaviors. In *Pro-*

ceedings of The First International Conference on Autonomous Agents, Agents'97, Marina del Rey, CA, USA, February 1997.

[60] M. S. Hamdi and K. Kaiser. Self-improving behavior arbitration. In *Foundations of Computer Science Potential - Theory - Cognition*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[61] M. S. Hamdi and K. Kaiser. Adaptable arbitration of behaviors: Some simulation results. *Journal of Intelligent Manufacturing*, 9(2):161–166, April 1998.

[62] M. S. Hamdi and K. Kaiser. Improving behavior arbitration using exploration and dynamic programming. In *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Benicàssim, Castellón, Spain, June 1998.

[63] M. S. Hamdi and K. Kaiser. Learning intelligent behavior. In *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence*, Brisbane, Australia, July 1998.

[64] M. S. Hamdi and K. Kaiser. Learning to coordinate behaviors. In *Proceedings of the 13th biennial European Conference on Artificial Intelligence (ECAI-98)*, Brighton, UK, August 1998.

[65] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[66] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[67] J. Herzberg. Planerstellungs-Methoden der Künstlichen Intelligenz. *Informatik-Spektrum*, 9:149–161, 1986.

[68] R. Hofstetter. Simulation eines Verfolgungsalgorithmus mittels neuronaler Netze. Technical report, Institut für Prozessrechentechnik und Robotik, Universität Karlsruhe, 1990.

[69] M. Holsheimer and A. Siebes. Data mining - the search for knowlege in databases. Report CS-R9406, CWI, University of Amsterdam, 1994.

[70] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, pages 2554–2558, 1982.

[71] Thomas Hoppe. Kriterien zur Auswahl maschineller Lernverfahren. *Informatik-Spektrum*, 19:12–19, 1996.

[72] E. Jantsch. *The Self-Organizing Universe.* Pergamon Press, Oxford, New York, 1980.

[73] Nicholas R. Jennings and Michael J.Wooldridge, editors. *Agent Technology: Foundations, Applications, and Markets.* Springer-Verlag, 1998.

[74] J.L. Jones and A.M. Flynn. *Mobile Robots. Inspiration to Implementation.* Peters, Wellesley, MA, 1993. ISBN 1-568-81011-3.

[75] L. P. Kaelbling and S. J. Rosenschein. A situated view of representation and control. *Artificial Intelligence*, 73:149–173, 1995.

[76] L.P. Kaelbling. An architecture for intelligent reactive systems. In *Proceedings of the 1986 Workshop Reasoning about Actions and Plans*, pages 395–410. Morgan Kaufmann, San Mateo, CA, 1986.

[77] D. Keirsey, J. Mitchell, and D. Payton E. Preyss. Multilevel path planning for autonomous vehicles. *Applications of Artificial Intelligence, Proc. SPIE*, 485:133–137, 1984.

[78] F. Klix. *Information und Verhalten.* VEB Deutscher Verlag der Wissenschaften, Berlin, 1976.

[79] K. Kluge and C. Thorpe. Explicit models for road following. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1989.

[80] E. Koch, C. Yeh, G. Hillel, A. Meystel, and C. Isik. Simulation of path planning for a system with vision and map updating. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 146–160, St. Louis, Mo., 1985.

[81] T. Kohonen. *Self-Organization and Associative Memory.* Springer Series in Information Sciences 8, Heidelberg. Springer Verlag, 1984.

[82] J. R. Krebs and N. B. Davies. *An Introduction to Behavioral Ecology.* Blackwell Scientific Publications, Oxford, 3rd edition edition, 1993.

[83] B. J. A. Kröse and J. W. M. van Dam. Learning to avoid collision: a reinforcement learning paradigm for mobile robot navigation. In *IFAC/IFIP/IMACS Interna-*

tional Symposium on Artificial Intelligence in Real-Time Control, pages 295–300, Delft, 1992.

[84] C. G. Langton. Introduction. In C. G. Langton, editor, *Artificial Life*. Addison-Wesley, 1989.

[85] C. G. Langton, J. D. Farmer, S. Rasmussen, and C. Taylor, editors. *Artificial Life II*. Addison Wesley, 1991.

[86] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):293–321, 1992.

[87] K. Z. Lorenz. *The Foundations of Ethology*. Simon and Schuster, New York, NY, 1981.

[88] Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors. *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*. MIT Press, Cape Cod, MA, September 1996.

[89] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2):311–365, 1992.

[90] Hanspeter A. Mallot. Neuronale Netze. In Günter Görz, editor, *Einführung in die künstliche Intelligenz*. Addison-Wesley, 1995. 2. Aufl., Kap. 8.

[91] D. Marr. Artificial intelligence: A personal view. *Artificial Intelligence*, 9:37–48, 1977.

[92] D. Marr. *Vision*. W. H. Freeman, New York, 1982.

[93] M. Mataric. Learning to behave socially. In D. Cliff et al, editor, *From Animals to Animats : Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, pages 453–462, Cambridge, MA, 1994. MIT Press.

[94] M. Mataric. Reward functions for accelerated learning. In W. W. Cohen and H. Hirsh, editors, *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994.

[95] H. R. Maturana. Biology of cognition. In *Autopoiesis and Cognition: the Realiza-*

tion of the Living. Dordrecht: Reidel, 1970. Published in 1980, the book contains the piece by Maturana (dating from 1970).

[96] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1970.

[97] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[98] D. McFarland. *Animal Behavior*. Benjamin/Cummings, Menlo Park, CA, 1985.

[99] D. McFarland. *Problems of Animal Behavior*. Lognman, Harlow, UK, 1988.

[100] D. McFarland. Autonomy and self-sufficiency in robots. Memo 92-3, VUB AI Lab, Brussels, 1992.

[101] J.-A. Meyer and S. W. Wilson, editors. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press, Cambridge, MA, 1991.

[102] A. Meystel. *Autonomous Mobile Robots. Vehicles with Cognitive Control*. World Scientific Series in Automation 1. Singapore: World Scientific, 1991. ISBN 9971-50-089/2.

[103] J.d.R. Millàn and C. Torras. Efficient reinforcement learning of navigation strategies in an autonomous robot. In *Proceedings of The International Conference on Intelligent Robots and Systems, IROS'94*, 1994.

[104] M. Minsky. *Matter, Minds and Models, Semantic Information Processing*. MIT Press, Cambridge, MA, 1968.

[105] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[106] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of the AAAI*, pages 1051–1058, Boston, MA, 1990.

[107] Tom M. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, and J. Schlimmer. Theo: A framework for self-improving systems. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, 1990.

[108] Hans P. Moravec. The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, 71(7):872–884, 1982.

[109] K. Morik. Maschinelles Lernen. In Günter Görz, editor, *Einführung in die künstliche Intelligenz*. Addison-Wesley, 1995. 2. Aufl., Kap. 3.

[110] J. P. Müller, M. Pischel, and M. Thiel. Modeling reactive behavior in vertically layered agent architectures. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, number 890 in LNAI, pages 261–276, Heidelberg, Germany, January 1995. Springer Verlag.

[111] Ulrich Nehmzow and Tim Smithers. Map-building using self-organizing networks in 'really useful robots'. Technical Report 489, University of Edinburgh, Department of Artificial Intelligence Research, 1990.

[112] A. Newell. Reasoning, problem solving and decision processes: The problem space as a fundamental category. Technical Report 79-133, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1979.

[113] A. Newell, J. C. Show, and H. A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264. UNESCO, 1960.

[114] A. Newell and H. A. Simon. Computer science as empirical enquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19(3):113–126, March 1976.

[115] Nils J. Nilsson. *Learning Machines*. McGraw-Hill, New York, NY, 1965.

[116] Nils J. Nilsson. Shakey the robot. Technical Note 323, SRI AI center, 1984.

[117] Fabrice R. Noreils and Raja G. Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(2), April 1995.

[118] R. Opitz. Das Lernfahrzeug, neural network application for autonomous mobile robots. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 373–379. Elsevier, North Holland, 1990.

[119] D. Payton. An architecture for reflexive autonomous vehicle control. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1838–1845, 1986.

[120] B. A. Pearlmutter. Dynamic recurrent neural networks. Technical Report CMU-CS-90-196, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, December 1990.

[121] G. Pearson and D. Kuan. Mission planning system for an autonomous vehicle. In *Proceedings of the IEEE Second conference on Artificial Intelligence Applications*, pages 162–167, December 1985.

[122] Miles Pebody. Learning and adaptivity: Enhancing reactive behavior architectures in real-world interaction systems. In *Advances in artificial life: Proceedings of the Third European Conference on Artificial Life*, Granada, Spain, June 1995.

[123] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:88–97, 1991.

[124] E. Prem. The behavior-based firm. *Applied Artificial Intelligence*, 11(3):173–195, 1997.

[125] T. Prescott and J. Mayhew. Adaptive local navigation. In A. Blake and A. Yuille, editors, *Active Vision*. MIT Press, 1992.

[126] D. Psaltis, A. Sideris, and A. Yamamura. Neural controllers. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 551–558, 1987.

[127] Zenon W. Pylyshyn, editor. *The Robot's Dilemma*. Ablex Publishing, Norwood, NJ, 1987.

[128] S. K. Reed. *Cognition: Theory and Applications*. Brooks/Cole Publishing Co., Monterey, CA, 1982.

[129] F. Reither. Über die Selbstreflexion beim Problemlösen. Dissertation, Universität Gießen, 1979.

[130] U. Rembold. Autonome mobile Roboter. *Robotersysteme*, 4:17–26, 1988.

[131] E. Rich and K. Knight. *Artificial Intelligence*. Mcgraw-Hill, New York, 2nd edition, 1991. ISBN 0-07-052263-4.

[132] Mark Ridley. *Evolution*. Blackwell Scientific Publications, Boston, 1993.

[133] Mark B. Ring. Continual learning in reinforcement environments. Ph.D. Dissertation, University of Texas at Austin, Austin, Texas 78712, August 1994.

[134] H. Ritter, T. Martinetz, and K. Schulten. *Neuronale Netze. Eine Einführung in die Neuroinformatik selbstorganisierender Netwerke.* Addision–Wesley, 1990. ISBN: 3-89319-172-0.

[135] Larry G. Roberts. Machine perception of three-dimensional solids. Technical Report 315, MIT Lincoln Laboratory, May 1963.

[136] A. Rudström. Applications of machine learning. Report 95-018, Kungl Tekniska Högskolan, Universität Stockholm, Department of Computer and Systems Sciences, 1995.

[137] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Letters to Nature*, 323:533–535, 1986.

[138] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[139] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing.* MIT Press, Cambridge, MA, 1986.

[140] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[141] Earl Sacerdoti. The non-linear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214, Stanford, CA, 1975.

[142] Earl Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, New York, 1977.

[143] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:211–229, July 1959. Also in [39].

[144] G. N. Saridis. *Self-Organizing Control of Stochastic Systems.* Marcel Dekker, New York, 1977.

[145] M. Sekiguchi, S. Nagata, and K. Asakawa. Behavior control for a mobile robot by dual-hierarchical network. In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 122–127, 1989.

[146] R. G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[147] Herbert A. Simon. *Sciences of the Artificial.* MIT Press, Cambridge, MA, 1970.

[148] Herbert A. Simon. Why should machines learn? In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 25–38, Palo Alto, CA, 1983. Tioga.

[149] B. F. Skinner. *Behavior of Organisms.* Appleton-Century-Crofts, New York, 1958.

[150] B. C. Smith. Prologue to "reflection and semantics in a procedural language". In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, Los Altos, CA, 1985. Morgan Kaufmann Publishers.

[151] A. C. Staugaard. *Robotics and AI: an introduction to applied machine intelligence.* Prentice Hall, New Jersey, 1987.

[152] Luc Steels. A case study in the behavior-oriented design of autonomous agents. In D. Cliff et al, editor, *From Animals to Animats: Proceedings of the 3d International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1994. MIT Press.

[153] Luc Steels. When are robots intelligent autonomous agents? *Journal of Robotics and Autonomous Systems*, 15(1/2), 3-9 1995.

[154] J. Stone. Evolutionary robotics: our hands in their brains? In R. Brooks and P. Maes, editors, *Proceedings A-Life IV.* MIT Press, 1994.

[155] Gerald Jay Sussman. *A Computational Model of Skill Acquisition.* American Elsevier, New York, 1975.

[156] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[157] R. S. Sutton. Reinforcement learning architectures for animats. In Jean-Arcady Meyer and Stewart W. Wilson, editors, *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1990. MIT Press.

[158] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction.* MIT Press, March 1998. ISBN 0-262-19398-1.

[159] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning in direct adaptive optimal control. In *Proceedings of the American Control Conference*, Boston, MA, 1991.

[160] A. Tate. A review of knowledge-based planning techniques. In *Proceedings of the 5th Technical Conference of the British Computer Society (Expert Systems 85)*, pages 89–111. Cambridge University Press, 1986.

[161] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, October 1950.

[162] Elpida S. Tzafestas. Implementing reactive algorithms on a cellular control architecture. In *Autonome Mobile Systeme AMS'94, 10. Fachgespräch*, Stuttgart, Germany, 13. und 14. Oktober 1994.

[163] Paul F.M.J. Verschure and Rolf Pfeifer. Categorization, representations and the dynamics of system-environment interaction: A case study in autonomous systems. In *Proceedings of the Second international Conference on the Simulation of Adaptive Behavior*. Bradford books MIT Press, 1993.

[164] W. Grey Walter. An imitation of life. *Scientific American*, 182(5):42–45, May 1950.

[165] W. Grey Walter. A machine that learns. *Scientific American*, 185(2):60–63, August 1951.

[166] W. Grey Walter. *The Living Brain.* Duckworth, London, 1953. Republished by Penguin, Harmondsworth, UK, 1961.

[167] Philip D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold, 1989.

[168] C. J. C. H. Watkins. *Learning from Delayed Rewards.* University of Cambridge, England, 1989. Ph.D. Thesis.

[169] C. J. C. H. Watkins. Q-learning. *Machine Learning*, 8:279–292, 1992.

[170] Rüdiger Wehner. 'matched filters' - neural models of the external world. *Journal of Computational physiology*, A 161:511–531, 1987.

[171] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. Thesis, Harvard University, Committee on Applied Mathematics, 1974.

[172] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[173] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*, pages 96–104, New York, 1960.

[174] Norbert Wiener. *Cybernetics.* John Wiley and Sons, New York, NY, 1948.

[175] Norbert Wiener. *Cybernetics.* MIT Press, Cambridge, MA, second edition edition, 1961.

[176] S. W. Wilson. Knowledge growth in an artificial animal. In J. J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA85)*, pages 16–23. Lawrence Erlbaum Associates, July 1985.

[177] S. W. Wilson. The animat path to AI. In J. A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, pages 15–21. MIT Press, 1991.

[178] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design.* Ablex Publishing Corporation, Norwood, NJ, 1986.

[179] Patrick Henry Winston. The MIT Robot. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 431–463, New York, NY, 1972. John Wiley and Sons.

[180] Patrick Henry Winston. *Artificial intelligence.* Addison-Wesley, Reading, MA, second edition edition, 1984.

[181] S. Winston. The animat path to AI. In *Proceedings of Simulation of Adaptive Behavior - From Animals to Animats*, Paris, September 1990.

[182] Niklaus Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, 1995.

Lebenslauf

Name	Hamdi
Vorname	Mohamed Salah
geboren am	01.04.1967
in	Sidi Bouzid (Tunesien)
Familienstand	verheiratet
Staatsangehörigkeit	tunesisch
1974 - 1979	Grundschule in Faidh (Tunesien) Abschluß: Grundschulabschluß
1980 - 1986	Gymnasium in Sidi Bouzid (Tunesien) Abschluß: Abitur
1987	Studienkolleg in München: Deutschkurs und fachliche Vorbereitung zur Universität
1988 - 1993	Studium der Informatik mit Nebenfach Elektrotechnik an der Technischen Universität München Abschluß: Informatik-Diplom
Seit 1994	Wissenschaftlicher Mitarbeiter am Arbeitsbereich Technische Informatiksysteme (TIS) im Fachbereich Informatik der Universität Hamburg

Erklärung

Hiermit erkläre ich, daß ich diese Dissertation selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 19.1.1999

Mohamed Salah Hamdi