Ultraschallbasierte Navigation und adaptive Hindernisvermeidung eines autonomen mobilen Roboters

Diplomarbeit
am Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Lehrstuhl für kognitive Systeme
Prof. Dr. Ing. Gerald Sommer

vorgelegt von

Ingo Ahrns

Betreuer: Dipl. Inform. Jörg Bruske

1. April 1996

Zusammenfassung

Gegenstand dieser Arbeit ist die ultraschallbasierte Echtzeitnavigation eines autonomen mobilen Roboters (AMR). Der Schwerpunkt dieser Arbeit liegt in einem Teilproblem der Navigation, der sogenannten Hindernisvermeidung. Bei dieser geht es darum, einen AMR kollisionsfrei durch eine unbekannte Umgebung zu lenken.
Für weitere Probleme der Navigation, wie z. B. der Wegplanung, der Kartographierung und der Selbstlokalisation, werden Verfahren angegeben, die in Simulationen getestet wurden.

Ausgangspunkt für die Hindernisvermeidung eines AMR's ist eine von Maja J. Mataric in [Mat92] vorgestellte reaktive verhaltensbasierte Architektur.
Eine Variante dieses Ansatzes wurde für den *LABMATE*, ein autonomes mobiles Roboterfahrzeug der Firma Transitions Research Corporation (TRC) entwickelt und im realen Einsatz getestet. Die hierbei implementierte Hindernisvermeidungsarchitektur besteht aus mehreren hierarchisch angeordneten Modulen, die einzelne Verhaltensweisen darstellen. Das Gesamtverhalten des AMR's erhält man, indem niedriger angeordnete Verhaltensweisen situationsabhängig die höher angeordneten unterdrücken können. Dieser Aufbau entspricht der von [Mat92] verwendeten Subsumtionsarchitektur und unterscheidet sich von der ursprünglich in [Bro86] eingeführten Subsumtion.
Die Schwächen dieses Ansatzes bestehen in einer harten Kontrolle der Lenkbewegungen, die zu Oszillationen im Fahrverhalten führen. Um dieses Problem zu beheben, werden zunächst einzelnen Verhaltensebenen durch einzelne Fuzzy-Regler ersetzt. Dadurch wird eine kontinuierliche Kontrolle der Motorstellgrößen und ein optimiertes Fahrverhalten erreicht.

Die Kodierung von Vorwissen in Fuzzy-Reglern ist nicht immer einfach. Ferner stehen die Fuzzy-Regler nach der Entwurfsphase fest und lassen keine Anpassung des Systems an sich ändernde Randbedingungen zu. Daher besteht ein weiterer Schwerpunkt dieser Arbeit in der Untersuchung lernfähig konzipierter Regelungssyteme zur Hindernisvermeidung eines autonomen mobilen Roboters.
Da dennoch nicht auf das in Fuzzy-Regeln vorkodierte Wissen verzichtet werden soll, besteht die Aufgabe darin, einen lernfähigen Fuzzy-Ansatz zu entwickeln, indem Konzepte der Fuzzy-Regelung und der Adaptivität künstlicher neuronaler Netze in einem Regelungssystem verknüpft werden.

Als Basis solcher Regler werden die sogenannten Dynamischen Zellstrukturen (DCS) verwendet [BS95], die auf Radialen-Basis-Funktions-Netzwerken (RBF) beruhen. Sie werden so erweitert, daß auf einfache Weise Vorwissen in Form von Fuzzy-Regeln in das Netzwerk eingebracht werden kann.
Im letzten Teil dieser Arbeit werden dann zwei Ansätze angegeben, wie ein AMR auf der Basis der zuvor entwickelten neuronalen Regler ein Verhalten zur Hindernisvermeidung selbständig erlernt.

Als Lernverfahren kommen hierzu Reinforcement-Lernansätze zur Anwendung, die speziell auf die entworfenen Regler angepaßt werden. Da es sich hier um kontinuierliche Regler handelt, werden insbesondere das reellwertige stochastische Reinforcement-Lernen [Wil92], [GF94] und eine neue reellwertige Variante des klassischen Q-Lernens (vgl. [Wat89]) behandelt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein autonomer mobiler Roboter (engl. autonomous mobile robot, AMR) ist eine Maschine, die die beiden Grundprinzipien der *Autonomie* und der *Mobilität* in sich vereint.
Unter Autonomie versteht man die Möglichkeit, daß die Maschine *ohne äußere Hilfe* eine ihr gestellte Aufgabe bewältigen kann. Daher sind Fahrzeuge, die auf Schienen fahren oder über Funk ferngesteuert sind, hiernach nicht als autonom anzusehen.
Unter Mobilität versteht man die Möglichkeit, daß sich die Maschine *aus eigener Kraft* bewegen kann. Meistens soll die Bewegung dazu dienen, einen spezifizierten Ort in der Umwelt zu erreichen. Die Anwendungsgebiete solcher autonomen mobilen Roboter sind dabei sehr vielfältig. Ihr Einsatz ist in natürlicher Umgebung ebenso denkbar wie der Einsatz in künstlicher, von Menschen geschaffener Umgebung, wie z. B. in Gebäuden. Insbesondere in künstlichen Umgebungen können AMR zur Bewältigung von Routineaufgaben eingesetzt werden. Möglich ist hier der Einsatz als mobiler Überwachungsroboter, Reinigungsroboter, Transportroboter oder als Reparaturroboter in Umgebungen, die für den Menschen gefährlich sind.
Ebenso umfangreich wie die Einsatzmöglichkeiten autonomer mobiler Roboter sind die Forschungsgebiete, die sich mit dem Einsatz solcher Systeme beschäftigen. Dieses liegt nicht zuletzt daran, daß autonome mobile Roboter im Gegensatz zu klassischen Rechnersystemen nicht in ein statisches System eingebunden sind, sondern sich in einer dynamischen Umwelt zurechtfinden müssen. Die Kenntnis der internen Systemzustände reicht allein nicht aus, um eine gestellte Aufgabe lösen zu können. Vielmehr spielt die Wechselwirkung mit der Umwelt eine große Rolle. Während ein konventionelles Rechnersystem ausschließlich interne Zustandsänderungen erfährt, ist der AMR extern über die Umwelt rückgekoppelt. Das Rechnersystem eines AMR steuert die Motorik des Roboters, wodurch dieser dann eine Bewegung ausführt. Aufgrund dieser Bewegung ändert sich der sensorische Eindruck der Umwelt, worauf eine neue Steuerung des AMR's berechnet wird. Damit stellt sich der Regelkreis aus AMR und dynamischer Umwelt wie in Abbildung 1.1 dar [Hop92]. Zusätzlich können auch externe Umwelteinflüsse den Zustand des Systems verändern. Die extern gestellte Aufgabe beeinflußt schließlich das Verhalten des AMR's.
Der erste AMR wurde im Projekt *Shakey* am Stanford Research Institute unter der Leitung von N. J. Nilsson 1969 entwickelt [Nil69]. Seitdem gibt es eine Vielzahl von AMR-Projekten. Ein ausführlicher Überblick über die historische Entwicklung wird in [Hop92] gegeben.

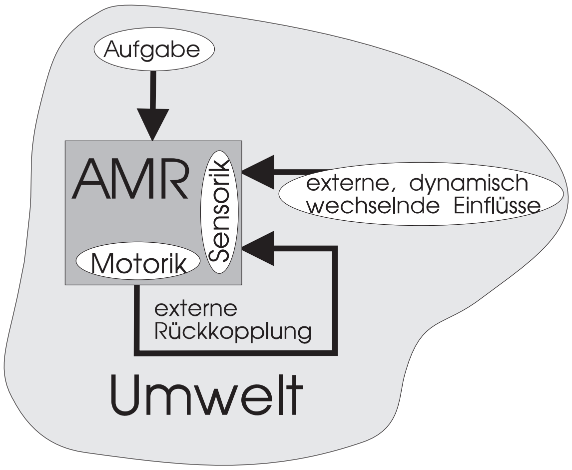


Abbildung 1.1: Regelkreis AMR - Umwelt

Um eine Einordnung dieser Arbeit in das Gebiet der autonomen mobilen Roboter angeben zu können, werden im folgenden Probleme und Aufgabenstellungen beim Entwurf von Navigationssystemen für autonome mobile Roboter angegeben.

1.1 Die Navigationsaufgabe

In [Hop92] wird der Begriff der *Navigation* folgendermaßen definiert:

> Ein *bewegliches Objekt* soll ausgehend von einer *momentanen Position* auf der Basis von teilweiser *unvollständiger Information* unter Berücksichtigung *vorgegebener Randbedingungen* zu einem *vorgegebenen Ziel* gebracht werden.

Als bewegliches Objekt kann das autonome mobile Robotersystem verstanden werden. Es muß auf der Basis von unvollständigen Informationen arbeiten können, da durch die Sensorik immer nur ein sehr beschränkter Ausschnitt der Umwelt erfaßt wird. Als Randbedingungen können weitere Aufgabenstellungen angesehen werden, wie z. B. Anforderungen an das Fahrverhalten oder an die Wahl von Touren.
Daraus ergeben sich zwei wesentliche Teilprobleme der Navigationsaufgabe. Vor Fahrtantritt wird zunächst ein Plan aufgestellt, der angibt, auf welchem Wege ein bestimmtes Ziel erreicht werden kann. Diese Teilaufgabe wird als *Wegplanung* (engl. path planning) bezeichnet. Daran anschließend findet die zweite Phase der Navigation statt, die man als eigentliche *Fahrt* bezeichnet. Hier besteht die Aufgabe, den geplanten Weg in eine physische Bewegung des AMR's umzusetzen.
Im Regelfall wird man von dieser strikten Zweiteilung der Navigation in Wegplanung und Fahrt abweichen müssen, weil der geplante Weg möglicherweise nicht mehr befahrbar ist oder unerwartete Hindernisse sich dem AMR in den Weg stellen. Die Fahrt des AMR's ist

also nicht nur eine Umsetzung der zuvor geplanten Route in eine Bewegung, sondern es handelt sich bei der Fahrt um eine komplizierte Teilaufgabe der Navigation. Diese wird in der Literatur allgemein als *Hindernisvermeidung* (obstacle avoidance) bezeichnet.
Um beide Aufgaben zu lösen, muß eine Schnittstelle zwischen der Umwelt und dem AMR hergestellt werden. Dieses kann über verschiedene Arten von Sensoren vorgenommen und als *Wahrnehmungsfähigkeit* des Roboters aufgefaßt werden.

1. Wahrnehmung des AMR's

Damit der AMR die ihm gestellten Aufgaben erfüllen kann, muß das System Informationen über seine Umwelt aufnehmen können, um daraus Erkenntnisse über den Zustand des Systems zu gewinnen.
Hierzu dient die *Sensorik* des Roboters, die eine Schnittstelle zwischen Umwelt und Robotersteuerung bildet. Die gebräuchlichsten Sensoren für AMR sind *Ultraschallsensoren*, *taktile Sensoren*, *Infrarotsensoren*, *Radargeräte*, *Lasersensoren*, *Kameras*, *Odometrie* und *Kreiselkompaß*. Die ersten sechs Sensorarten dienen der Wahrnehmung der Umwelt und der darin befindlichen Hindernisse. Sie ermöglichen die Durchführung der Fahrt des AMR's und auch die Wegplanung. Die letzten beiden Sensorarten liefern Informationen über die Position und die Orientierung des Roboters (*Selbstlokalisation*). Sie sind insbesondere nötig, um während der Fahrt Informationen über die Beschaffenheit der Umwelt in eine Karte einzutragen.

2. Fahrt des AMR's

Ist das Robotersystem in der Lage, seine Umwelt und seine eigene Stellung darin durch geeignete Sensorik wahrzunehmen, besteht als nächstes die Aufgabe, sich in der Umwelt autonom zu bewegen, ohne dabei die Sicherheit des Systems durch Kollisionen mit Hindernissen zu gefährden. Neben der Bewegung zur Erreichung eines nahen Zielpunktes ist auch eine nicht zielgerichtete Bewegung möglich. Eine solche Bewegung ist dann sinnvoll, wenn eine unbekannte Umgebung erkundet werden soll, und noch keine Informationen über diese Umgebung zur Verfügung stehen (*Exploration*). In beiden Fällen ist die kollisionsfreie Bewegung des Roboters die wesentliche Leistung.
Prinzipiell gibt es zwei Möglichkeiten, die kollisionsvermeidende Fahrt durchzuführen. Die erste Möglichkeit besteht darin, explizites Wissen über die Umwelt auszunutzen, um eine möglichst präzise Bewegungstrajektorie zu berechnen und entlang dieser Trajektorie den AMR zu steuern. Diese Vorausberechnung kann nur über einen zeitlich und räumlich lokalen Bereich erfolgen. Sie muß daher immer wieder von neuem durchgeführt werden. Ferner benötigen diese Ansätze eine genaue Umgebungsrepräsentation, um die Berechnung der Trajektorien vornehmen zu können. Diese Umgebungsrepräsentationen können zumeist aus Bestandteilen der Wegplanung übernommen werden. Bei ausreichender Auflösung der Umgebung durch die Sensorik können auch direkt die gelesenen Sensordaten verwendet werden.
Der zweite Ansatz wird als rein *reaktive Hindernisvermeidung* bezeichnet. Im Gegensatz zum ersten Ansatz wird hier keine explizite Trajektorienberechnung durchgeführt. Vielmehr werden direkte Sensoreindrücke über die Umwelt in direkte Reaktionen des Roboters

umgesetzt. Diese Vorgehensweise zeichnet sich durch den möglichen Verzicht auf eine Umgebungsrepräsentation aus. Die motorische Steuerung des Systems wird aber schwieriger, denn Optimalitätskriterien, wie beispielsweise die Glattheit der zu fahrenden Trajektorien, können im ersten Ansatz direkt in die Berechnungen der Bahnen einfließen, während sie im reaktiven Ansatz nur als Ergebnis einer Vielzahl von nacheinander ausgeführten motorischen Befehlen sichtbar werden.

3. Wegplanung des AMR's

Die Hindernisvermeidung gewährleistet, daß das Robotersystem durch die Hindernisse innerhalb der Umgebung gefahrlos fährt. Für die Lösung einer gestellten Aufgabe, z. B. das Erreichen eines bestimmten Ortes, genügt dieses jedoch nicht. Um aus möglichen Sackgassen entkommen zu können, reichen lokale Umgebungsinformationen, die der AMR durch seine Sensorik vermittelt bekommt, nicht aus. Hierzu müssen globale Informationen über die Umgebung des Roboters verwendet werden.
Bei der *Kartographierung* werden solche globalen Informationen über die Umwelt in einer *Karte* abgespeichert. Das Erstellen einer solchen Karte kann auf zwei Weisen geschehen. Zum einen kann die Karte manuell erstellt werden. In diesem Fall besteht die Karte schon vor Fahrtantritt. Sie kann sofort zur Navigation herangezogen werden. Deshalb spricht man hier auch von der *Navigation in bekannter Umgebung*. Steht die Karte nicht zur Verfügung, muß der AMR die Umgebungsinformationen sammeln und die Karte selbst erstellen. Dabei ist eine ausreichende *Selbstlokalisation* des AMR's nötig, um eine korrekte Karte aufbauen zu können. In diesem Fall spricht man von der *Navigation in unbekannter Umgebung*.
Ist eine Karte vorhanden oder vom AMR selbst erstellt worden, können anhand der Karte verschiedene Wege zur Erreichung eines Ziels bestimmt werden. Des weiteren können auch Optimalitätskriterien die Wahl einer Tour beeinflussen. So ist es möglich, mit einer Karte einen kürzesten Weg zum Zielpunkt zu bestimmen (*Tourenoptimierung*).
Damit sind die wichtigsten Probleme der Navigation angesprochen. Für eine ausführliche Zusammenstellung der klassischen Verfahren zur Navigation sei hier auf [Hop92] oder [Cro87] verwiesen.

1.2 Adaptivität

Die bisher besprochenen Probleme der Navigation eines autonomen mobilen Roboters sind in vielen Projekten bereits gelöst und erfolgreich angewendet worden. Eine zentrale Frage der gegenwärtigen Forschung ist, wie Kontrollarchitekturen für autonome mobile Roboter aussehen können, mit denen ein Verhalten erlernt oder ein gegebenes Verhalten verbessert und an an neue Gegebenheiten angepaßt werden kann. Dabei ist Verhalten die beobachtbare Äußerungsform eines Systems, welche Kompetenz in der Auseinandersetzung mit der Umwelt voraussetzt [Som95]. Zu erlernen ist also eine kompetente Verarbeitung des durch die externe Rückkopplung geschlossenen Wahrnehmungs-Handlungs-Zyklus. Als Bestandteile dieser Verarbeitung kann nicht nur die motorische Kontrolle, sondern auch die Interpretation von sensorischen Eingangsdaten durch lernfähige Ansätze verbessert werden. So werden beispielsweise neuronale Netze in der Interpretation von Ultraschallsensordaten in

[Thr93] verwendet.
Soll ein System wie beispielsweise die Hindernisvermeidungskomponente eines AMR-Navigationsmoduls eine Verhaltensweise erlernen, so stellt sich die Frage nach einem geeigneten Lernverfahren. Wenn das System extern durch einen menschlichen Trainer trainiert werden soll, können alle gängigen überwachten Lernstrategien für neuronale Regler angewandt werden. Schwieriger wird die Lernaufgabe dann, wenn kein Trainer mehr zur Verfügung steht, sondern das System selbst aus eigener Erfahrung lernen muß. Als Ersatz für ein Fehlersignal, das im überwachten Lernen mit einem Trainer durch direkten Vergleich zwischen berechneter und der vom Trainer vorgegebenen Ausgabe gegeben ist, kann ein Verstärkungssignal verwendet werden. Dieses Signal kann als Belohnung oder Bestrafung für die zuletzt berechnete Ausgabe angesehen und zur Adaption des Netzwerkes verwendet werden. Dieses Lernverfahren bezeichnet man auch als *Verstärkungslernen* bzw. mit dem gebräuchlicheren englischen Ausdruck als *Reinforcement-Lernen*. Oft genügt es beim Reinforcement-Lernen nicht, nur die Verstärkungssignale heranzuziehen, die man direkt auf eine berechnete Ausgabe erhält. Es ist vielmehr nötig, zeitlich verzögerte Verstärkungssignale zur Adaption heranzuziehen. Für diesen Fall nennt die Literatur zahlreiche Lernverfahren, die auch im Bereich der Steuerung autonomer mobiler Robotersysteme eingesetzt wurden ([Sut88], [Sut90], [BSA83], [BB93], [Wat92], [RN94], [Wil92]).

1.3 Zielsetzung

Hauptgegenstand der Untersuchungen dieser Arbeit ist die ultraschallbasierte Hindernisvermeidung. Dazu werden in dieser Arbeit verschiedene Ansätze zur Hindernisvermeidung eines AMR's angegeben.
Die erste Art der Hindernisvermeidung orientiert sich an der verhaltensbasierten Hindernisvermeidung in [Mat92]. Sie besteht aus verschiedenen hierarchisch angeordneten Verhaltensweisen, die jeweils für bestimmte Teilaufgaben der Hindernisvermeidung notwendig sind. Eine höhere Verhaltensweise (z. B. das Folgen von Wänden) kommt erst dann zur Anwendung, wenn eine niedrigere Verhaltensweise (z. B. Vermeidung von direkten Kollisionen) dieses zuläßt.
Aufbauend auf diesem Verfahren werden dann neue Architekturen für die hindernisvermeidende Steuerung eines AMR's angegeben, die folgende Grundprinzipien berücksichtigen:

- *Reaktives Verhalten:* Das Verhalten des Systems soll unmittelbar durch die sensorischen Daten und daraus abgeleiteten Merkmalen gesteuert werden. Dieser Ansatz entspricht einer *bottom-up* Vorgehensweise im Gegensatz zu einer *top-down* Vorgehensweise, bei der das Verhalten mehr mittelbar von der Zielstellung des Systems beeinflußt wird. Charakteristisch für ein reaktives Verhalten ist der Verzicht auf eine explizite Repräsentation der Umwelt. Eine Trajektorienberechnung findet daher auch nicht statt.

- *Verhaltensbasierte Hindernisvermeidung:* Die oben bereits angesprochene Gesamtstruktur der Kontrolleinheiten eines AMR's machen das Gesamtverhalten eines Roboters aus. Die darin enthaltenen Komponenten können als Teilverhaltensweisen des

Systems aufgefaßt werden (z. B. Kartographierung, Wegplanung, Hindernisvermeidung). Die Hindernisvermeidung kann ebenfalls in weitere Teilverhaltensweisen aufgespalten werden. Dieses entspricht der in [Mat92] verwendeten verhaltensbasierten Hindernisvermeidung. Es wird untersucht, inwieweit die Trennung der Einzelverhaltensweisen (z. B. Verfolge Ziele, Folge Wänden, passiere Korridore, etc.) auch in einem Fuzzy-Ansatz aufrecht erhalten werden kann.

- *Glatte Steuerung:* Im Ansatz von [Mat92] wird die Motorik durch drei feste Fahrbefehle ausgedrückt (fahre nach links, fahre geradeaus und fahre nach rechts). Hier soll dagegen gezeigt werden, wie es mit Hilfe eines Fuzzy-Ansatzes möglich ist, abrupte Änderungen im Fahrverhalten zu vermeiden und dieses zu optimieren (weitgehende Vermeidung von Oszillationen).

- *Adaptive Steuerung:* Der Ansatz von [Mat92] ist nicht adaptiv, d. h. das Verhalten des AMR's wird zum Zeitpunkt des Entwurfs vollständig festgelegt und ist nicht in der Lage, im Laufe der Lebenszeit zur Verbesserung des Systems verändert zu werden. Derjenige, der das Robotersystem entwirft, muß schon in der Entwurfsphase alle Situationen kennen, in die der AMR möglicherweise kommen wird, um auf diese Situationen den AMR angemessen reagieren zu lassen. Dieses ist schwierig und motiviert den Wunsch nach einem System, das sein Verhalten selbständig verbessert und damit auch mit sich verändernden Randbedingungen zurecht kommt.

1.4 Aufbau der Arbeit

Im zweiten Kapitel wird zunächst auf das dieser Arbeit zugrundeliegende autonome mobile Robotersystem eingegangen. Dabei wird insbesondere der technische Aufbau des hier verwendeten Systems erklärt.
Im dritten Kapitel wird dann ein Verfahren zur ultraschallbasierten Navigation eines realen autonomen mobilen Roboters angegeben. Die Hindernisvermeidung bildet den Schwerpunkt dieses Kapitels und orientiert sich an dem in [Mat92] angegebenen Verfahren. Auf Fragen der Wegplanung, Kartographierung und Selbstlokalisation wird am Ende des Kapitels eingegangen.
Damit ist dann ein verhaltensbasierter Ansatz zur Navigation angegeben, dessen Hindernisvermeidung auf einfachen statischen Verhaltensweisen basiert. Um diese Verhaltensweisen zu einer glatten Steuerung auszubauen, die auch an sich verändernde Randbedingungen anpassungsfähig ist, werden in den folgenden Kapiteln insbesondere auf neuronale und Fuzzy-basierte Regelungssysteme und deren Adaptivität eingegangen.
Im vierten Kapitel werden dazu die Grundlagen durch die Angabe eines neuronalen Reglers gelegt. Dieser neuronale Regler enthält zusätzliche Konzepte von Fuzzy-Reglern, so daß Vorwissen auf einfache Weise durch Fuzzy-Regeln in das Regelungssytem eingebracht werden kann. Das zugrundeliegende neuronale Netzwerk besteht aus Dynamischen Zellstrukturen [BS95], die hinsichtlich der Kodierung von Vorwissen erweitert werden.
Das fünfte Kapitel beinhaltet den ersten Fuzzy-basierten Ansatz zur Hindernisvermeidung eines autonomen mobilen Roboters. Hier werden zwei Architekturen vorgestellt, die auf den im vierten Kapitel entworfenen Reglern beruhen und die verhaltensbasierten Konzepte des

dritten Kapitels berücksichtigen. Dabei wird außerdem auf eine Möglichkeit der Merkmalsextraktion aus Ultraschalldaten eingegangen, die geeignete Merkmale für die Fuzzy-basierte Hindernisvermeidung liefert.
Im sechsten Kapitel wird die im vorherigen Kapitel entwickelte Architektur um die Lernfähigkeit erweitert. Dabei wird insbesondere auf die Probleme des Reinforcement-Lernens eingegangen und wie die zuvor entwickelten Regler an diese Problematik angepaßt werden sollten. Eine Simulation schließt dieses Kapitel ab. Mit ihr wird gezeigt, daß die vorgestellte Architektur in der Lage ist, einen AMR selbständig Verhalten erlernen zu lassen.
Da für die Entwicklung eines Robotersystems Arbeiten und Tests am realen Roboter zeitintensiv sind, wird ein Simulator für den AMR angegeben, der möglichst realitätsnah die Dynamik des Roboters und die Wechselwirkung mit seiner Umwelt simuliert. Insbesondere die im sechsten Kapitel vorgenommenen Untersuchungen zur Adaptivität zwingen dazu, einen Versuch sehr oft nacheinander durchzuführen. Dieses in der Simulation auf einfache Weise möglich. Auf den Simulator wird daher im Anhang A eingegangen.
Theoretische Fragestellungen, die die im vierten Kapitel entworfenen Regler betreffen, werden im Anhang B behandelt.
Auf Einzelheiten der Merkmalsextraktion des fünften Kapitels wird schließlich im Anhang C näher eingegangen.

Kapitel 2

Autonomes Mobiles Robotersystem: TRC LABMATE

Bevor auf die einzelnen Möglichkeiten zur Steuerung eines autonomen mobilen Roboters eingegangen wird, werden in diesem Kapitel zunächst die technischen Gegebenheiten des hier zugrundeliegenden AMR's vorgestellt. Im folgenden wird die Motorik und die Sensorik des Systems dargestellt, sowie die verwendeten Hardware-Komponenten näher erläutert.

2.1 Der TRC LABMATE

Der für diese Arbeit zugrundeliegende autonome mobile Roboter ist der *LABMATE* der Firma *Transitions Research Corporation (TRC)*. Er stellt eine autonome mobile Roboterplattform dar, die für weitere Anwendungszwecke erweitert werden kann.

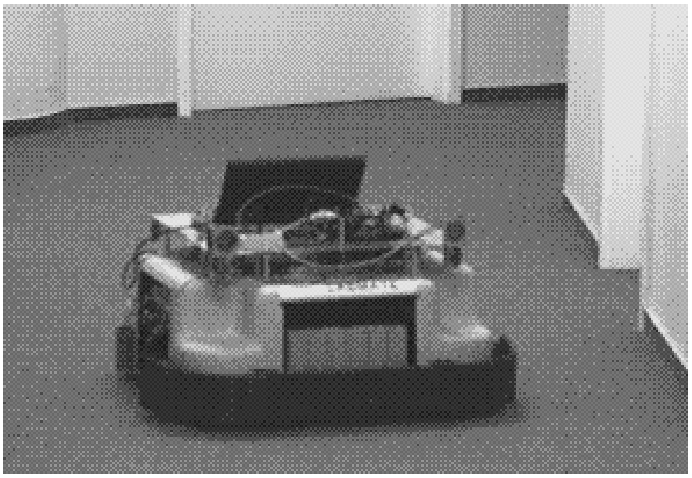


Abbildung 2.1: Das autonome mobile Robotersystem *LABMATE* der Firma *TRC*

2.1.1 Motorik

Der Antrieb erfolgt über zwei servomotorgetriebene Räder, die an einer im Zentrum des Roboters angebrachten Achse befestigt sind. Das darauf befindliche Gehäuse wird durch vier weitere nicht angetrieben Stützräder gehalten. Durch unterschiedliche Drehgeschwindigkeiten der beiden Antriebsräder ist der *LABMATE* damit in der Lage, Kurven beliebiger Radien fahren zu können bis hin zur Drehung auf der Stelle.
Neben einer Ansteuerung der Servomotoren über einen Joystick unterscheidet der *LABMATE* im wesentlichen folgende Betriebsarten:

Go-Modus: In diesem Modus fährt das Fahrzeug mit vorgegebener Geschwindigkeit geradeaus. Die Geschwindigkeit kann als globaler Systemparameter explizit gesetzt werden. Durch Wahl dieses Parameters im Go-Modus wird die Fahrtgeschwindigkeit kontinuierlich geregelt. Die dabei verwendete Beschleunigung bzw. Verzögerung kann explizit festgelegt werden.

Point-to-Point-Modus: In dieser Betriebsart ist es möglich, sowohl gerade Strecken vorgegebener Länge, als auch Kurven vorgegebener Länge und Radien zu fahren. Dieser Modus ist für die reaktive Steuerung eines Navigationssystems nicht praktikabel, da in der Regel die Länge der zu fahrenden Strecke nicht von vornherein feststeht und alle Befehle in dieser Betriebsart mit dem Stillstand des Systems beginnen und enden. Ist das Fahrzeug in Bewegung, so wird es zu Beginn eines Point-to-Point-Befehls abrupt gestoppt. Eine Aufeinanderfolge dieser Befehlstypen kann damit nicht für eine kontinuierlichen Fahrt eingesetzt werden.

Jog-Modus: Diese Betriebsart dient dazu, der momentanen Bewegung des Fahrzeuges im Go-Modus eine rotatorische Bewegung zu überlagern. Diese überlagerte Rotation bleibt solange aufrechterhalten, bis eine neue Rotationsrate (oder auch *Jog-Rate*) gewählt oder in den Go-Modus geschaltet wird. Der Jog-Modus liefert zusammen mit dem Go-Modus eine gute Möglichkeit, die Bewegung des *LABMATE* zu steuern, da sich durch die Überlagerung der beiden Bewegungskomponenten jede beliebige Bewegung vorgeben läßt. Da das Umschalten zwischen Jog- und Go-Modus ohne zwischenzeitliches abruptes Stoppen geschieht, wird diese Art der Ansteuerung der Robotermotoren hier verwendet.

Ferner gibt es weitere Befehle zum Setzen und Abfragen der momentanen Geschwindigkeiten. Für Notsituationen kann ein Notfallstop ausgeführt werden, bei dem das Fahrzeug abrupt stoppt. Für eine ausführliche Beschreibung der Ansteuerung des *LABMATE* sei auf [Tra91a] verwiesen.

2.1.2 Sensorik

Der Hauptbestandteil der Sensorik (vgl. [Tra91b]) besteht aus acht *Ultraschallsensoren*, die in unterschiedlicher Weise auf dem Fahrzeug angebracht werden können. Der Ultraschallkegel, in dem ein einzelner Sensor auf eingebrachte Objekte antwortet, hat einen Öffnungswinkel von ca. 18°. Damit ist schon in geringer Entfernung ein Objekt nur noch

sehr unsicher zu lokalisieren. Weitere Schwierigkeiten bereiten die Ultraschallsensoren hinsichtlich der Schallreflektion an bestimmten Materialien. An glatten Oberflächen, wie z. B. Türen, werden Hindernisse durch die Ultraschallsensorik nur dann wahrgenommen, wenn die Neigung der Achse des Ultraschallkegels nicht zu stark von der Oberflächennormalen des Hindernisses abweicht.
Ferner sollten die Sensoren so am Fahrzeug angebracht und ihre Feuerungsreihenfolge so geartet sein, daß es zu möglichst geringen Störungen der Ultraschallsensoren untereinander kommt. Auf diese Problematik wird im nächsten Kapitel näher eingegangen, wenn die Sensorkonfiguration explizit behandelt wird.
Um das Rauschen der Sensorwerte zu lindern, werden die Sensormessungen über die Zeit geglättet. Dazu wird hier als Glättungsfilter ein gleitender Mittelwert verwendet. Bezeichnet s_t einen Ultraschallmeßwert zum Zeitpunkt t, dann wird durch $\bar{s}_t$ ein geglätteter Meßwert zum Zeitpunkt t angegeben. Die hier verwendete Glättung ist gegeben durch

$$\begin{aligned} \bar{s}_0 &= s_0 \\ \bar{s}_t &= \gamma_{sensor} \cdot s_t + (1 - \gamma_{sensor}) \cdot \bar{s}_{t-1}, \quad \gamma_{sensor} \in]0, 1[. \end{aligned} \tag{2.1}$$

Durch geeignete Wahl von γ_{sensor} werden die Rauscheffekte der Sensoren verringert. Allerdings werden bei zu starker Glättung die Sensormessungen zeitlich verzögert, so daß auch die Reaktionszeit des Systems geringer wird.
Die eingebaute Odometrie ermöglicht es dem Roboter, anhand der gefahrenen Wege der beiden Antriebsräder auf die Position und die Orientierung des Fahrzeuges in seiner Umwelt zu schließen. Dabei verwendet der *LABMATE* ein in Abbildung 2.2 dargestelltes *allozentrisches Koordinatensystem*, um seine Position (x, y) und Orientierung α auf die Umwelt bezogen anzugeben. Der Ursprung des allozentrischen Koordinatensystems wird zum Zeitpunkt des Einschaltens des Fahrzeuges festgelegt. Im Gegensatz dazu bezeichnet man als *egozentrisches Koordinatensystem* das Koordinatensystem, das seinen Ursprung im Fahrzeugzentrum besitzt und immer in Fahrtrichtung ausgerichtet ist. Die relative Lage von Hindernissen zum Fahrzeug wird in egozentrischen Koordinaten angegeben.

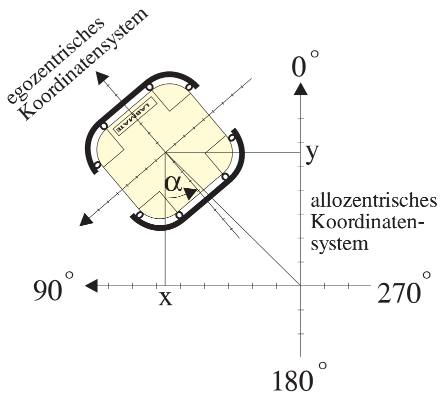


Abbildung 2.2: Allozentrisches und egozentrisches Koordinatensystem des *LABMATE*

Die allozentrischen Koordinaten, die durch die Odometrie geliefert werden, sind allerdings nur bedingt verwertbar, da sie einem starken Meßfehler unterworfen sind. Auf die Pro-

bleme die mit der Odometrie zusammenhängen, wird im dritten Kapitel im Rahmen der Kartographierung genauer eingegangen.
Für den äußersten Notfall, nämlich der Kollision mit einem Hindernis, ist der *LABMATE* mit zwei taktilen Sensoren ausgestattet, die als vordere und hintere Stoßstange an das Fahrzeug angebracht sind. Eine Berührung dieser Stoßstangen führt zu einem Hardware-Reset und dem sofortigen Stop des Fahrzeuges. Leider ist die Handhabung dieser taktilen Sensoren schwer, da ihre Empfindlichkeit schlecht einzustellen ist. In den meisten realen Experimenten mußten die Stoßstangen entfernt werden, weil schon geringe Erschütterungen des Fahrzeuges während der Fahrt einen Stoßstangenkontakt herstellten. In diesem Fall ist unbedingt darauf zu achten, daß durch die Ultraschallsensoren ein Minimalabstand zu Hindernissen überprüft wird. Bei Unterschreitung dieses Minimalabstandes muß als Ersatz für einen Stoßstangenkontakt eine Notfallsituation angenommen werden und das Fahrzeug sofort gestoppt werden.
Außer durch einen Stoßstangenkontakt wird ein Notfallstop des *LABMATE* durch einen Ausfall der seriellen Datenverbindung zwischen Steuerungselektronik und Rechner hervorgerufen. Dazu gibt es einen Zeitzähler (*watchdog-timer*), der die Zeit zwischen zwei aufeinanderfolgenden über die serielle Verbindung eintreffenden Daten mißt. Überschreitet diese Zeitdauer einen einstellbaren Schwellwert, so wird das System sofort angehalten. Diese Zeitüberwachung ist unbedingt zu aktivieren, da sie im Falle eines Ausfalls des Steuerungs-PC's oder der seriellen Verbindung die Sicherheit des Roboters gewährleistet. In den für diese Arbeit durchgeführten Experimenten wurde eine Zeitdauer von 400 ms gewählt.

2.1.3 Abmessungen

Der Grundriß des *LABMATE* ist nahezu quadratisch, was die Navigation erschwert. Durch Wahrung eines Sicherheitsabstandes zu den Hindernissen kann dennoch für die Navigation eine Radialsymmetrie des Roboters angenommen werden. Bei einer Seitenlänge des Fahrzeuges von ca. 63 cm muß damit aber ein Umfang von $\sqrt{2} \cdot 63\ cm \approx 89\ cm$ angenommen werden. Dieser relativ große Umfang bereitet insbesondere Schwierigkeiten beim Passieren von Türen, die nur geringfügig breiter sind ($\approx\ 1\ m$).

2.1.4 Steuerungs-Hardware

Ursprünglich sollte der *LABMATE* über Funkmodems mit einer leistungsfähigen SUN-Workstation verbunden und durch sie gesteuert werden. Probleme mit der Funkverbindung führten schließlich dazu, daß die gesamte Navigation des AMR durch einen 486 SX (25 MHz) Laptop bewältigt werden mußte. Zusätzliche Schwierigkeiten traten bei der gleichzeitigen Inbetriebnahme des *LABMATE* und der Ultraschallsensorik auf. Nachweislich stören die Servomotoren des Roboters die Elektronik der Sensorik. Erst eine galvanische Trennung der Sensorik und der Motorkontrolle über einen Optokoppler konnte dieses Problem lösen. Die Hardware-Konfiguration des *LABMATE*, mit der die in den folgenden Kapiteln beschriebenen Experimente durchgeführt wurden, ist in Abbildung 2.3 dargestellt.
Für die adaptiven Steuerungsansätze genügt die für diese Arbeit zur Verfügung stehende Rechenleistung des Laptops nicht mehr aus, so daß diese Ansätze hier nur im Simulator getestet werden konnten.

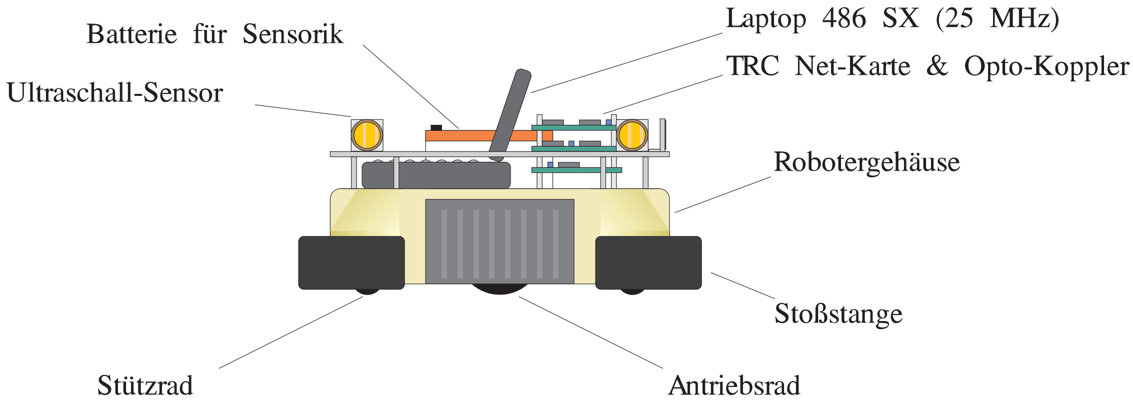


Abbildung 2.3: Hardware-Konfiguration des *LABMATE*

2.2 Der Simulator für den TRC LABMATE

Um die Möglichkeit zu haben, Ansätze zur Steuerung eines AMR's nicht ausschließlich durch Experimente am realen Fahrzeug testen zu müssen, wurde ein Simulator für den *LABMATE* entwickelt. Meistens werden Experimente, die schon im Simulator scheitern, in der Realität erst recht scheitern, da Simulationen fast immer von vereinfachten Modellannahmen der Realität ausgehen. Daher ist es sinnvoll, einen Lösungsansatz erst dann am realen System zu testen, wenn er sich in der Simulation bewährt hat. Diese Vorgehensweise spart Entwicklungszeit und hat sich auch in dieser Arbeit beim Entwurf der Hindernisvermeidung für den hier verwendeten AMR bewährt.
Des weiteren können insbesondere die Verfahren zur adaptiven Steuerung eines AMR's optimal in Simulationen getestet werden. Nur mit Hilfe von Simulationen ist es möglich, Tests unter gleichen Bedingungen ablaufen zu lassen, und die daraus gewonnen statistischen Daten zu vergleichen.
In der Literatur ist die Simulation ein gängiges Verfahren, um Kontrollprobleme und auch Lernansätze zu testen (z. B. [PM92], [RN94], u.a.). Dennoch weisen Kritiker auf die Unverzichtbarkeit von realen Experimente hin. In [Neh92] wird diese Kritik darauf gestützt, daß die reale Welt eine viel größere Komplexität aufweist, als in Simulationen jemals nachgeahmt werden kann.
Dennoch reicht für viele Aussagen die Komplexität eines möglichst realitätsnahen Simulators aus. So weisen z. B. Experimente mit dem realen Roboter beim Passieren einer Tür die gleichen Eigenheiten wie im Simulator auf. Im simulierten wie im realen Fall entstanden durch ein offenstehendes Türblatt die gleichen Probleme.
Ferner wird in [Neh92] angegeben, daß die stochastischen Prozesse der realen Welt helfen können, gewisse Aufgaben zu bewältigen. Ein Fahrzeug mit zwei Fühlern (vgl. [Bra93]) war bei [Neh92] in der Lage, alleine aus einem von Wänden abstoßenden Verhalten heraus sich aus einer Sackgasse zu befreien.
Aber selbst dieses Ergebnis läßt sich auch in einer realitätsnahen Simulation beobachten. Überlagert man den Sensoren und der Motorik ein geeignetes Rauschsignal, so ist der Simulator ebenfalls in der Lage, zufällige Beobachtungen zu liefern.

Da der Entwurf des Simulators dennoch nur als Hilfsmittel gedacht ist und nicht den Schwerpunkt dieser Arbeit darstellt, wird der Simulator an dieser Stelle nicht weiter vertieft. Eine Beschreibung und Hinweise zur Implementation sind in Anhang A gegeben.

Kapitel 3

Ultraschallbasierte Navigation eines autonomen mobilen Roboters

In diesem Kapitel wird die allgemeine Betrachtung über die Navigation von autonomen mobilen Robotern beendet und ein konkretes Verfahren angegeben. Im Zentrum der Untersuchungen steht dabei die Hindernisvermeidung, also die Realisierung der zweiten Teilaufgabe der Navigation, der Fahrt. Die erste Teilaufgabe der Navigation besteht aus der Wegplanung und der Kartographierung und wird in den letzten Abschnitten dieses Kapitels anhand eines einfachen Ansatzes diskutiert. Ein dabei auftretendes schweres Problem ist die Selbstlokalisation.

3.1 Verhaltensbasierte Hindernisvermeidung

Im Projekt *Shakey* [Nil69] ist die verwendete Kontrollstruktur eine sogenannte *horizontale Kontrollstruktur* [Hop92], in der einzelne Komponenten sequentiell angeordnet sind. Die Aufgaben werden Schritt für Schritt abgearbeitet, so daß der Ausfall einer Komponente zum Gesamtausfall des Systems führt.
Dagegen hat Rodney A. Brooks 1986 ein *vertikales Kontrollprinzip* angegeben [Bro86]. In dieser Kontrollstruktur werden alle Komponenten als Verhalten des Systems angesehen und in hierarchisch angeordnete Ebenen unterteilt. Jede Ebene repräsentiert eine bestimmte Verhaltensweise des Systems, die ein eigenes Ziel verfolgt. Jede Verhaltensebene ist autonom und enthält die Verhaltensweisen tieferer Schichten als Teilverhalten, auf denen es aufbauen kann. Hierarchisch tiefer angeordnete Verhaltensweisen sind von höheren Verhaltensweisen unabhängig. Sind die einzelnen Verhaltensweisen als parallel arbeitende Prozesse in der Kontrollstruktur implementiert, so führt der Ausfall einer Verhaltensebene nicht zum Gesamtausfall des Systems, sondern nur zum Verlust bestimmter Teilkompetenzen. Alle Verhaltensweisen unterhalb der ausgefallenen Verhaltensebene können weiterarbeiten.

Jede Verhaltensebene hat in der Regel zugriff auf alle Sensordaten und kann direkt auf die Motorik des Roboters Einfluß nehmen. Höhere Verhaltensebenen können weiter auf die berechneten Ergebnisse niedrigerer Verhaltensebenen zugreifen. Die tiefer angeordneten Verhaltensweisen stellen grundlegendere Verhaltensweisen dar. Sie sind für die Sicherheit

des Systems von größerer Wichtigkeit, während die höheren Ebenen auch komplexere Verhaltensweisen repräsentieren.
Die niedrigeren Verhaltensebenen müssen daher in der Lage sein, sich den höheren Verhaltensebenen unterzuordnen. Höhere Verhaltensebenen können also niedrigere unterdrücken und deren Steuerbefehle durch eigene überschreiben. Diese Kontrollstruktur wird in [Bro86] als *Subsumtionsarchitektur* bezeichnet.
Prinzipiell kann diese Architektur zur Lösung der gesamten Navigationsaufgabe von der Hindernisvermeidung bis hin zur Wegplanung verwendet werden. Hier soll im folgenden jedoch lediglich die Hindernisvermeidung betrachtet werden, dessen Aufteilung in Verhaltensebenen durch [Mat92] motiviert ist.
Der Subsumtionsbegriff in der Hindernisvermeidung von [Mat92] unterscheidet sich von dem, der von Brooks eingeführt wurde. Bei der in [Mat92] verwendeten Architektur unterdrücken niedrigere Verhaltensweisen die höheren. Sicherheitskritische Verhaltensweisen geben die Kontrolle erst dann an höhere und damit oft zielgerichtete Verhaltensebenen ab, wenn die Situation dieses zuläßt. Einer höheren Verhaltensebene werden also durch niedrigere Verhaltensebenen Kompetenzen abgenommen. Zusammen mit allen unteren Verhaltensweisen stellt eine höhere Ebene ein Verhalten dar, das die unteren Verhaltensweisen als Teilverhalten enthält. Daher kann auch die Kontrollarchitektur in [Mat92] als eine Subsumtionsarchitektur aufgefaßt werden.

3.1.1 Zerlegung der Hindernisvermeidung in Verhaltensebenen

Eigentliches Ziel der Hindernisvermeidungskomponente in [Mat92] ist das kollisionsfreie Folgen von Wänden, das zur Exploration der Umwelt verwendet wird. Hier wird die Aufgabe um das Verhalten der Zielverfolgung ergänzt, die ähnlich wie das Wandfolgen als höchste Verhaltensebene der Hindernisvermeidung anzusehen ist.
In [Mat92] wird das Gesamtverhalten der Hindernisvermeidung auf folgende Verhaltensebenen verteilt:

Zielloses Umherfahren: Die unterste Verhaltensebene ist lediglich in der Lage, geradeaus zu fahren. Wird ein nahes frontales Hindernis entdeckt, wird das Fahrzeug gestoppt und zurückgesetzt. Sonst fährt der AMR mit konstanter Geschwindigkeit ($0.2\ \frac{m}{s}$) geradeaus.

Kollisionsvermeidung: Wird ein weiter entferntes frontales Hindernis entdeckt, wird in die Richtung ausgewichen, in der kein frontales Hindernis durch die Sensoren erkannt wird. Zeigen die Sensoren auf der gesamten Front Hindernisse an, so wird standardmäßig nach links ausgewichen. Die untersten beiden Verhaltensebenen gewährleisten ein kollisionsfreies Umherfahren des AMR's.

Ausrichtung an Wänden: Zeigen die Sensoren an, daß sich nur an einer hinteren Seite des Fahrzeuges ein Hindernis befindet, wird dieses Hindernis als eine Wand gedeutet, von der das Fahrzeug sich entfernt. Um eine Ausrichtung an dieser Wand vorzunehmen, wird in Richtung der Wand eine Drehung durchgeführt. Damit ermöglichen die drei untersten Verhaltensweisen das Folgen von Wänden, die eine konvexe Begrenzung darstellen.

Korrektur: Die oberste Ebene verhindert, daß das Fahrzeug eine verfolgte Wand an einer scharfen Ecke in der Umgebung verliert. Dazu wird im Falle einer durch die Sensoren detektierten Ecke eine Drehung in Richtung dieser Ecke vorgenommen.

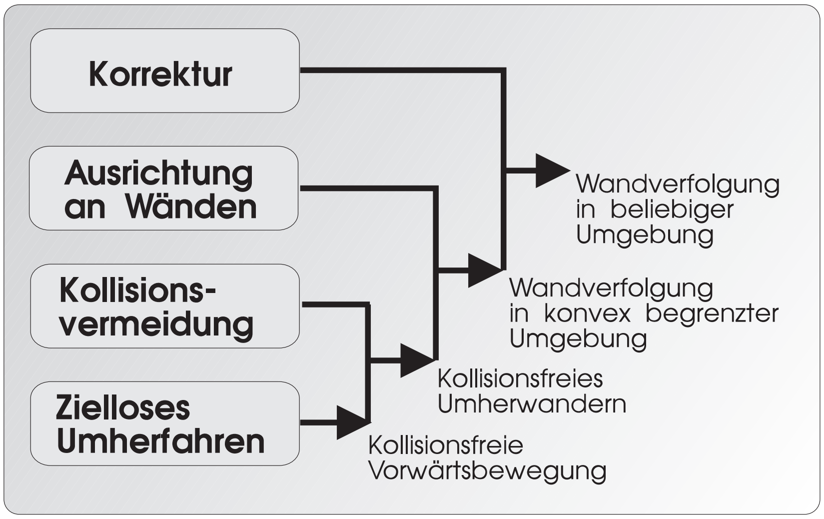


Abbildung 3.1: Verhaltensebenen im Mataric-Ansatz

Die Abbildung 3.1 veranschaulicht die von Maja Mataric [Mat92] durchgeführte Aufteilung des Gesamtverhaltens in vier Verhaltensebenen. Dabei wird lediglich eine sehr begrenzte Anzahl von Aktionen verwendet. Das Fahrzeug bewegt sich mit konstanter Geschwindigkeit, d. h. eine Regelung der Geschwindigkeit entfällt vollständig. Für die Drehungen werden nur zwei Bewegungskommandos benutzt, nämlich eine Drehung nach links und eine Drehung nach rechts. Dazu wird das Fahrzeug *gestoppt* und jeweils um 30° nach links oder rechts gedreht.
Der für die Versuche von Mataric verwendete AMR hat einen Durchmesser von nur 30 cm. Dieses erleichtert insbesondere die Navigation dieses Fahrzeuges in schmalen Gängen und durch Türen. Das Fahrzeug ist mit 12 Ultraschallsensoren ausgerüstet, die radialsymmetrisch um das Fahrzeug angeordnet sind. Jeder einzelne Sensor deckt dabei einen Bereich von 30° ab. Zur Unterscheidung von nahen und fernen frontalen Hindernissen wird der Sensorbereich der nach vorn gerichteten Sensoren in zwei Zonen eingeteilt.
Die Zone im Frontbereich bis zu einer Entfernung von 30 cm wird als *Gefahrenzone* bezeichnet und löst insbesondere eine Reaktion der untersten Verhaltensebene aus. Die Zone im Frontbereich in einer Entfernung von 30 cm bis 60 cm wird als *Sicherheitszone* bezeichnet. Hindernisse, die in diese Zone eindringen, werden als ferne frontale Hindernisse angesehen und lösen eine Reaktion des Kollisionsvermeidungsverhalten aus. Auf ähnliche Weise werden bei Mataric seitliche Hindernisse als solche angesehen, die seitlich bis auf eine bestimmte Entfernung (80 cm) an das Fahrzeug herankommen. Eine feinere Sensordateninterpretation findet bei Mataric nicht statt.

3.1.2 Ein verhaltensbasierter Ansatz für den LABMATE

Ausgehend von dem oben beschriebenen verhaltensbasierten Ansatz zur Hindernisvermeidung eines autonomen mobilen Roboters, soll in diesem Abschnitt eine verhaltensbasierte Hindernisvermeidung für den *LABMATE* angegeben werden. Da hier die technischen Voraussetzungen andere als bei Mataric sind, können die Verhaltensebenen nicht direkt übernommen werden.
Außerdem soll die Aufgabenstellung erweitert werden. Neben dem Folgen von Wänden wird auch eine Verfolgung von vorgegebenen Zielpunkten ermöglicht. Die Steuerung des Fahrzeuges soll möglichst ohne Zwischenstops durchgeführt werden. Im Gegensatz zum obigen Ansatz sollen also echte Kurven gefahren werden und nicht nur Punktdrehungen im Stillstand des Fahrzeuges ausgeführt werden. Dieses führt in einigen Umgebungen zu starken Oszillationen, wenn man direkt die von Mataric vorgeschlagene Implementation der Verhaltensweisen übernimmt.
Der *LABMATE* ist mit acht Ultraschallsensoren ausgerüstet, die in verschiedener Weise am Fahrzeug angebracht werden können.

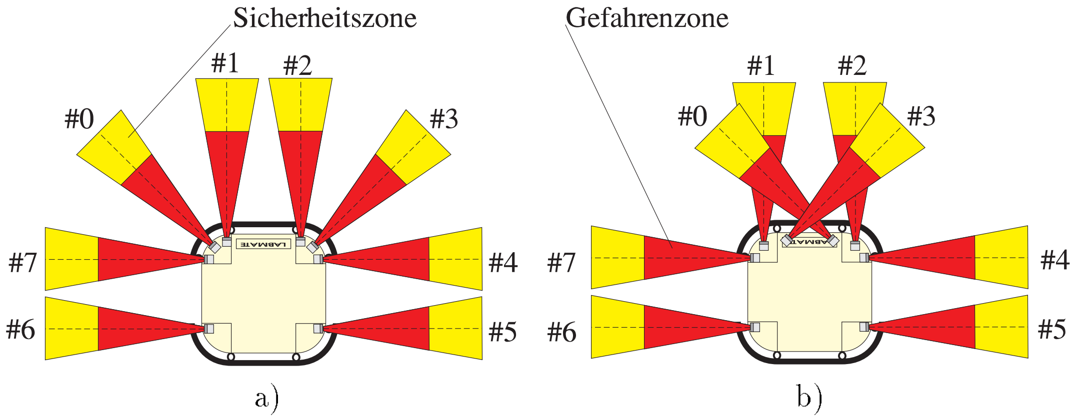


Abbildung 3.2: Sensorkonfigurationen

Zwei mögliche Sensorkonfigurationen mit acht Sensoren sind in Abbildung 3.2 dargestellt. In beiden dargestellten Konfigurationen decken vier Sensoren den Frontbereich ab, da Bewegungen nach vorne gerichtet sind, und hier die Kollisionsgefahr am größten ist. An beiden Seiten werden mindestens zwei Sensoren benötigt, um die Neigung des Fahrzeugs zu einer Wand zu ermitteln. Da für die Experimente im Rahmen dieser Arbeit nicht mehr Sensoren zur Verfügung standen, wurde auf hintere Sensoren verzichtet. Dieses ist insofern unkritisch, als daß keine Rückwärtsbewegung[1] durchgeführt werden muß. In Abbildung 3.2 a) sind die Sensoren so angeordnet, daß eine möglichst gute Überdeckung der Umgebung durch die Ultraschallkegel erreicht wird. Problematisch ist der nahe Frontbereich des Fahrzeuges, der durch die Ultraschallkegel nicht überdeckt wird. Eine vollständige Über-

[1]Eine Ausnahme davon bildet ein Zurücksetzen des Fahrzeuges, nachdem es zu einer nicht vermiedenen Kollision gekommen ist. Allerdings wird hier nur soweit zurückgesetzt, daß eine Punktdrehung ermöglicht wird (z. B. 26 *cm* zur Wiederherstellung der angenommenen Radialsymmetrie).

deckung des nahen Frontbereichs ist dagegen mit den überkreuzenden Ultraschallkegeln aus Abbildung 3.2 b) möglich. Hier ist allerdings die nahe Umgebung der vorderen linken und rechten Seite des Fahrzeuges nicht überdeckt. Diese Lücken führen insbesondere beim Durchfahren einer Tür zu Schwierigkeiten, da Türrahmen schlecht mit dieser Sensorkonfiguration erkannt werden können. Daher werden für alle folgende Experimente die Konfigurationen aus Abbildung 3.2 a) verwendet.
Ähnlich wie bei [Mat92] werden die Meßwerte der Ultraschallsensoren in zwei Zonen eingeteilt. Die erste Zone deckt den Nahbereich eines Sensors ab und wird als *Gefahrenzone* bezeichnet. Die zweite Zone deckt die weitere Umgebung des Sensors ab und wird als *Sicherheitszone* bezeichnet. Die Reichweite der jeweiligen Zone wird für jeden Sensor separat eingestellt. Wie bei Mataric hängt das Verhalten der einzelnen Ebenen der Hindernisvermeidung hauptsächlich von den Zugehörigkeiten der gemessenen Sensorwerte zu den jeweiligen Sensorzonen ab. Das hat einerseits den Vorteil, daß Rauscheffekte in der Sensorik, die sich innerhalb einer Zone befinden, nicht auf die Steuerung auswirken können. Andererseits hat es den Nachteil, daß sich diese Rauscheffekte im Übergangsbereich der verschiedenen Zonen umso stärker bemerkbar machen. Schwankungen in den Sensormeßwerten im Bereich des Zonenüberganges führen zu einem abrupten Wechsel der Verhaltensweise.
Trotz dieser Nachteile liegt der entscheidende Vorteil in der einfachen Handhabung der Sensorik durch Einteilung in Zonen und dadurch in der einfachen Kodierbarkeit der einzelnen Verhaltensweisen.

Verhaltensebenen der Hindernisvermeidung

Im Prinzip werden ähnliche Verhaltensebenen zur Steuerung des *LABMATE* benötigt, wie sie auch von Mataric angegeben wurden. Während der vielen Experimente und damit einhergehenden Optimierungen ist eine verhaltensbasierte Architektur für die Hindernisvermeidung entstanden, wie sie in Abbildung 3.3 dargestellt ist. Zunächst wird eine grobe Erläuterung zu den einzelnen Verhaltensebenen gegeben, bevor im Anschluß auf die algorithmische Umsetzung eingegangen wird.

Entkommen aus einer Gefahrensituation: Eigentlich sollte es vermieden werden, daß das Fahrzeug in eine Gefahrensituation kommt, indem es mit einem Hindernis kollidiert. Aufgrund der dynamischen Umgebung kann es aber nicht ausgeschlossen werden, daß es doch zu einer Kollision kommt. Übersehene Hindernisse aufgrund einer nicht vollständiger Überdeckung der Umgebung durch die Ultraschallkegel oder sehr schnell bewegte Hindernisse können eine Kollision und damit einen Kontakt der taktilen Sensoren hervorrufen. Die unterste Verhaltensebene reagiert darauf mit sofortigem Stop des Fahrzeuges. Ist es einmal zur Kollision gekommen, so kann aufgrund der fehlenden Radialsymmetrie des *LABMATE* nicht einfach eine Drehung durchgeführt werden. Es kann auch keine längere Rückwärtsfahrt eingeleitet werden, da bei Verwendung der Sensorkonfigurationen aus Abbildung 3.2 hinten keine Sensoren angebracht sind. Es muß daher ein kurzes Stück *blind* zurückgesetzt werden, um einen Freiraum um das Fahrzeug herum herzustellen, der eine Punktdrehung erlaubt. Das blinde Zurücksetzen auf kurzer Strecke (ca. 26 cm - 30 cm) kann gefahrlos durchgeführt werden, da das Fahrzeug zuvor über diesen Weg gekommen sein muß.

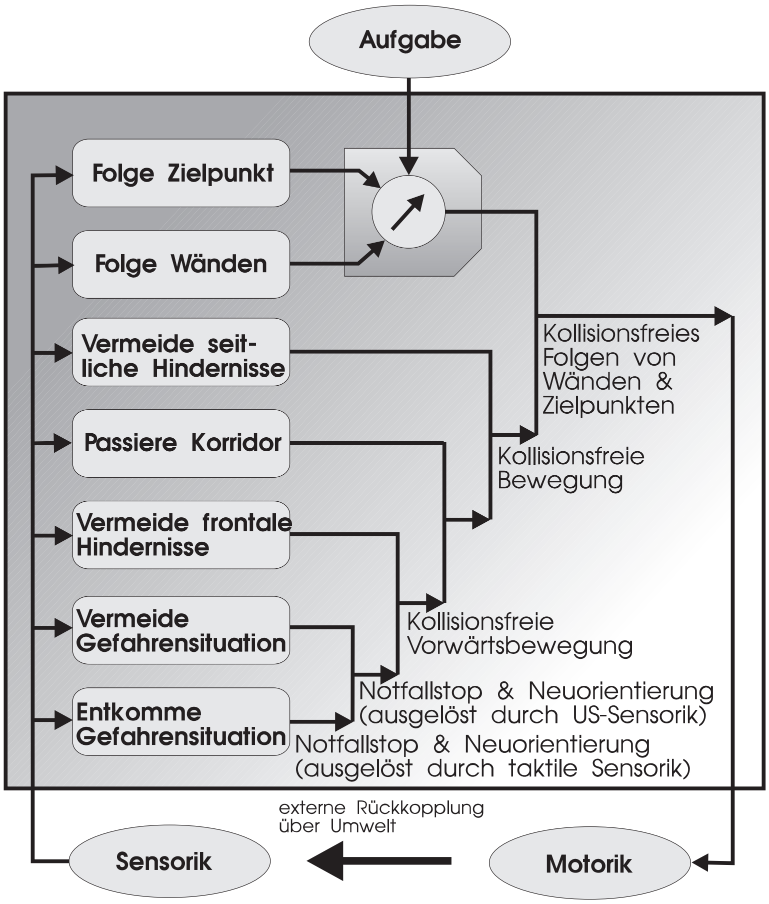


Abbildung 3.3: Verhaltensbasierte Hindernisvermeidungsarchitektur zur Steuerung des *LABMATE*

Nach dem Rücksetzmanöver wird eine *Orientierungsphase* eingeleitet, bei der das Fahrzeug eine Punktdrehung durchführt, bis die Frontsensoren (Sensoren #1 und #2) einen freien Weg anzeigen.

Damit gewährleistet die unterste Verhaltensebene, daß sich das Fahrzeug aus allen Gefahrensituationen, bei denen es zur Kollision gekommen ist, aus eigener Kraft befreien kann.

Vermeidung von Gefahrensituationen: Dieses Verhalten unterscheidet sich nur geringfügig von der untersten Ebene. Es wird hier versucht, gefährliche Situationen nur aufgrund der Sensordaten zu vermeiden. Als gefährliche Situationen sind hier solche zu verstehen, bei denen mindestens einer der Sensoren #0 bis #3 einen Wert innerhalb der Gefahrenzone liefert. Die Reaktion auf diese Situationen ist die gleiche wie in der untersten Ebene. Erst wird gestoppt, zurückgesetzt und dann eine Orientierung eingeleitet. Diese Verhaltensebene gewährleistet zusammen mit der untersten Ebene, daß in Notfallsituationen, die sowohl durch Kollisionen, als auch durch sehr geringe

Entfernungen zu Hindernissen verursacht werden, sich das Fahrzeug durch Neuorientierung aus diesen Situationen befreit. Die untersten beiden Ebenen entsprechen der untersten Ebene im Mataric-Ansatz.

Vermeidung frontaler Hindernisse: Diese Ebene entspricht im Mataric-Ansatz der Ebene der Kollisionsvermeidung. Wird die Sicherheitszone der Sensoren #1 und #2 betreten, so wird in die Richtung ausgewichen, dessen Seitenabstand[2] größer ist. Sind linker und rechter Seitenabstand gleich groß, so wird standardmäßig nach links ausgewichen.

Wird dagegen nur die Sicherheitszone eines der Sensoren #1 und #2 betreten, so wird in die dem jeweiligen Sensor gegenüberliegende Richtung ausgewichen.

Durch die Abhängigkeit der Ausweichrichtung von den Seitensensoren wird vermieden, daß das Fahrzeug in Ecken ausweicht und dann zum Stillstand gezwungen wird. Dieses ist eine Folge des Ausweichens in eine standardmäßige Richtung, wie Mataric es vorschlägt (vgl. auch Abb. 3.4). Damit gewährleisten die untersten drei Ebenen ein kollisionsfreies Umherfahren des AMR's.

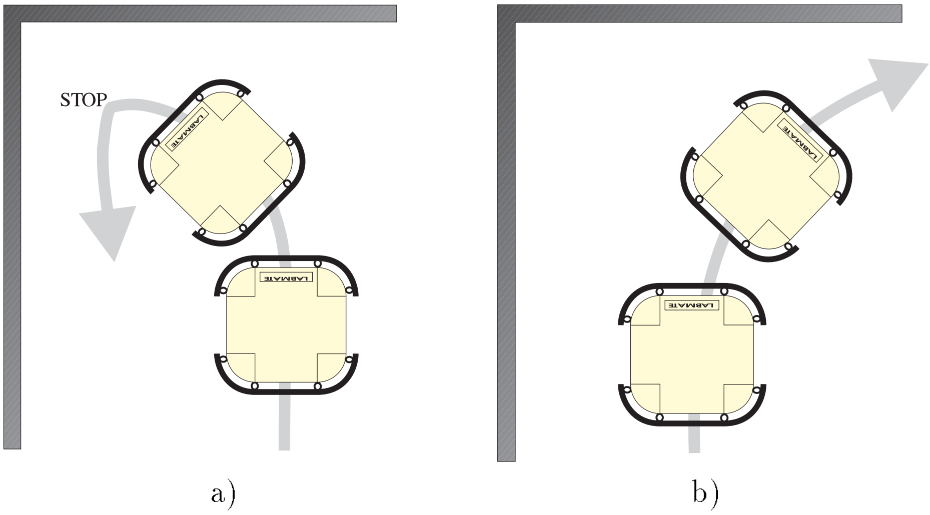


Abbildung 3.4: a) Standarmäßiges Ausweichen nach links in Ecken (Mataric),
b) Seitenfreiraumabhängiges Ausweichen

Die bisherigen Verhaltensweisen dienen nur zur Vermeidung frontaler Hindernisse. Um zusätzlich zu verhindern, daß das Fahrzeug seitlich mit einer Wand kollidiert, werden zwei weitere Verhaltensweisen eingeführt, die seitliche Kollisionen mit Hindernissen vermeiden. Die nächste Verhaltensweise verhindert die seitliche Kollisionen auf beiden Seiten. Diese Gefahr besteht beim Passieren eines Korridors.

[2]Als Seitenabstand kann dabei der minimale Sensorwert der Sensoren #4 und #5 bzw. #6 und #7 verwendet werden, sofern die Sensorkonfiguration aus Abb. 3.2 b) gewählt wird. Für die Konfiguration aus Abb. 3.2 a) haben sich als Seitenabstände die Sensorwerte der Sensoren #0 und #3 bewährt.

Passieren eines Korridors: Befindet sich das Fahrzeug in einem Korridor, so daß Hindernisse in die Gefahrenzonen der Sensoren #4 und #7 eingetreten sind, darf die Stellgröße zur Lenkung keine zu große Wirkung haben. Ansonsten kann es zur Kollision kommen. Auch wenn kein Kollision eintritt, wird eine scharfe Kurve den größeren Freiraum im nächsten Schritt auf die gegenüberliegende Seite verlagern. Im nächsten Schritt wird dann eine entgegengesetzte Drehung benötigt. Eine solche Verhaltensweise führt zwangsläufig zu Oszillationen in der gefahrenen Trajektorie, da abwechselnd beide Seitenwände das Fahrzeug abstoßen. Um die Oszillationen zu verringern wird der Betrag der rotatorischen Bewegung dort niedrig gehalten, wo es möglich ist, da mit ihm auch die Oszillationen steigen. Ist der Unterschied der linken und rechten Seitenabstände eher klein[3], so werden die Oszillationen dadurch weiter verringert, daß einfach geradeaus gefahren wird.

Da in der Ebene der frontalen Hindernisvermeidung *scharfe* Kurven nötig sind, um den Hindernissen auszuweichen, hier allerdings sehr *weiche* Kurven zur Durchquerung eines Korridores nötig sind, reicht zur AMR-Steuerung eine feste Rotationsgeschwindigkeit nicht aus. Es wird daher eine zweite Rotationsgeschwindigkeit zur Feinsteuerung des Fahrzeuges eingeführt.

Vermeidung von seitlichen Hindernisse: Die Gefahrenzonen der Seitensensoren sind so bemessen, daß sie einen seitlichen Abstand angeben, der zu den Hindernissen unbedingt eingehalten werden soll. Tritt ein Hindernis nur auf einer Seite des Fahrzeuges in die Gefahrenzone ein (Sensoren #4 bzw. #7), so ist eine Kurve in die entgegengesetzte Richtung einzuschlagen. Die Sensoren #0 und #3 können in analoger Weise eingesetzt werden, um schräg seitlich liegende Hindernisse zu umgehen.

Um Oszillationen durch zu scharfe Kurven zu vermeiden, werden die Ausweichmanöver dieser Ebene durch weiche Kurven realisiert.

Die Verhaltensebenen bis einschließlich zu dieser Ebene gewährleisten das kollisionsfreie Umherfahren in konvex begrenzten Räumen und entspricht den unteren drei Ebenen im Mataric-Ansatz.

Die beiden folgenden Verhaltensebenen werden aufgabenabhängig verwendet und sind somit auf der gleichen Ebene anzuordnen. Das Wandfolgeverhalten ermöglicht ein Gesamtverhalten wie bei Mataric, während die Zielverfolgung eine direkte Ansteuerung von vorgegebenen Zielpunkten erlaubt.

Folgen von Wänden: Bis zu dieser Ebene gewährleisten die Verhalten lediglich das Folgen von Wänden eines konvexen Raumes. Um auch in Räumen mit konkaver Grundfläche an der Wand zu bleiben, wird diese Verhaltensebene eingeführt. Sie entspricht bei Mataric der Korrekturebene.

Kommt das Fahrzeug aus einem Korridor in einen offenen Raum und befinden sich dabei Hindernisse in den Sicherheitszonen der Sensoren #5 und #6 aber nicht in den Sicherheitszonen der Sensoren #4 und #7, wird eine Kurve in die Richtung eingeschlagen, dessen Seitenwand näher am Fahrzeug liegt. Dadurch wird gewährleistet,

[3] Hier wird überprüft, ob $|sensor[4] - sensor[7]| < \xi$ für ein vorgegebenes $\xi \in \mathbb{R}_{>0}$ gilt.

daß das Fahrzeug eine Wand weiterverfolgt, auch wenn es in einen offenen Raum fährt. Dieses ist insofern wünschenswert, als daß ein Raum ohne Hindernisse sehr schlechte Orientierungsmöglichkeiten für den AMR bietet.

Ähnlich sind die unsymmetrischen Fälle zu behandeln. Befindet sich ein Hindernis in der Sicherheitszone des Sensors #5 aber nicht in der Sicherheitszone des Sensors #4, so fährt das Fahrzeug an einer rechten Ecke vorbei und sollte daher eine Rechtskurve fahren, um der Wand weiter zu folgen. Analog dazu kann das Vorbeifahren an einer linken Ecke mit den Sensoren #6 und #7 behandelt werden. Tritt keiner dieser Fälle ein, so wird geradeaus gefahren.

Da oft rechtwinklige Ecken umfahren werden sollen und der AMR dabei ständigen Kontakt über die Ultraschallsensoren zur Wand haben muß, sind in dieser Verhaltensebene ausschließlich hohe Rotationsgeschwindigkeiten zu verwenden.

Folgen eines Zielpunktes: Bei der Verfolgung eines Zielpunktes werden keine Sensorwerte mehr benötigt, da die Kollisionsvermeidung durch die Verhaltensweise der unteren Ebenen gewährleistet wird. Dafür wird nun die Richtung des Zielpunktes im egozentrischen Roboterkoordinatensystem benötigt. Dabei werden nur Kombinationen aus folgenden Richtungen verwendet: *vorne* bzw. *hinten* und *links*, *mitte* bzw. *rechts*.

Die Steuerung kann recht einfach vorgenommen werden. Liegt das Ziel in der Mitte vor dem Fahrzeug, so muß es geradeaus fahren. Liegt das Ziel links oder rechts, so muß es entsprechend nach links oder rechts abbiegen. Liegt dagegen der Zielpunkt in der Mitte hinter dem Fahrzeug, könnte entweder eine Kurve in eine standardmäßig festgelegte Richtung gefahren werden oder es wird in die Richtung gedreht, die den kleineren Drehwinkel zum Zielpunkt besitzt. In der Implementation für den *LABMATE* wurde die letzte Möglichkeit gewählt.

Implementation der Verhaltensebenen

Jede einzelne Verhaltensebene ist als eine eigene Funktion implementiert, die über globale Variablen vollständigen Zugriff auf alle Sensorwerte und daraus abgeleiteten Merkmalen besitzt. Die *Zustandsanalyse* liest alle Sensordaten ein und bereitet diese Daten auf einfache Weise für die Verhaltensebenen auf[4]. Dieses Modul stellt im wesentlichen die in den Abbildungen 3.5 und 3.6 aufgeführten Variablen zur Verfügung.
Jedes einzelne Verhaltensmodul (vgl. Abbildung 3.8 - 3.14) greift auf diese Variablen zu und schlägt seinerseits in der globalen Variablen *command* ein Steuerungskommando vor. Prinzipiell könnten die einzelnen Verhaltensmodule auch parallelisiert werden und als eigene Prozesse arbeiten. Für die praktischen Versuche mit dem *LABMATE* wurde eine einfachere sequentielle Abarbeitung der Verhaltensebenen gewählt. Da die einzelnen Verhaltensmodule sehr einfach realisierbar sind, wird dennoch eine Echtzeitfähigkeit erreicht[5].

[4]Im folgenden wird die Zustandsanalyse durch die hier nicht näher spezifizierte Funktion *analyzeState* durchgeführt (vgl. Abb. 3.15).

[5]Auf dem 486 SX Laptop wurde bei sequentieller Abarbeitung eine Zykluszeit von 170 ms erreicht. Bei einer Schallgeschwindigkeit von 332 $\frac{m}{s}$ und Verwendung von 8 Ultraschallsensoren mit einer maximal

$emergency$	=	$\begin{cases} true, \text{ Hardware-Fehler} \\ false, \text{ sonst} \end{cases}$
$frontbumper$	=	$\begin{cases} true, \text{ Kollision an vorderer Stoßstange} \\ false, \text{ sonst} \end{cases}$
$rearbumper$	=	$\begin{cases} true, \text{ Kollision an hinterer Stoßstange} \\ false, \text{ sonst} \end{cases}$
$sensor[id]$	=	gemessener Sensorwert von Sensor $\#id$.
$dangerzone[id]$	=	Länge der Gefahrenzone des Sensors $\#id$.
$safezone[id]$	=	Länge der Sicherheitszone des Sensors $\#id$.
$dangerzoneEntered[id]$	=	$\begin{cases} true, sensor[id] < dangerzone[id] \\ false, \text{ sonst} \end{cases}$
$safezoneEntered[id]$	=	$\begin{cases} true, sensor[id] < safezone[id] \\ false, \text{ sonst} \end{cases}$
$leftMin$	=	$\min\{sensor[6], sensor[7]\}$
$rightMin$	=	$\min\{sensor[4], sensor[5]\}$
x_{goal}	=	x-Wert des Zielpunktes (allozentrische Koordinaten)
y_{goal}	=	y-Wert des Zielpunktes
x_{pos}	=	x-Wert der Roboterposition (allozentrische Koordinaten)
y_{pos}	=	y-Wert der Roboterposition
α	=	Orientierung des Roboters
$\begin{pmatrix} x \\ y \end{pmatrix}$	=	$\begin{pmatrix} \cos\left(\frac{\alpha\cdot\pi}{180^\circ}\right) & -\sin\left(\frac{\alpha\cdot\pi}{180^\circ}\right) \\ \sin\left(\frac{\alpha\cdot\pi}{180^\circ}\right) & \cos\left(\frac{\alpha\cdot\pi}{180^\circ}\right) \end{pmatrix} \begin{pmatrix} x_{goal} - x_{pos} \\ y_{goal} - y_{pos} \end{pmatrix}$, Zielpunkt im egozentrischen Koordinatensystem (vgl. Abb. 2.3)

Abbildung 3.5: Variablen zur Hindernisvermeidung

$tolerance$	=	Genauigkeit zur Zielpunktansteuerung
$goalInFront$	=	$(y > tolerance)$
$goalBehind$	=	$(y < -tolerance)$
$goalLeft$	=	$(x > tolerance)$
$goalRight$	=	$(x < -tolerance)$
$command$	=	Kommando zur Motoransteuerung
$jogRate$	=	Rotationsrate für *scharfe* Kurven
$smJogRate$	=	Rotationsrate für *weiche* Kurven
vel	=	momentane Fahrtgeschwindigkeit
ξ	=	vorgegebene Differenz linker und rechter Sensorwerte, für die in Korridoren geradeaus gefahren wird.

Abbildung 3.6: Variablen zur Hindernisvermeidung

Durch die sequentielle Implementierung der einzelnen Verhaltensweisen, kann auf einfache Weise die Auswahl der aktiven Verhaltensebene realisiert werden. Dazu wird zu Beginn eines Kontrollzyklus die Variable *command* mit IDLE belegt. Sequentiell werden dann die Verhaltensebenen nacheinander aufgerufen, wobei niedrigere Verhaltensweisen vor höheren

meßbaren Entfernung von 5 m erhält man höchstens alle $\frac{8\cdot 2\cdot 5\ m}{332\ \frac{m}{s}} \approx 241\ ms$ neue Signale. Damit liegt die Zykluszeit in der gleichen Größenordnung wie die Zeit in der die durch die Sensorik zur Verfügung gestellten Daten eintreffen.

MOVE_FWD	=	Konstante für Geradeausbewegung
TURN_LEFT	=	Konstante für scharfe Linkskurve
TURN_RIGHT	=	Konstante für scharfe Rechtskurve
SM_TURN_LEFT	=	Konstante für weiche (engl. smooth) Linkskurve
SM_TURN_RIGHT	=	Konstante für weiche Rechtskurve
ORIENTATE	=	Konstante für Orientierungsphase nach einem Notstop
ESCAPE_BACK	=	Konstante für Zurücksetzen nach einem Notfallstop
ESCAPE_FWD	=	Konstante für Vorsetzen nach einem Notfallstop
STOP	=	Konstante für einen sofortigen Stop
IDLE	=	Konstante für das *unbestimmte* Motorkommando

Abbildung 3.7: Konstanten zur Hindernisvermeidung

```
function escapeAfterCrash()
    if emergency then
        apply(STOP);
        command := ORIENTATE;
    else if frontbumper then
        apply(STOP);
        command := ESCAPE_BACK;
    else if rearbumper then
        apply(STOP);
        command := ESCAPE_FWD;
    fi;
end;
```

Abbildung 3.8: Verhalten: *Entkomme Gefahrensituation*

```
function avoidDanger()
    if  dangerzoneEntered[0] ∨
        dangerzoneEntered[1] ∨
        dangerzoneEntered[2] ∨
        dangerzoneEntered[3] then
        apply(STOP);
        command := ESCAPE_BACK;
    fi;
end;
```

Abbildung 3.9: Verhalten: *Vermeide Gefahrensituation*

Verhaltensweisen den Vorrang haben. Dieses geschieht solange, bis ein Verhalten ein Kommando zur Motorsteuerung vorschlägt und dieses Kommando in der Variablen *command* gespeichert wird. Damit steht das Kommando für den aktuellen Kontrollzyklus fest und höhere Verhaltensebenen werden nicht mehr aufgerufen. Diese Vorgehensweise beschreibt den in Abbildung 3.15 dargestellten Algorithmus. Im Gegensatz zur Subsumtion bei [Bro86] unterdrücken hier niedrigere Verhaltensebenen die Ausgaben von höheren Verhaltensebenen.

Die möglichen Befehle, die *command* annehmen kann sind in Abbildung 3.7 dargestellt. Nach Berechnung einer Motorstellgröße wird das in *command* gespeicherte Kommando an die Einheit übergeben, die die Kommandos in für die TRC-Schnittstelle verständliche Befehle umsetzt und diese an den Roboter sendet. Diese Einheit hat als einzige direkten Zugang zum Lokomotionssystem des Roboters (Motorik) und wird als *Pilot* bezeichnet.

Der Pilot ist in Abbildung 3.16 dargestellt. Außer im Falle eines Notstops wird wie bei Mataric zunächst eine konstante Geschwindigkeit für das Fahrzeug gewählt ($0.2\ \frac{m}{s}$). Motiviert durch [Mat92] wurden für die Richtungssteuerung in ersten Versuchen auch nur

```
function avoidFrontObstacle()
    if  safezoneEntered[1] ∧
        safezoneEntered[2] then
        if sensor[0] < sensor[3] then
            command := TURN_RIGHT;
        else
            command := TURN_LEFT;
        fi;
    else if safezoneEntered[1] then
            command := TURN_RIGHT;
    else if safezoneEntered[2] then
            command := TURN_LEFT;
    fi;
end;
```

Abbildung 3.10: Verhalten: *Vermeide frontale Hindernisse*

```
function passCorridor()
    if  dangerzoneEntered[4] ∧
        dangerzoneEntered[7] then
        if sensor[7] − sensor[4] < ξ then
            command := SM_TURN_LEFT;
        else if sensor[4] − sensor[7] > ξ then
            command := SM_TURN_RIGHT;
        else
            command := MOVE_FWD;
        fi;
    fi;
end;
```

Abbildung 3.11: Verhalten: *Passiere Korridor*

```
function avoidLateralObstacle()
    if dangerzoneEntered[7] then
        command := SM_TURN_RIGHT;
    else if dangerzoneEntered[4] then
        command := SM_TURN_LEFT;
    else if  safezoneEntered[0] ∧
             safezoneEntered[3] then
        command := MOVE_FWD;
    else if safezoneEntered[0] ∧
        command := SM_TURN_RIGHT;
    else if safezoneEntered[3] ∧
        command := SM_TURN_LEFT;
    fi;
end;
```

Abbildung 3.12: Verhalten: *Vermeide seitliche Hindernisse*

```
function followWall()
    if  ¬ safezoneEntered[7] ∧
        safezoneEntered[6] ∧
        ¬ safezoneEntered[4] ∧
        safezoneEntered[5] then
        if leftMin < rightMin then
            command := TURN_RIGHT;
        else
            command := TURN_LEFT;
        fi;
    else if  ¬ safezoneEntered[7] ∧
             safezoneEntered[6] then
        command := TURN_LEFT;
    else if  ¬ safezoneEntered[4] ∧
             safezoneEntered[5] then
        command := TURN_RIGHT;
    else
        command := MOVE_FWD;
    fi;
end;
```

Abbildung 3.13: Verhalten: *Folge Wänden* (inkl. konkaven Begrenzungen)

drei Arten von Befehlen (fahre geradeaus, fahre Rechtskurve und fahre Linkskurve) verwendet. Dabei stellte sich heraus, daß für die frontale Kollisionsvermeidung relativ hohe Rotationsgeschwindigkeiten (in der Größenordnung von $15°s^{-1} - 20°s^{-1}$) notwendig sind. Allerdings resultieren aus hohen Rotationsgeschwindigkeiten starke Oszillationen, wenn

```
function followGoal()
    if  goalInFront ∧
            ¬goalLeft ∧
            ¬goalRight then
        command := MOVE_FWD;
    else if goalLeft then
        command := TURN_LEFT;
    else if  goalRight then
        command := TURN_RIGHT;
    else if goalBehind then
        if leftMin < rightMin then
            command := TURN_RIGHT;
        else
            command := TURN_LEFT;
        fi;
    else
        command := STOP;
    fi;
end;
```

Abbildung 3.14: Verhalten: *Folge Zielpunkt*

```
function select()
    do
        analyzeState();
        command := IDLE;
        escapeAfterCrash();
        if command = IDLE then
            avoidDanger();
        if command = IDLE then
            avoidFrontObstacle();
        if command = IDLE then
            passCorridor();
        if command = IDLE then
            avoidLateralObstacle();
        if command = IDLE then
            followWall();
        apply(command);
    loop;
end;
```

Abbildung 3.15: Verhaltensauswahl

```
function apply(command)
    case command
        STOP :
            jog(0);
            emergency_stop();
        MOVE_FWD :
            jog(0);
            set_velocity(vel,vel);
        MOVE_BACK :
            jog(0);
            set_velocity(−vel,vel);
        ESCAPE_FWD :
            escape(ESCAPE_FWD);
        ESCAPE_BACK :
            escape(ESCAPE_BACK);
        TURN_RIGHT :
            jog(−jogRate);
            set_velocity(vel,vel);
        TURN_LEFT :
            jog(jogRate);
            set_velocity(vel,vel);
        SM_TURN_RIGHT :
            jog(−smJogRate);
            set_velocity(vel,vel);
        SM_TURN_LEFT :
            jog(smJogRate);
            set_velocity(vel,vel);
        ORIENTATE :
            orientate();
    end;
end;
```

Abbildung 3.16: Pilot

abwechselnd Links- und Rechtskurven gefahren werden, um z. B. das Fahrzeug in der Mitte eines Ganges zu halten. Dieses Problem tritt bei Mataric in dieser Form nicht auf, da der dort verwendete Roboter erheblich kleiner als der *LABMATE* ist und somit auch in Gängen von Bürogebäuden nicht so oft eine Richtungskorrektur vornimmt. Ferner wird die Kurvenfahrt bei Mataric nicht durch eine überlagerte Rotation realisiert. Vielmehr wird dort das Fahrzeug gestoppt und dann auf der Stelle gedreht. Anschließend wird die Fahrt

fortgesetzt. Aus diesen Gründen wird an dieser Stelle geringfügig von der Ansteuerung des Roboters wie in [Mat92] abgewichen.
Die Kurvenfahrt, die durch TURN_LEFT und TURN_RIGHT angegeben wird, realisiert eine scharfe Links- bzw. Rechtskurve, die insbesondere zum Ausweichen von frontalen Hindernissen geeignet ist. Zur Feinkorrektur der Fahrtrichtung wird eine Kurvenfahrt durchgeführt, die mit den Kommandos SM_TURN_LEFT bzw. SM_TURN_RIGHT ausgelöst wird. Diese Kommandos veranlassen den Piloten dazu, eine Kurve zu fahren bei der der Vorwärtsbewegung des *LABMATE* eine betragsmäßig kleine Rotationsgeschwindigkeit überlagert wird (z. B. $1°s^{-1} - 3°s^{-1}$).

Exploration

Mit *Exploration* bezeichnet man die Phase, in der der Roboter ziellos durch seine Umgebung fährt, um diese zu erkunden und dabei Daten über die Umgebung zu sammeln.
Ein Schritt in diese Richtung wurde schon durch die Verhaltensebene des Wandfolgens angegeben. Dieses Verhalten gewährleistet die Erkundung eines Raumes entlang der ihn begrenzenden Wände. Um die vollständige Umgebung zu erkunden und nicht immer nur an einer Wand zu bleiben, muß das Fahrzeug in der Explorationsphase auch in Richtungen fahren, in die es durch das Folgen von Wänden nicht gelangt.
Da bis zu diesem Entwicklungsstand noch keine Informationen über die Umgebung in Form einer Karte vorliegen, ist es sehr schwer, die Exploration so zu gestalten, daß das Fahrzeug alle Bereiche seiner Umgebung erkundet. Um zumindestens langfristig alle Bereiche der Umgebung zu erkunden, wird das in Abbildung 3.13 dargestellte Verhalten der Wandverfolgung so abgeändert, daß mit einer vorgegebenen Wahrscheinlichkeit $p \in [0, 1]$ von der Geradeausfahrt, die das Wandfolgeverhalten im letzten $else$-Zweig durchführt, abgewichen wird. Dazu wird eine auf $[0, 1]$ gleichverteilte Zufallsvariable X beobachtet und im Falle $X \leq p$ von der Geradeausfahrt abgewichen. Tritt dieser Fall ein, wird eine Kurve in die Richtung eingeschlagen, in die der Freiraum in der Umgebung am größten ist. Ist der Freiraum schließlich auf beiden Seiten gleich groß, so wird geradeaus gefahren.
Diese Exploration in den freien Raum wird solange durchgeführt, bis der AMR wieder in die Nähe eines Hindernisses gelangt und niedrigere Verhaltensebenen wieder aktiv werden.

Orientierung des AMR's nach einem Notfall

Bei zu geringem Abstand zu einem Hindernis oder nach einer Kollision leitet der AMR einen Notfallstop ein. Anschließend wird das Fahrzeug zurückgesetzt und neu orientiert.
Während das Zurücksetzen durch einen einfachen Motorsteuerbefehl ausgelöst wird, bedarf es bei der Neuorientierung etwas mehr Aufwand.
Befindet sich der AMR im Stillstand, so kann durch einfaches Setzen der jog-Rate das Fahrzeug auf der Stelle gedreht werden. Dabei wird ständig die Orientierung des Fahrzeuges überwacht und die Drehung spätestens nach einer Drehung um 360° abgebrochen. Außerdem werden während der Punktdrehung das Minimum der Sensorwerte #1 und #2 überwacht. Hat sich das Fahrzeug soweit gedreht, daß sich vor dem Fahrzeug keine Hindernisse mehr befinden, so kann die Rotation schon vorzeitig abgebrochen und die Fahrt

fortgesetzt werden. Als Abbruchkriterium für die Rotation verwendet man

$$\min\{sensor[1], sensor[2]\} \geq \Theta, \tag{3.1}$$

wobei Θ die minimale Länge eines Fluchtweges angibt.
Am häufigsten kommt es beim Durchqueren von schmalen Türrahmen zur Neuorientierung, da das Fahrzeug zunächst auf den Türrahmen gefahrlos zusteuert, aber kurz vor dem Rahmen einer der Sensoren #0 oder #3 einen Wert innerhalb seiner Gefahrenzone liefert. Der AMR wird dann so orientiert, daß er anschließend durch die Tür fahren kann. Für dieses Verhalten ist die Drehrichtung während der Orientierungsphase von entscheidender Bedeutung. Die Drehrichtung wird so bestimmt, daß der erste durch die Frontsensoren detektierte Freiraum der Türrahmen ist. Dazu wird die Drehrichtung von den Werten der Sensoren #0 bis #3 abhängig gemacht. Man vergleicht die Werte $l := \min\{sensor[0], sensor[1]\}$ und $r := \min\{sensor[2], sensor[3]\}$ und führt eine Rechtsrotation durch, wenn $l \leq r$ und sonst eine Linksrotation.
Ein anderes Verhalten ist dagegen bei der Zielpunktverfolgung notwendig. Hier richtet sich die Drehrichtung nach der Lage des Zielpunktes.
Im Algorithmus der Abbildung 3.15 wird die Orientierung durch einen Aufruf der Funktion *orientate* und das Zurücksetzen mit anschließender Orientierung durch Aufruf der Funktion *escape* durchgeführt. Beide Funktionen werden hier nicht weiter ausgeführt.

3.1.3 Experimente

Zum Abschluß dieses Abschnittes wird die Wirksamkeit der hier vorgestellten Methoden an einigen Experimenten diskutiert.
Dabei wurden alle hier dargestellten Versuche sowohl mit dem realen Robotersystem als auch im Simulator durchgeführt. Die hier dargestellten Ergebnisse zeigen nur die Resultate im Simulator, da die Aufzeichnung der gefahrenen Strecken des Roboters im Zusammenspiel mit der realen Umgebung ohne weitere Vorkehrungen in der Umwelt und am Roboter nicht möglich sind[6]. In vielen Experimenten wurden verschiedene Parameter für die Hindernisvermeidung untersucht. Gute Resultate wurden dabei mit den Parametern aus Tabelle 3.1 und 3.2 erzielt.
Die in Tabelle 3.2 angegebene physikalische Sensornummer bezeichnet die Anschlußports an der Hauptplatine der Ultraschallsensorik. Diese Reihenfolge ist daher wichtig, da die Anfragen an die Ultraschallsensoren in der Reihenfolge der physikalischen Numerierung erfolgt. Um Störeffekte zwischen den Sensoren durch Inferenz der Schallsignale weitgehend zu vermeiden, wurden in der Reihenfolge benachbarte Sensoren räumlich möglichst weit voneinander getrennt am AMR angebracht. Die logische Numerierung der Sensoren entspricht der von Abbildung 3.2 a). Für die Anordnung der Sensoren wurde ebenfalls der Aufbau aus Abbildung 3.2 a) verwendet. Diese Anordnung hat sich insbesondere beim Passieren von Türen gegenüber der Anordnung aus Abbildung 3.2 b) als besser herausgestellt, da mit ihr Türrahmen besser erkannt werden können.

[6] In [Mat92] wird die gefahrene Strecke des Roboters dadurch aufgemessen, daß an der Roboterplattform ein Stift angebracht ist, der die Fahrtroute auf einem gekachelten Boden markiert. Die markierten Kacheln ($30\ cm \times 30\ cm$) geben dann die ungefähre Fahrroute des AMR's an.

Parametername	verwendeter Wert für den Parameter
Geschwindigkeit	200 $\frac{mm}{s}$
hohe *jog*-Rate	$19°s^{-1}$
niedrige *jog*-Rate	$2°s^{-1}$
Rücksetzweite	260 mm
Zielpunkttoleranz	500 mm
Korridordifferenz ξ	20 mm
Explorationswahrscheinlichkeit p	0.04
Länge eines Fluchtweges Θ	700 mm

Tabelle 3.1: Parameter der Hindernisvermeidung

Sensor-Nr.	phys. Sensor-Nr.	Timeout	Gefahrenzone	Sicherheitszone
0	0	1100 mm	250 mm	920 mm
1	4	1320 mm	350 mm	1300 mm
2	2	1320 mm	350 mm	1300 mm
3	6	1100 mm	250 mm	920 mm
4	3	1200 mm	550 mm	800 mm
5	1	1200 mm	550 mm	800 mm
6	7	1200 mm	550 mm	800 mm
7	5	1200 mm	550 mm	800 mm

Tabelle 3.2: Parameter für die Ultraschallsensoren

Die in den Abbildungen 3.17 bis 3.24 dargestellten Versuche zeigen jeweils die im Simulator modellierte Umwelt und den *LABMATE*, der anhand der simulierten Ultraschallkegel zu erkennen ist. Um das Fahrverhalten des AMR's darzustellen, ist die Fahrtroute des Roboters in der Umgebung als einhüllender Schlauch dargestellt, der die gesamte Breite des Roboters umfaßt. Dadurch ist auch sichtbar, wie nah Hindernisse passiert werden.
In Abbildung 3.17 ist dargestellt, wie der AMR durch den Flur eines Bürogebäudes fährt, und dabei zwei Türen passiert.
Welche Wirkung die Verhaltensebene des Wandfolgens hat, ist in Abbildung 3.18 angegeben. Das Fahrzeug fährt entlang einer Wand auf eine Ecke zu. Um diese Ecke nicht zu verlieren, wird eine scharfe Linkskurve eingeschlagen, die das Fahrzeug schließlich wieder an die nächste Wand bringt.
In den Abbildungen 3.19 und 3.20 ist die Wirkung des Verhaltens zum Passieren von Korridoren veranschaulicht. Während in Abbildung 3.20 ausschließlich scharfe Kurven gefahren werden und diese zu starken Oszillationen im Gang führen, schafft es das Fahrzeug in Abbildung 3.19, sich allmählich am Gang auszurichten, bis es schließlich nahezu gerade fährt. Dazu ist eine geringe Rotationsgeschwindigkeit notwendig.
Die Abbildungen 3.21 und 3.22 zeigen zwei Versuche zur Durchquerung eines Türrahmens.

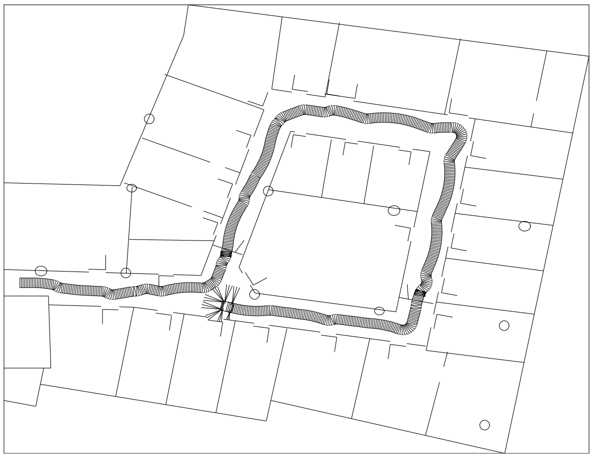


Abbildung 3.17: Der *LABMATE* beim Durchqueren eines Bürogebäudes

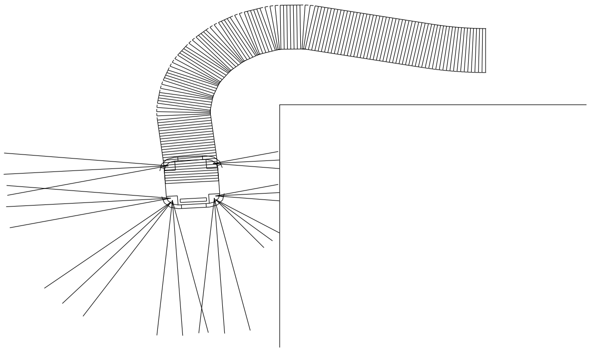


Abbildung 3.18: Umfahren einer konkaven Umgebungsbegrenzung

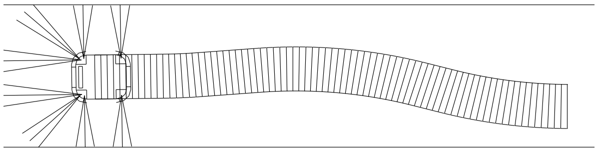


Abbildung 3.19: Vermeidung von Oszillationen in Korridoren durch *weiche* Kurven

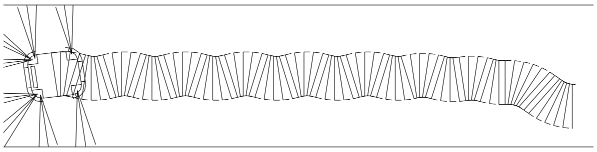


Abbildung 3.20: Oszillationen in Korridoren bei Verwendung *scharfer* Kurven

Die verwendeten Abmessungen der Umgebung sind möglichst naturgetreu an die reale Testumgebung des *LABMATE* angepaßt. In beiden Abbildungen ist zu sehen, wie der AMR im ersten Versuch nicht durch die Tür kommt. Stattdessen setzt er zurück und leitet eine kurze Orientierungsphase ein, bis der Türrahmen erkannt wird. Dann wird ein neuer Versuch unternommen. In Abbildung 3.21 steht die Tür offen und das Türblatt ragt in den Gang hinein. Das führt dazu, daß das Fahrzeug schon frühzeitig dem Türblatt ausweicht und seine Position zur Türdurchquerung dadurch verschlechtert. Erst nach vier Versuchen gelingt es, die Tür zu durchqueren. Ohne das störende Türblatt reichen zwei Versuche, um durch die Tür zu gelangen (vgl. Abb. 3.22). Dieses Ergebnis läßt sich sowohl im Simulator als auch in der Realität beobachten. Man erkennt daran, wie der Simulator auch eine komplexe Wechselwirkung zwischen dem Roboter und seiner Umwelt nachahmen kann.

Die letzten beiden Abbildungen 3.23 und 3.24 veranschaulichen den Unterschied zwischen einem reinen Wandfolgen und der Exploration der Umgebung durch zufälliges Abweichen von dem Wandfolgen durch das Fahren in den freien Raum. In Abbildung 3.23 findet ausschließlich ein Wandfolgen statt. Das Fahrzeug wird keine neuen Regionen in seiner Umgebung mehr erreichen, da die umgebenden Wände seine Route festlegen. Fährt es stattdessen von Zeit zu Zeit in den freien Raum, erreicht das Fahrzeug in Abbildung 3.24

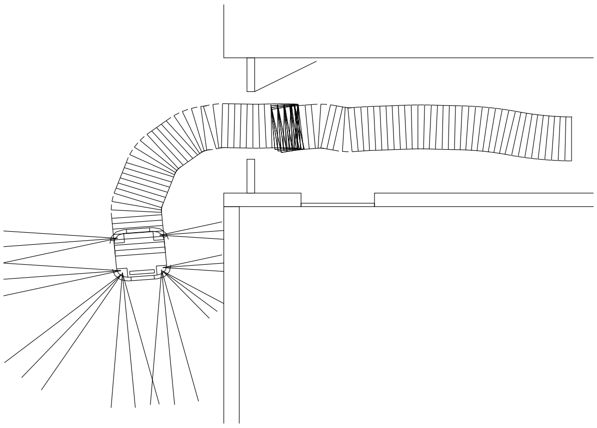


Abbildung 3.21: Türdurchquerung mit offen stehender Tür

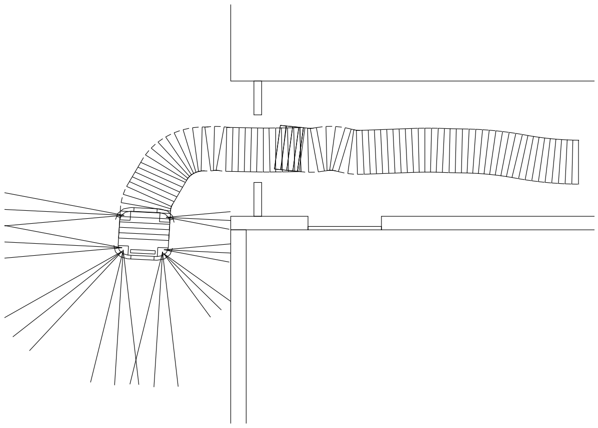


Abbildung 3.22: Türdurchquerung ohne offenstehende Tür

jeden Ort der Umgebung. Das Fahren in den freien Raum kann in der Abbildung an den Stellen beobachtet werden, wo kein Grund zum Ausweichen besteht und das Fahrzeug die Richtung dennoch ändert.

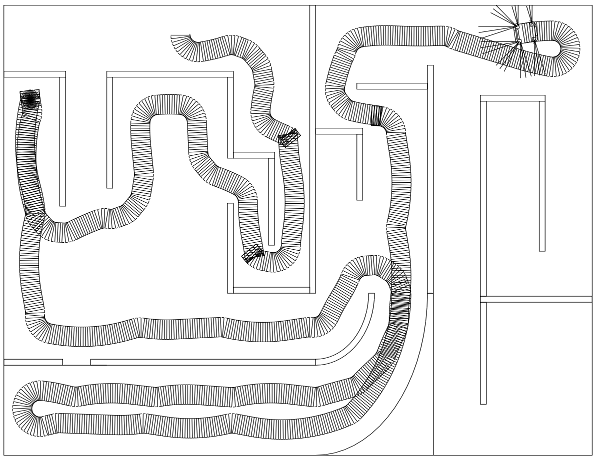


Abbildung 3.23: Exploration durch ausschließliche Wandverfolgung

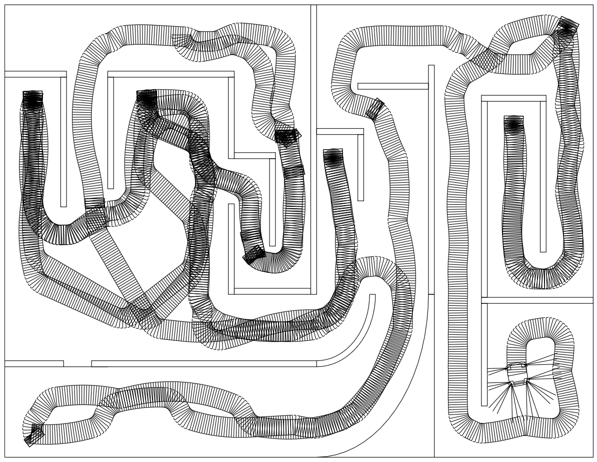


Abbildung 3.24: Exploration mit zufälliger Fahrt in den freien Raum

3.1.4 Variable Geschwindigkeitskontrolle

Die bisherige Motorsteuerung ist recht primitiv. Das Fahrzeug fährt mit konstanter Geschwindigkeit und kann zwischen fünf Rotationsgeschwindigkeiten wählen, um den Hindernissen auszuweichen.
Mit diesen Steuerkommandos konnte bei relativ kleinen Geschwindigkeiten ein gutes Fahrverhalten erreicht werden. Erhöht man jedoch die Fahrtgeschwindigkeit, müssen auch die Rotationsgeschwindigkeiten erhöht werden. Damit kann das Fahrzeug eine Kurve ohne

Zwischenstopp und ohne anschließende Neuorientierung durchqueren. Eine Erhöhung der Rotationsgeschwindigkeiten für die scharfe Links- bzw. Rechtskurve hat aber gleichzeitig den Nachteil, daß ebenfalls an Wänden oder in Korridoren (vgl. Wandfolgeverhaltenerhalten) hohe Rotationsgeschwindigkeiten verwendet werden. Dieses führt aber zu starken Oszillationen.

Um dennoch eine Steigerung der Fahrtgeschwindigkeit zu erlauben, wird die Forderung nach einer konstanten Fahrtgeschwindigkeit wie bei [Mat92] nicht weiter aufrecht erhalten. Dagegen wird eine variable Kontrolle der Fahrtgeschwindigkeit eingeführt. Sie richtet sich nach dem frontalen Abstand des Fahrzeuges zu Hindernissen. Eine solche Geschwindigkeitskontrolle wird z. B. in [BB+95] verwendet. Möchte man gewährleisten, daß eine frontale Annäherung des Fahrzeuges an ein Hindernis bis auf eine Distanz von d_{stop} mindestens eine Zeit T in Anspruch nimmt, so muß für die Fahrtgeschwindigkeit

$$v \leq \frac{d_{min} - d_{stop}}{T} \tag{3.2}$$

gelten, wobei d_{min} den minimalen frontalen Abstand zu einem Hindernis angibt. Für $T = 2s$ erhält man die in [BB+95] angegebene 2-Sekunden-Regel. Berechnet man dabei d_{min} durch $\min\{sensors[1], sensors[2]\}$, so werden die beiden vorderen Ecken (Sensoren #0 und #3) nicht berücksichtigt. Das führt insbesondere in Türrahmen zu einer überhöhten Geschwindigkeit. Eine Minimumbildung über die vorderen vier Sensoren kommt aber auch nicht in Frage, da sonst in Korridoren zu geringe Geschwindigkeiten erreicht werden. Daher wird hier ein Schwellwert $d_s > 0$ eingeführt, mit dem die Meßwerte der diagonal nach vorne gerichteten Sensoren verglichen werden. Liegt ihr Minimum unterhalb des Schwellwertes, so ist d_{min} das Minimum aller vier Frontsensoren und sonst das Minimum der Sensoren #1 und #2. Der Wert von d_{min} errechnet sich daher durch

$$\begin{aligned}
m_f &:= \min\{sensor[1], sensor[2]\}, && (3.3)\\
m_{lr} &:= \min\{sensor[0], sensor[3]\}, && (3.4)\\
m_{all} &:= \min\{sensor[0], \cdots, sensor[3]\}, && (3.5)\\
d_{min} &:= \begin{cases} m_f & : \quad m_{lr} > d_s \\ m_{all} & : \quad sonst \end{cases} && (3.6)
\end{aligned}$$

Für $d_{min} < d_{stop}$ wird $v = 0$ gesetzt. Zusätzlich führt man eine Maximalgeschwindigkeit v_{max} ein, die durch v nicht überschritten werden darf.

Durch diese variable Geschwindigkeitskontrolle kann das Fahrzeug in freier Umgebung mit erhöhter Geschwindigkeit fahren, während es in einer engen Umgebung seine Geschwindigkeit stark reduziert.

Da der minimale frontale Abstand d_{min} direkt von den Sensorwerten abhängt, ist dieser Wert jedoch stark verrauscht, so daß sich das Rauschen in den Sensordaten durch die Kopplung der Fahrtgeschwindigkeit an den frontalen Hindernisabstand direkt auf die Fahrtgeschwindigkeit auswirkt. Ist dabei die Beschleunigung des Fahrzeuges groß (z. B. $1000 \, \frac{mm}{s^2}$), so bewirkt eine durch das Rauschen bedingte sprunghafte Änderung des gemessenen frontalen Hindernisabstandes eine sprunghafte Änderung im Fahrverhalten. Eine Verringerung der Beschleunigung führt zwar zu einer Verbesserung und vermeidet weitgehend ein ruckartiges Ändern der Fahrtgeschwindigkeit, führt aber gleichzeitig auch dazu,

daß die Beschleunigung der Rotationen verringert werden. Durch diese hardwarebedingte Kopplung von Rotations- und Fahrtbeschleunigung wird die Beschleunigung zu einer schwer handhabbaren Größe. Für die Versuche mit dem *LABMATE* wurde diese auf 900 $\frac{mm}{s^2}$ gesetzt.
Versuche, das Fahrverhalten durch Glättung der Rotations- und Fahrtgeschwindigkeit über die Zeit zu verbessern, waren nicht erfolgreich, da durch die Glättung die Reaktionszeiten verschlechtert werden und wieder Oszillationen entstehen. Geringe Verbesserungen konnten mit einer Glättung der Sensormessungen erreicht werden. Dazu wurde die Glättung aus (2.1) mit $\gamma_{sensor} = 0.6$ verwendet.
Ein Problem stellt die statische Rotationsgeschwindigkeit der scharfen Links- und Rechtskurve dar. Ihre Geschwindigkeit war bisher an die Fahrtgeschwindigkeit angepaßt, um so ein möglichst optimales Fahrverhalten zu erzeugen. Da die Fahrtgeschwindigkeit nun variabel gehalten ist, bietet sich auch eine geschwindigkeitsabhängige Kontrolle der Rotationen an. Dazu werden die Rotationsgeschwindigkeiten der scharfen Links- und Rechtsrotation um einen Faktor $f(v)$ verstärkt bzw. gedämpft, der von der momentanen Fahrtgeschwindigkeit v abhängt. Dieser Faktor wird definiert durch

$$f(v) = f_1 + \frac{f_2 - f_1}{v_{max}} \cdot v. \tag{3.7}$$

Dabei gibt f_1 die Verstärkung bei $v = 0$ und f_2 die Verstärkung bei $v = v_{max}$ an. Zwischen 0 und v_{max} wird linear interpoliert. In Experimenten mit dem *LABMATE* haben sich $f_1 = 1.5$ und $f_2 = 0.5$ als gute Werte herausgestellt. Bei hoher Geschwindigkeit, also bei wenigen Hindernissen, werden die Rotationen gedämpft und die Oszillationen dadurch verringert. Bei geringen Geschwindigkeiten werden die Rotationen dagegen verstärkt, um vorhandenen Hindernissen möglichst schnell ausweichen zu können.
Als weitere Parameter wurden $d_{stop} = 300\ mm$, $d_s = 400\ mm$, $v_{max} = 600\ \frac{mm}{s}$ und $T = 3\ s$ verwendet, mit denen ein recht gutes Fahrverhalten bei relativ hoher Fahrtgeschwindigkeit erreicht werden konnte.
Diese hier angestellten Überlegungen zeigen die Notwendigkeit für kontinuierliche Aktionen, um ein gutes Fahrverhalten des AMR zu erzeugen. Kontinuierliche Aktionen, die sogar stetig von den Sensorwerten abhängen, sind mit den bisherigen Verhaltensregeln nicht zu erreichen. Kontrollmechanismen, die sich für solche Probleme anbieten, sind die Fuzzy-Regler. Ein Fuzzy-Ansatz zur Steuerung eines AMR's wird in Kapitel 5 angegeben.

3.2 Kartographierung und Selbstlokalisation

Nachdem im vorherigen Abschnitt eine Hindernisvermeidungskomponente für die Navigation eines AMR's angegeben wurde, soll in diesem Abschnitt gezeigt werden, wie die lokalen Umweltinformationen zu einer globalen Umgebungskarte zusammengefügt werden. Dabei wird in zwei Schritten vorgegangen. Bevor überhaupt Informationen über die Umwelt gesammelt werden können, bedarf es der Bestimmung und Orientierung des Roboters im allozentrischen Koordinatensystem. Dieses Problem wird im folgenden als *Selbstlokalisation* bezeichnet.

Im zweiten Schritt werden Daten über die Umwelt gewonnen, aus denen eine Karte erstellt wird. Zur Erstellung solcher Karten gibt es in der Literatur zwei grundsätzliche Vorgehensweisen.
Die erste Möglichkeit orientiert sich an den Hindernissen in der Umgebung und versucht diese in einer Karte zu verzeichnen. So kann man beispielsweise die Umgebung als ein Netz von Gitterplätzen (*occupancy grids*) auffassen, in die eingetragen wird, ob die korrespondierenden Plätze in der Umwelt mit Hindernissen besetzt sind. Ein derartiger Ansatz wird hier nicht behandelt, da er recht aufwendig und speicherintensiv ist. Ein Beispiel für eine derartige Umweltrepräsentation findet sich in [Thr93],[BB+95] und [TM93].
Die zweite Möglichkeit orientiert sich nicht an den Hindernissen, sondern an den Freiräumen in der Umgebung. Ein solcher Ansatz wird hier behandelt, da er der bereits angegebenen Hindernisvermeidung sehr entgegen kommt.
Die Hindernisvermeidung ist in der Lage, das Fahrzeug durch komplexe Umgebungen zu lenken, ohne eine genaue Umweltrepräsentation zu benötigen. Damit der AMR von einem beliebigen Ort zu einem anderen Ort gelangt, werden Informationen über den zu fahrenden Weg benötigt. Dazu ist es ausreichend, einen möglichen Weg durch eine Folge von Markierungen zu kennzeichnen. Um von einer Markierung zur nächsten zu gelangen, verläßt sich das System vollständig auf die Hindernisvermeidung. Eine passende Repräsentation solcher Wege sind Graphen.

3.2.1 Selbstlokalisation

Zunächst soll das Problem der Ortsbestimmung des Roboters im allozentrischen Koordinatensystem behandelt werden. Der *LABMATE* ist mit einer Meßvorrichtung ausgestattet, mit der über die Länge der Abwicklungen der beiden Antriebsräder entlang eines gefahrenen Weges auf die Position und die Orientierung des Fahrzeuges geschlossen wird. Dieses Verfahren wird auch mit *Odometrie* bezeichnet.
Eigentlich könnte die Selbstlokalisation mit Hilfe der Odometrie schon gelöst sein. Problematisch daran ist aber, daß die Messungen über den Kontakt der Antriebsräder mit dem Boden stattfinden. Durch Schlupf der Antriebsräder oder Verrutschen des Fahrzeuges nach Kollisionen sind die Meßdaten der Odometrie enormen Fehlern unterworfen. Diese Fehler sind besonders groß bei der Bestimmung der Orientierung des Fahrzeuges. Dieses führt im schlimmsten Fall dazu, daß die Erzeugung einer Karte unmöglich wird.
Die in Abbildung 3.25 dargestellte Tour wurde von der Odometrie des *LABMATE* während einer geschlossenen Fahrt durch ein Bürogebäude aufgezeichnet. Dabei ist das Fahrzeug nach Umrundung eines Hausflures am Startpunkt wieder angekommen. Die aufgezeichnete Tour ist dagegen nicht geschlossen und liefert damit eine unbrauchbare Karte der Umgebung.
Es stellt sich nun die Frage, wie man die Meßfehler korrigiert. In [BB+95] und [CM93] wird dieses Problem dadurch gelöst, daß zusätzliche Annahmen an die Beschaffenheit der Umwelt verwendet werden. Es wird davon ausgegangen, daß der AMR in Bürogebäuden eingesetzt wird, die ausschließlich aus Gängen bestehen, die zueinander parallel oder senkrecht sind.
In [Mat92] wird die Orientierung mit Hilfe eines Kompasses durchgeführt. Weitere Möglichkeiten bestehen darin, Punkte in der Umwelt, deren Koordinaten bekannt sind, durch den

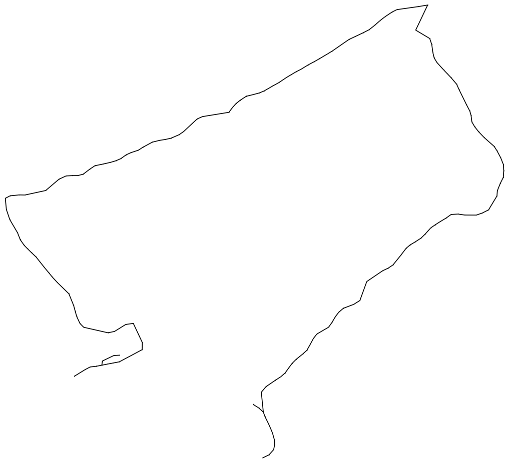


Abbildung 3.25: Durch Meßfehler der Odometrie verfälschte Topologie der Umgebung

Roboter eindeutig zu identifizieren und die Roboterposition an die bekannten Koordinaten des Punktes anzupassen. Die Identifikation kann über Funksender, Infrarotlichtschranken oder auch über Kameras durchgeführt werden. Die Identifikation von Punkten in der Umwelt durch Ultraschallsensoren ist dagegen ungeeignet.

Da die aufgezählten technischen Erweiterungen für dieses Projekt nicht zur Verfügung standen und auch die für den *LABMATE* zur Verfügung stehende Testumgebung nicht aus parallelen und senkrechten Gängen besteht, konnte die *Rekalibrierung* der allozentrischen Koordinaten für den realen AMR nicht umgesetzt werden. Dennoch wird im folgenden eine Lösung angegeben, die ähnlich wie bei [CM93] und [BB⁺95] von Zusatzannahmen über die Beschaffenheit der Umgebung ausgeht.

Statt davon auszugehen, daß die Umgebung nur aus Gängen besteht, die parallel oder senkrecht zueinander stehen, wird hier vorausgesetzt, daß die Umwelt nur aus Gängen besteht, die Vielfache eines vorgegebenen Winkels $\alpha \in]0, 360]$ mit $n_\alpha := \frac{360}{\alpha} \in I\!N$ einschließen. Für $\alpha = 90°$ erhält man also die gleiche Annahme wie [CM93] und [BB⁺95].

Sei $(\alpha_k)_{k \in I\!N_0}$ die Folge der durch die Odometrie gemessenen Orientierungen des Fahrzeuges. Dann lassen sich jeweils die letzten L Messungen in einem Ringspeicher der Länge L speichern.

Da nach Annahme über die vorkommenden Orientierungen in der Umwelt nur vielfache Winkel von α auftreten, wird der Winkelbereich von 0° bis 360° in n_α Bereiche aufgeteilt und für jedes dieser Bereiche gezählt, wieviele der letzten L gemessenen Orientierungen in diesen Bereich gehören. Auf diese Weise bildet man das Histogramm der Häufigkeitsverteilungen der letzten L gemessenen Orientierungen über die Winkelbereiche. Formal werden diese Bereiche gegeben durch

$$\mathcal{J} := \left\{ \left[360 - \frac{\alpha}{2}, 360\right[\cup \left[0, \frac{\alpha}{2}\right[\right\} \cup \left\{ \left[\left(j - \frac{1}{2}\right) \cdot \alpha, \left(j + \frac{1}{2}\right) \cdot \alpha\right[\middle| j = 1, \cdots, n_\alpha - 1 \right\}. \quad (3.8)$$

Zu jedem Schritt $k \in I\!N_0$ definiert man die Histogrammfunktion durch

$$h_k : \mathcal{J} \to I\!N, \quad J \mapsto \sum_{i=0}^{\min\{L-1,k\}} \chi_J(\alpha_{k-i}). \tag{3.9}$$

Die Funktion χ_J bezeichnet hier die charakteristische Funktion zur Menge J. Dieses Histogramm, das in jedem Schritt k mit konstantem Aufwand aus dem Histogramm des vorherigen Schrittes $k-1$ bestimmt werden kann, gibt Auskunft über die momentane vorherrschende Richtung des Fahrzeuges. Jedes lokale Maximum im Histogramm zeigt an, daß das Fahrzeug in den letzten L Schritten mit einer vorherrschenden Richtung gefahren sein muß, dessen Richtung zum Maximum gehört. Fährt das Fahrzeug um eine Ecke und ändert seine vorherrschende Richtung, so bilden sich im Histogramm zwei lokale Maxima heraus. Ein Maximum gehört zur alten Richtung und bildet sich zurück. Das andere Maximum entsteht gerade durch die neue vorherrschende Richtung des Fahrzeuges. Eine Korrektur der Odometrie darf nur auf den Winkel des jüngsten lokalen Maximums im Histogramm eingestellt werden. Daher ist es nötig, das Alter eines Winkelbereichs im Histogramm festzulegen. Im $k$ten Schritt wird das Alter eines vor l Schritten gemessenen Winkels α_{k-l} durch l festgelegt. Das Alter eines Winkelbereichs J ist dann durch die Summe der Altersgrößen der Winkelmessungen angegeben, die in den letzten L Messungen zu J gehörten. Formal wird das Alter eines Winkelbereichs durch

$$a_k : \mathcal{J} \to I\!N_0, \quad J \mapsto \sum_{i=0}^{\min\{L-1,k\}} i \cdot \chi_J(\alpha_{k-i}). \tag{3.10}$$

definiert. Das jüngste lokale Maximum im Histogramm ist also ein lokales Maximum mit minimalem Alter. Dazu sei

$$M_k := \max\{h_k(J) | J \in \mathcal{J}\} \tag{3.11}$$

der maximale Wert des Histogramms im $k$ten Schritt und

$$\mathcal{M}_k := \{J \in \mathcal{J} \,| h_k(J) \geq \theta \cdot M_k\} \tag{3.12}$$

die Menge der lokalen Maxima, deren Werte größer oder gleich $\theta \cdot M_k$ für einen gegebenen Schwellwert $\theta \in [0,1]$ sind. Als jüngstes dieser lokalen Maxima erhält man den Winkelbereich J_k^* durch:

$$\forall J \in \mathcal{M}_k : a_k(J) \geq a_k(J_k^*). \tag{3.13}$$

Damit steht der Bereich fest, aus dem die momentan vorherrschende Richtung des Fahrzeuges stammt. Als eine gute Schätzung für die momentane Fahrtrichtung kann der Mittelpunkt des Bereichs J_k^* verwendet werden. Eine Ausnahme bildet die Menge $[360 - \frac{\alpha}{2}, 360[\cup [0, \frac{\alpha}{2}[$, die aus zwei nicht zusammenhängenden Intervallen besteht. In diesem Fall wird 0 als Schätzwert verwendet. Insgesamt ergibt sich β_k als Richtungsschätzung im $k$ten Schritt:

$$\beta_k := \begin{cases} 0, \text{ falls } J_k^* = \left[360 - \frac{\alpha}{2}, 360\right[\cup \left[0, \frac{\alpha}{2}\right[\\ j \cdot \alpha, \text{ falls } J_k^* = \left[(j - \frac{1}{2}) \cdot \alpha, \left(j + \frac{1}{2}\right) \cdot \alpha\right[, j \in \{1, \cdots, n_\alpha\} \end{cases} . \tag{3.14}$$

Um mit Hilfe dieses Schätzwertes die Odometrie zu rekalibrieren, ist es nötig, daß die Histogrammdaten verläßlich sind. Die Verläßlichkeit dieser Daten steigt je länger das Fahrzeug in einem geraden Gang fährt. Damit steigen die Histogrammwerte für einen Winkelbereich langsam an und bilden ein neues lokales Maximum. Erst wenn ausreichend viele Meßwerte in den Bereich J_k^* gehören, kann das lokale Maximum als sicher angenommen werden. Die ausreichende Anzahl wird durch einen Vergleich des Histogrammwertes mit einem Schwellwert N überprüft.
Dennoch genügt auch diese Überprüfung der Histogrammdaten nicht aus, um eine Rekalibrierung vorzunehmen. Man betrachte beispielsweise den Fall, daß das Fahrzeug eine Weile in einem Gang mit der Orientierung 0° gefahren ist. Im Histogramm hat sich dadurch ein lokales Maximum im Intervall um 0° herausgebildet. Fährt das Fahrzeug in eine Kurve und weichen die gemessenen Orientierungen α_k allmählich immer stärker von 0° ab, so darf keine Korrektur vorgenommen werden, da sonst diese bereits gefahrene Kurve nicht mehr als solche erkannt wird.
Um dieses Problem zu lösen, wird im folgenden davon ausgegangen, daß die Fehler in der Richtungsbestimmung durch die Odometrie deutlich geringer ausfallen, als gemessene Richtungsänderungen bei einer tatsächlichen Änderung der Fahrtrichtung. Man gibt eine maximal erwartete Abweichung σ der Meßwerte von der geschätzten Fahrtrichtung β_k vor. Damit können Richtungsänderungen aufgrund von Meßfehlern oder einer Kurvenfahrt unterschieden werden. Da die gesamte Methode auf dem Vorwissen beruht, daß in der Umgebung nur Gänge existieren, die untereinander vielfache Winkel von α einschließen, darf die Rekalibrierung erst dann stattfinden, wenn ein Gang mit seiner Orientierung sicher erkannt wurde. Um diese Sicherheit zu erhöhen wird zusätzlich gefordert, daß die Abweichung der gemessenen α_{k-l} von den geschätzten β_{k-l} kleiner als σ für alle $l \in \{0, \cdots, m\}$ ist. Damit wird sichergestellt, daß in den letzten m Schritten keine Kurve gefahren wurde und eine Rekalibrierung damit möglich ist.
Zusammenfassend wird die durch die Odometrie gegebene Fahrtrichtung im $k$ten Schritt durch die Schätzung β_k korrigiert, falls folgende beiden Bedingungen erfüllt sind:

1. Jüngstes Maximum ist ausreichend ausgeprägt: $h_k(J_k^*) \geq N$,
2. Keine Kurvenfahrt in den letzten m Schritten: $\forall \quad l \leq m : \quad |\alpha_{k-l} - \beta_{k-l}| \leq \sigma$.

Als geeignete Parameter haben sich in der Simulation die Werte aus Tabelle 3.3 ergeben.

α	L	θ	N	σ	m
22.5°, 30°, 45°, 90°	50, 100	0.7	10, 20	5.5°	3

Tabelle 3.3: Parameter der Selbstlokalisation

Die Wahl von α hängt stark von der Beschaffenheit der Umwelt ab. Besteht die Umwelt nur aus zueinander parallel und rechtwinklig stehenden Gängen, genügt eine Einteilung in vier Winkelbereiche mit $\alpha = 90°$. Besteht die Umgebung hingegen aus Gängen, die andere Winkel einschließen, so ist α entsprechend anzupassen.

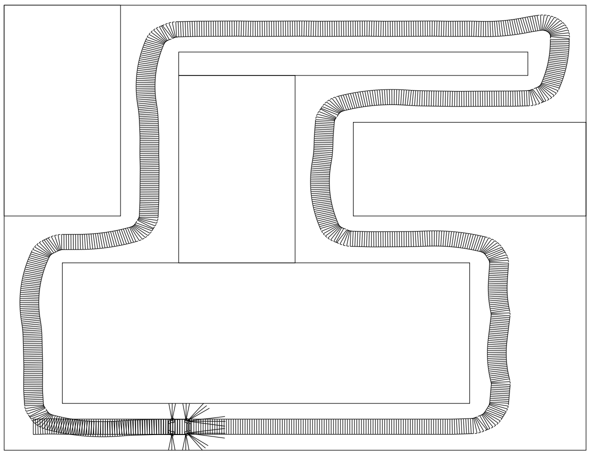


Abbildung 3.26: Gefahrene Wegstrecke

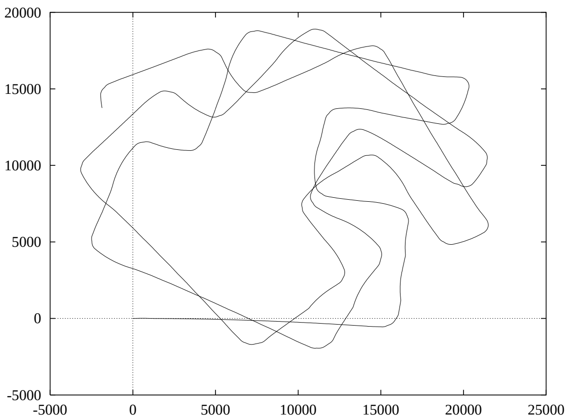


Abbildung 3.27: Aufgemessener Weg ohne Rekalibrierung

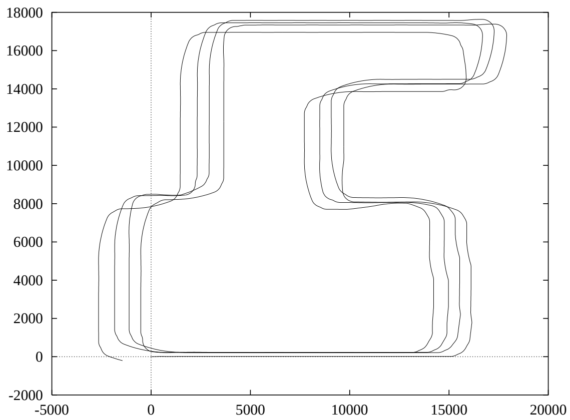


Abbildung 3.28: Aufgemessener Weg mit Rekalibrierung

Die Abbildungen 3.27 und 3.28 zeigen die Wirkung der Rekalibrierung. Die in Abbildung 3.26 dargestellte gefahrene Tour ist einem starken Fehler[7] der Odometrie unterworfen und gibt keine brauchbare Meßung der gefahrenen Strecke an (vgl. Abb. 3.27). In Abbildung 3.28 dagegen wurde die gleiche Strecke bei Verwendung der Rekalibrierung gefahren. Hier wird deutlich, daß die Winkel den tatsächlich gefahrenen Orientierungen entsprechen. Die geschlossenen Trajektorien in der Realität werden auch durch geschlossene Wegbahnen in der Positionsaufzeichnung beschrieben. Diese Daten können für den Aufbau einer Karte verwendet werden.

[7]Die in Abb. 3.27 und 3.28 dargestellten Versuche wurden in der Simulation durchgeführt. Dabei wurde der Fehler der Odometrie durch ein normalverteiltes Rauschen mit Erwartungswert 4° und einer Varianz von 6° simuliert. Damit lassen sich in der Simulation realistische Fehler beobachten.

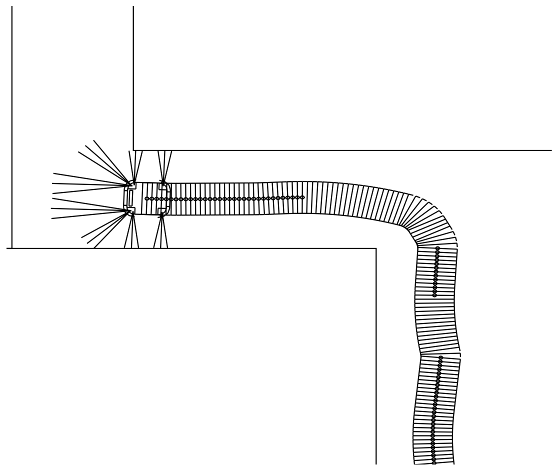


Abbildung 3.29: Fahrt des Roboters durch eine Kurve

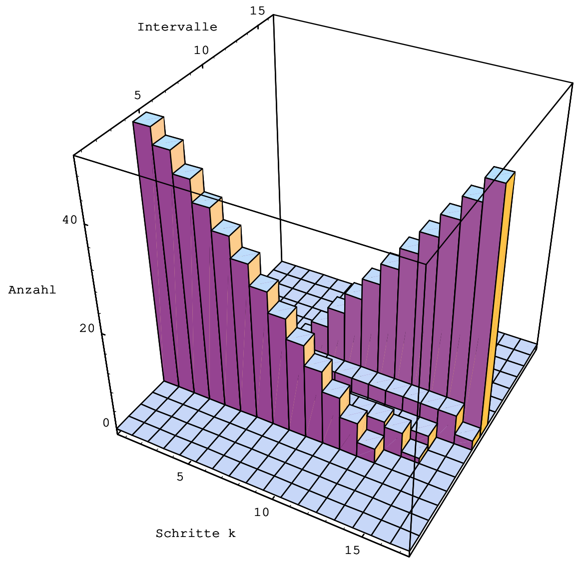


Abbildung 3.30: Zeitlicher Verlauf der Histogramme

In der Abbildung 3.29 ist eine Fahrt des AMR's durch eine Kurve dargestellt. In der Abbildung 3.30 sind mehrere Histogramme in Abhängigkeit der Schrittzahl während der Fahrt durch die Kurve abgebildet. Dabei ist deutlich zu erkennen, wie sich der Wechsel zwischen zwei vorherrschenden Richtungen in der Folge der Histogramme darstellt. Vor Beginn der Kurvenfahrt wird die vorherrschende Richtung durch ein lokales Maximum bei ca. 90° (Intervall Nr. 4) repräsentiert. Dieses geht allmählich in ein Maximum bei 180° (Intervall Nr. 8) über, nachdem das Fahrzeug die Kurve verlassen hat. Die Striche in der Mitte der Fahrbahn des Roboters geben jeweils den Zeitraum an, in dem eine Winkelkorrektur stattgefunden hat. Der Bereich der Kurve wird dabei ausgelassen.

Bisher wurde nur die Orientierung des Roboters korrigiert. Dieses ist die größte Fehlerquelle und beeinflußt auch die gemessene absolute Position des Fahrzeuges am stärksten. Der Fehler in der Positionsbestimmung kann allerdings nicht vollständig durch eine Winkelkorrektur behoben werden. Für eine zusätzliche Korrektur der x- und y-Koordinaten des Fahrzeuges werden Positionsmarken benötigt, die in einer Karte vermerkt werden. Diese Karte mit den Positionsmarken kann zur zusätzlichen Rekalibrierung der Odometrie verwendet werden.

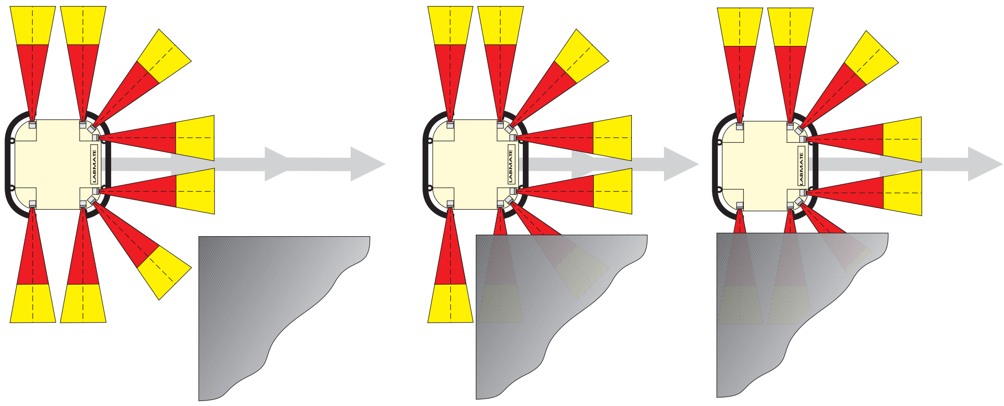
3.2.2 Kartographierung

Unter Ausnutzung des Vorwissens über die Umgebung wird eine ausreichende Selbstlokalisation des AMR's gewährleistet. Die gesammelten Positionsdaten werden für die Kartographierung so gespeichert, daß sie für weitere Aufgaben, wie z. B. der Wegplanung und der Wegoptimierung, genutzt werden können.

Dabei werden keine Daten über die Hindernisse in der Umwelt in die Karte eingetragen, wie es z. B. bei [BB+95] oder auch bei [Hop92] geschieht. Insbesondere bei [Hop92] wird jedes durch einen Laserradar erkannte Hindernis möglichst exakt in ein Karte eingetragen, um so auch die Hindernisvermeidung aufgrund der Karteninformationen vornehmen zu können. Dieses ist hier jedoch nicht notwendig, da das zuvor entworfenen Hindernisvermeidungssystem ein rein reaktives System ist. Es kommt daher ohne eine aufwendig erstellte Umweltrepräsentation aus und lenkt den Roboter kollisionsfrei durch die Umwelt. Stattdessen genügt hier eine einfache Repräsentation von markanten Positionen in der Umwelt und einer Verbindungsstruktur zwischen diesen Marken. Die Verbindungen kennzeichnen, ob der AMR durch ausschließliche Verwendung seines reaktiven Hindernisvermeidungsverhaltens von einer Position zur nächsten gelangen kann. Damit kann mit geringem Aufwand ein Wegeplan erstellt werden. In der Wegplanungsphase wird der Wegeplan dazu verwendet, optimale Touren zwischen zwei Punkten zu berechnen.

Bestimmung von Wegmarken

Um die Konstruktion eines Wegegraphens speicherplatzsparend zu gestalten, ist zunächst das Problem zu lösen, welche Positionen als Knoten in den Graphen einzutragen sind. Um Redundanz in der Karte zu vermeiden, dürfen nicht zu viele Positionen im Graphen vermerkt werden. Allerdings sind solche Positionen zu speichern, die für eine spätere Wegverfolgung durch den AMR nötig sind.
Da Gänge ohne Abzweigungen für die Hindernisvermeidung problemlos sind, sind Informationen über solche Umgebungen unnötig.
Schwieriger ist es, wenn die Gänge verzweigt sind oder in Kreuzungen einmünden, da sich dort die Frage stellt, in welche Richtung der AMR weiterfahren soll. Dieses wird im Normallfall nur von der Wegplanungskomponente bestimmt. Somit müssen Kreuzungen und Verzweigungen im Wegegraphen durch Knoten markiert werden. Verzweigte Wege werden durch Kanten im Graphen repräsentiert. In künstlichen Umgebungen, wie beispielsweise in Gebäuden mit Zimmern und Gängen, sind solche markanten Positionen durch Wände umgeben, die teilweise scharfe Ecken aufweisen. Daher genügt es, nach solchen Ecken zu suchen, und diese als markante Punkte zu vermerken. Hier müssen derartige markante Punkte mit Ultraschallsensoren in der Umgebung gefunden werden. Dazu wird versucht, die scharfen Ecken in den umgebungsbegrenzenden Wänden zu ermitteln und dadurch auf Kreuzungen, Ecken oder Verzweigungen zu schließen. Eine ähnliche Idee wird z. B. in [CM93] angegeben, wo Lücken in der Umgebungsbegrenzung detektiert werden.
Mit der in Abbildung 3.2 a) angegebenen Sensorkonfiguration wird versucht, Ecken in der Umwelt dadurch zu finden, daß die Sensoren #4, #5, #6 und #7 einen bestimmten Signalverlauf anzeigen, der auf Unstetigkeiten im Seitenabstand der Hindernisse zum Fahrzeug schließen läßt. Dazu betrachte man die Abbildung 3.31, in der drei aufeinanderfolgende Situationen dargestellt sind, die sich ergeben, falls der AMR an einer Ecke vorbeifährt. In der ersten Phase zeigen die Sensorwerte der Seitensensoren bis auf Rauscheffekte keine größeren Schwankungen. Beim Vorbeifahren an einer Ecke verringert sich erst der Meßwert des vorderen Seitensensors und dann der des hinteren Seitensensors sprunghaft. In der dann folgenden dritten Phase schwanken die Meßwerte der Seitensensoren kaum noch.
Durch Beobachtung solcher sprunghaften Veränderungen in den Meßwerten der Seitensensoren können Ecken ermittelt werden. Dazu werden zusätzlich zu den



a) b) c)

Abbildung 3.31: Drei Phasen der Wegmarkendetektion

in $sensor[4], \cdots, sensor[7]$ gespeicherten aktuellen Meßwerten alte Meßwerte in $sensor_{old}[4], \cdots, sensor_{old}[7]$ gespeichert, die im vorherigen oder einem noch früheren Kontrollzyklus aktuell waren. Ob eine sprunghafte Veränderung der Sensorwerte stattgefunden hat, wird durch Untersuchung der Differenz aus alten und aktuellen Sensorwerten ermittelt. Dazu werden je zwei Schwellwerte $\eta_{low}[j], \eta_{high}[j] \in I\!N$ mit $\eta_{low}[j] < \eta_{high}[j]$ eingeführt. Falls $|sensor[j] - sensor_{old}[j]| \leq \eta_{low}[j]$, ist die Änderung im Sensorwert $j \in \{4, 5, 6, 7\}$ so klein, daß sie nicht als Sprungstelle interpretiert wird. Eine Ecke wird dagegen angenommen, falls $|sensor[j] - sensor_{old}[j]| \geq \eta_{high}[j]$ gilt. Eine sprunghafte Veränderung der Sensorwerte auf der linken Seite wird durch das Prädikat P_L und eine Sprungstelle in der Umgebung auf der rechten Fahrzeugseite durch P_R in (3.15) angegeben. Befindet sich hingegen das Fahrzeug im freien Raum, wird keine Sprungstelle detektiert, da alle Sensorwerte die maximal meßbare Entfernung angeben. Bei der Detektion einer Sprungstelle ist also das Fahrzeug in der Nähe eines Hindernisses. Durch weitere Abfragen der Sicherheitszonen der Sensoren #4 bzw. #7 wird ermittelt, ob der obere oder der untere Seitensensor ein Hindernis anzeigt. Dadurch können die linken und rechten Ecken weiter in untere und obere Ecken untergliedert werden. Das Ergebnis dieser Analyse wird durch P_{LU}, P_{RU}, P_{LO} und P_{RO} bestimmt. Ob die Sensoren überhaupt eine Sprungstelle in der Umgebung entdecken, wird schließlich durch P in (3.15) festgelegt.

$$
\begin{aligned}
P_L &:= ((|sensor[7] - sensor_{old}[7]| > \eta_{high}[7]) \wedge \\
&\quad\; (|sensor[6] - sensor_{old}[6]| \leq \eta_{low}[6])) \vee \\
&\quad\; ((|sensor[6] - sensor_{old}[6]| > \eta_{high}[6]) \wedge \\
&\quad\; (|sensor[7] - sensor_{old}[7]| \leq \eta_{low}[7])), \\
P_R &:= ((|sensor[4] - sensor_{old}[4]| > \eta_{high}[4]) \wedge \\
&\quad\; (|sensor[5] - sensor_{old}[5]| \leq \eta_{low}[5])) \vee \\
&\quad\; ((|sensor[5] - sensor_{old}[5]| > \eta_{high}[5]) \wedge \\
&\quad\; (|sensor[4] - sensor_{old}[4]| \leq \eta_{low}[4])), \\
P_{LU} &:= P_L \wedge (sensor[7] \leq safezone[7]),
\end{aligned}
\tag{3.15}
$$

$$\begin{aligned} P_{LO} &:= P_L \wedge (sensor[6] \leq safezone[6])\,, \\ P_{RU} &:= P_R \wedge (sensor[4] \leq safezone[4])\,, \\ P_{RO} &:= P_R \wedge (sensor[5] \leq safezone[5])\,, \\ P &:= P_{LU} \vee P_{LO} \vee P_{RU} \vee P_{RO}\,. \end{aligned}$$

Ist $P = true$, wird in die Karte an der momentanen Position eine Wegmarke als Knoten des Wegegraphen eingetragen. Die Wegmarke wird mit einer Signatur gemäß den Prädikaten aus (3.15) versehen und erhält zusätzlich die momentane Orientierung des AMR's (vgl. Abb. 3.32).

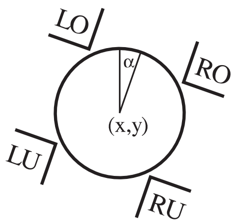


Abbildung 3.32: Wegmarke an der Position (x, y) mit Signaturen LU, LO, RU, RO und Orientierung α.

Wegegraphen

Eine *Wegmarke* wird als ein Element der folgenden Knotenmenge eingeführt:

$$V \subset \{v = (x_v, y_v, \alpha_v, s_v) | x_v, y_v \in I\!R, \alpha_v \in [0^\circ, 360^\circ[, s_v \in I\!B^4\}. \tag{3.16}$$

Dabei gibt (x_v, y_v) die Position, α_v die Orientierung und s_v eine Boolesche Belegung von $(P_{LU}, P_{RU}, P_{LO}, P_{RO})$ für einen Knoten $v \in V$ an.
Die Kanten des Wegegraphen werden dann als zweielementige Knotenmengen der verbundenen Knoten repräsentiert:

$$E \subset V^{(2)} := \{\{v, w\} | v, w \in V \wedge v \neq w\}\,. \tag{3.17}$$

Für die spätere Wegoptimierung wird eine Kantenmarkierung κ eingeführt, die jeder Kante den Euklidischen Abstand der durch sie verbundenen Knoten zuordnet.

$$\kappa : E \to I\!N, \quad \{v, w\} \mapsto \left\| \begin{pmatrix} x_v \\ y_v \end{pmatrix} - \begin{pmatrix} x_w \\ y_w \end{pmatrix} \right\|. \tag{3.18}$$

Das Tripel $W = (V, E, \kappa)$ wird dann als *Wegegraph* bezeichnet.

Aufbau eines Wegegraphen

Innerhalb der Explorationsphase, in der der Roboter seine Umgebung erkundet, werden gemäß der oben beschriebenen Wegmarkendetektion von Zeit zu Zeit Wegmarken $v_1, v_2, v_3, \cdots$ erkannt. Mit jeder neuen Wegmarke kann der momentane Wegegraph ergänzt werden.
Zu Beginn ($k = 0$) ist der Wegegraph der leere Graph und enthält noch keine Wegmarken.

$$\text{Schritt 0: } W_0 := (V_0, E_0, \kappa_0), \quad V_0 := E_0 := \kappa_0 := \emptyset. \tag{3.19}$$

Die erste Wegmarke v_1, die in der Umgebung erkannt wird, wird als erster Knoten direkt dem Graphen zugefügt.

$$\text{Schritt 1: } W_1 := (V_1 := \{v_1\}, E_1, \kappa_1), \quad E_1 := \kappa_1 := \emptyset, \ell_1 := v_1. \tag{3.20}$$

Die Kantenmenge ist im ersten Schritt noch leer.
Nachdem die Knotenmenge des Wegegraphens nicht mehr leer ist, wird der weitere Aufbau des Graphen für den Schritt $k+1$ betrachtet. In diesem Schritt ist der alte Wegegraph durch $W_k = (V_k, E_k, \kappa_k)$ gegeben. Der bis zum $k$ten Schritt zuletzt in den Wegegraphen W_k eingefügte Knoten wird mit ℓ_k bezeichnet. Im Schritt $k+1$ wird eine Wegmarke $v_{k+1} = (x_{k+1}, y_{k+1}, \alpha_{k+1}, s_{k+1})$ an der Position (x_{k+1}, y_{k+1}) mit der Orientierung α_{k+1} und der Signatur s_{k+1} erkannt.
Im folgenden werden zum Aufbau des Wegegraphen im Schritt $k+1$ drei Fälle unterschieden.
Falls die Wegmarke v_{k+1} einen Abstand zur zuletzt eingefügten Marke ℓ_k besitzt, der kleiner als ein vorgegebenes $\delta_0 > 0$ ist, so wird davon ausgegangen, daß die Detektion von v_{k+1} durch die gleiche Stelle in der Umgebung ausgelöst wurde, wie die Detektion der zuvor eingefügten Wegmarke ℓ_k. Somit ist diese Stelle der Umgebung im Graphen bereits verzeichnet und der Wegegraph wird im Schritt $k+1$ deshalb nicht verändert.

$$\begin{array}{ll} \text{Schritt } k+1: & W_{k+1} := W_k \wedge \ell_{k+1} := \ell_k, \\ \text{(1. Fall)} & \text{falls } \left\| \begin{pmatrix} x_l \\ y_l \end{pmatrix} - \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} \right\| \le \delta_0 \end{array} \tag{3.21}$$

Dabei geben x_l und y_l die Position der zuletzt eingefügten Wegmarke ℓ_k an.
Im zweiten Fall ist dagegen der Abstand der detektierten Wegmarke v_{k+1} zur zuletzt eingefügten Wegmarke ℓ_k größer als δ_0 und der Abstand von v_{k+1} zu allen anderen Wegmarken in V_k ist größer als ein vorgegebenes $\delta_1 > 0$. Dann wird v_{k+1} in den neuen Graphen aufgenommen. Ferner wird eine neue Verbindung zwischen ℓ_k und v_{k+1} hergestellt.

$$\begin{array}{ll} \text{Schritt } k+1: & W_{k+1} := (V_k \cup \{v_{k+1}\}, E_k \cup \{e_{k+1}\}, \kappa_k \cup \{(e_{k+1}, d_{k+1})\}), \\ \text{(2. Fall)} & \text{falls } d_{k+1} := \left\| \begin{pmatrix} x_l \\ y_l \end{pmatrix} - \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} \right\| > \min\{\delta_0, \delta_1\}, \\ & \text{wobei } e_{k+1} := \{\ell_k, v_k\}, \ell_{k+1} := v_{k+1}. \end{array} \tag{3.22}$$

Der Wert von δ_1 gibt an, bis zu welchem Abstand zwischen zwei Wegmarken diese als verschieden im Wegegraphen behandelt werden.

Im dritten und letzten Fall hat v_{k+1} zur zuletzt eingefügten Wegmarke ℓ_k einen Abstand, der größer als δ_0 ist. Im Gegensatz zum zweiten Fall gibt es in diesem Fall eine Wegmarke $v_j \in V_k \setminus \{\ell_k\}$ deren Abstand zu v_{k+1} kleiner als δ_1 ist. Die neu detektierte Wegmarke v_{k+1} liegt derart nah an der alten schon im Wegegraphen befindlichen Wegmarke v_j, daß diese beiden im künftigen Graphen als eine Wegmarke behandelt werden sollen. Die Wegmarke, die anstelle von v_j und v_{k+1} in den neuen Wegegraphen eingefügt wird, erhält die Position von v_j und eine neue Signatur $\mu(v_j, v_{k+1})$, die aus den alten Signaturen und Orientierungen von v_j und v_{k+1} gewonnen wird.
Da das Fahrzeug offenbar von Wegmarke ℓ_k nach v_j gefahren ist, wird eine neue Kante zwischen ℓ_k und der neuen Wegmarke in den Graphen eingefügt.

$$\begin{aligned}
\text{Schritt } k+1: \quad & W_{k+1} := ((V_k \setminus \{v_j\}) \cup \{v\}, \tilde{E}_k \cup \{e_{k+1}\}, \tilde{\kappa}_k \cup \{(e_{k+1}, d_{k+1})\}), \\
\text{(3. Fall)} \quad & \text{falls } \exists v_j \in V_k \setminus \{\ell_k\} : d_{k+1} := \left\| \begin{pmatrix} x_j \\ y_j \end{pmatrix} - \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} \right\| \leq \delta_1 \qquad (3.23) \\
& \text{und } \left\| \begin{pmatrix} x_l \\ y_l \end{pmatrix} - \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} \right\| > \delta_0, \\
& \text{wobei } v := (x_j, y_j, \alpha_j, \mu(v_j, v_{k+1})), e_{k+1} := \{\ell_k, v\}, \ell_{k+1} := v.
\end{aligned}$$

Dabei geben (x_j, y_j) die Position und α_j die Orientierung der alten Wegmarke v_j an. Die Kantenmenge $\tilde{E}_k$ geht aus der alten Kantenmenge E_k durch Ersetzen von v_j durch v hervor. Entsprechend geht die Kantenmarkierung $\tilde{\kappa}_k$ durch die gleiche Ersetzung aus κ_k hervor.
Es bestand die Hoffnung, mit Hilfe der Signaturen die Struktur der Umgebung ausreichend im Wegegraphen beschreiben zu können. Es sollte an Wegmarken darauf geschlossen werden, ob sich in der nahen Umgebung der Wegmarke eine bestimmte Ecke, eine Tür oder eine Kreuzung befindet. Um diese Informationen zu erhalten, haben sich die Signaturen als kein tragfähiges Konzept herausgestellt. Zum einen sind die in den Signaturen abgespeicherten Informationen sehr gering und zum anderen reichen auch die Ultraschallmessungen nicht aus, um die Struktur der Umgebung genau zu analysieren. Auf die Verarbeitung der Signaturen wird daher nicht weiter eingegangen.
Für Experimente im Simulator mit der Kartographierung wurden folgende Parameter verwendet.

δ_0	δ_1	$\eta_{low}[j],\ j \in \{1,2,3,4\}$	$\eta_{high}[j],\ j \in \{1,2,3,4\}$
800 mm	400 mm	50 mm	80 mm

Tabelle 3.4: Parameter der Kartographierung

Selbstlokalisation mit einer Karte

Nachdem eine Karte in Form eines Wegegraphen zur Verfügung steht und die Orientierungen bereits erfolgreich korrigiert wurden, soll nun auch der mögliche Positionsfehler in der Odometrie durch Verwendung der Karteninformationen ausgeglichen werden.

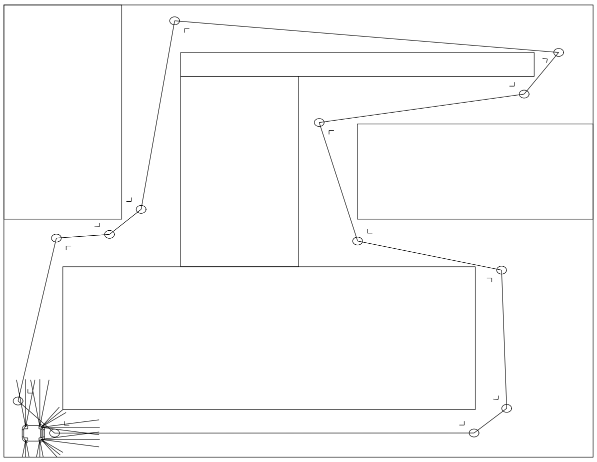


Abbildung 3.33: Beispiel eines Wegegraphen nach einer Exploration

Dieses kann durch einen einfachen Ansatz gewährleistet werden. Im Falle einer mehrfachen Detektion einer Wegmarke (3. Fall im Aufbau des Wegegraphen) wird die Position der alten Wegmarke zur Positionskorrektur der Odometrie herangezogen, d. h. die durch die Odometrie angegebene Position wird auf die Position der identifizierten Wegmarke gesetzt. Zusammen mit der Winkelkorrektur erhält man ein in Abbildung 3.35 dargestelltes Resultat. Ein Versatz in der aufgemessenen Tour gibt die Position an, bei der eine alte Wegmarke identifiziert wird und die Position der Odometrie korrigiert wird.

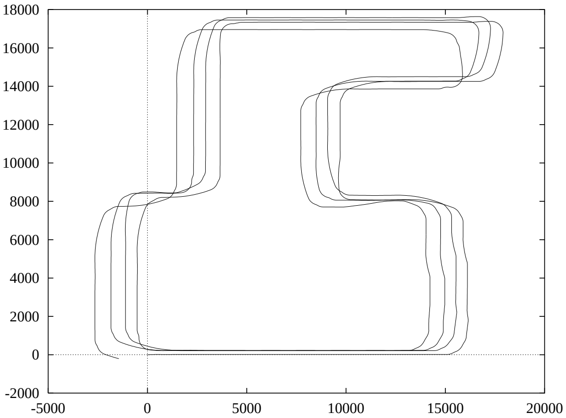


Abbildung 3.34: Aufgemessener Weg nur mit Winkelkorrektur

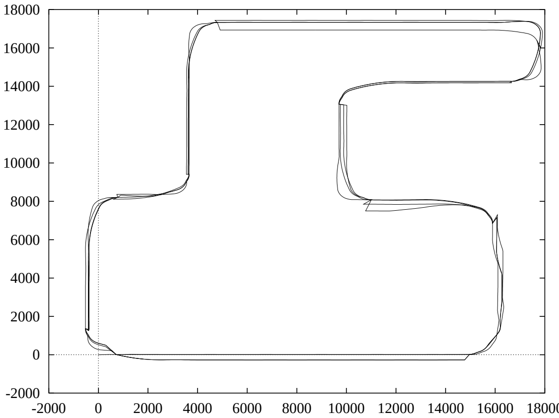


Abbildung 3.35: Aufgemessener Weg mit Winkel- und Positionskorrektur

3.3 Wegplanung

Nach einer erfolgreichen Exploration der Umgebung hat der AMR eine Karte seiner Umwelt aufbauen können. Sie besteht aus markanten Wegmarken und deren Verbindungen, so daß die Strecke zwischen verbundenen Wegmarken durch die Hindernisvermeidung des Roboters gefahrlos gefahren werden kann.
Damit das System einfache Transportaufgaben bewältigen kann, ist es jedoch nötig, beliebige Orte zu erreichen. Um bei dieser Aufgabe nicht an Sackgassen zu scheitern, wird die zuvor erstellte Karte herangezogen. In dieser Karte wird ein möglichst kurzer Weg bestimmt, der schließlich vom Roboter durch aufeinanderfolgendes Ansteuern der Wegmarken gefahren wird.
Die Bestimmung eines optimalen Weges bei gegebener Karte und gegebenem Start- und Zielpunkt wird als Wegplanung bezeichnet. Da die hier verwendete Karte als Graph dargestellt ist, stehen alle bekannten Algorithmen zur Suche kürzester Wege in Graphen zur Verfügung.
Bereits 1959 hat Dijkstra [Dij59] einen Algorithmus vorgeschlagen, der in ungerichteten kantenbewerteten Graphen kürzeste Wege von einem Knoten zu allen anderen Knoten im Graphen ermittelt. Dieser Algorithmus wird hier behandelt, da er sich für den hier verwendeten Wegegraphen und die hier gestellte Aufgabe anbietet. Weitere Algorithmen über kürzeste Wege in Distanzgraphen werden z. B. in [OW90] besprochen.

Vorgegeben wird ein Zielknoten $v_{goal} \in V$ des Wegegraphens, zu dem von jedem Knoten des Graphens ein kürzester Weg bestimmt werden soll.
Dijkstra unterscheidet dazu drei Klassen von Knoten. Die erste Klasse wird als Klasse der *erreichten Knoten* bezeichnet. Sie besteht aus den Knoten, für die ein optimaler Weg zum Zielknoten bereits bekannt ist. Die zweite Klasse besteht aus *Randknoten*, die zu den erreichten Knoten benachbart sind, selbst bisher aber nicht in die Klasse der erreichten Knoten gehören. Die dritte Klasse besteht aus den *unerreichten Knoten*, die weder Randknoten, noch erreichte Knoten sind.
Zu Beginn ist v_{goal} der einzige erreichte Knoten. Die zu ihm benachbarten Knoten bilden den Rand. Zu jedem Knoten werden zwei zusätzliche Informationen vermerkt. Die erste Information besteht aus der Angabe, ob ein Knoten erreicht ist. Die zweite Information besteht aus dem minimalen Abstand δ_v eines Knotens $v \in V$ zum Zielknoten v_{goal}. Dieser minimale Abstand wird für alle $v \in V \setminus \{v_{goal}\}$ mit $\delta_v := \infty$ initialisiert. Ein unerreichter Knoten $u \in V$ wird zu einem Randknoten, wenn er zu einem ehemaligen Randknoten benachbart ist. Sein minimaler Abstand zum Zielpunkt ermittelt sich durch

$$\delta_u := \min\left\{ \delta_v + \left\| \begin{pmatrix} x_v \\ y_v \end{pmatrix} - \begin{pmatrix} x_u \\ y_u \end{pmatrix} \right\| \, v \in V \text{ Randknoten} \wedge \{v,u\} \in E \right\}, \tag{3.24}$$

da jeder Weg von einem unerreichten Knoten zum Zielpunkt den Rand durchlaufen muß. Ausgehend von den erreichten Knoten können die Randknoten in erreichte Knoten umgewandelt werden. Dazu genügt es, alle erreichten Nachbarn des Randknoten mit minimalem Abstand zum Zielpunkt zu betrachten und den minimalen Abstand dieses Randknoten v

zum Ziel gemäß

$$\delta_v := \min\left\{ \delta_w + \left\| \begin{pmatrix} x_v \\ y_v \end{pmatrix} - \begin{pmatrix} x_w \\ y_w \end{pmatrix} \right\| w \in V \text{ erreicht} \wedge \{v, w\} \in E \right\} \tag{3.25}$$

zu aktualisieren. Damit steht der minimale Abstand von v zum Zielknoten fest, d. h. v ist ein erreichter Knoten. Dieses Verfahren wird solange wiederholt, bis jeder Knoten ein erreichter Knoten ist. Ein optimaler Weg findet sich, indem ausgehend von einem beliebigen Startknoten immer die Kante durchlaufen wird, die zu einem Nachbarn führt, der einen minimalen Abstand zum Zielpunkt hat.

```
function calcDistances(v_goal ∈ V)
    { Initialisierung }
    δ_{v_goal} := 0;
    r_{v_goal} := true;
    for v ∈ V \ {v_goal} do
        δ_v := ∞;
        r_v := false;
    od;
    r := 1;
    {Berechne Rand von v_goal }
    for v ∈ {w ∈ V|{v_goal, w} ∈ E} do δ_v := ‖(x_v, y_v) − (x_{v_goal}, y_{v_goal})‖;
    while r < |V| do
        { Suche nächstgelegenenen Randknoten v_closest }
        v_closest := arg min {δ_v |v ∈ V ∧ r_v = false ∧ ∃w ∈ V : r_w = true ∧ {w, v} ∈ E};
        if δ_{v_closest} < ∞ then
            { Füge v_closest zu den erreichten Knoten hinzu }
            r := r + 1;
            r_{v_closest} := true;
            δ_{v_closest} := min{ δ_w + ‖(x_{v_closest}, y_{v_closest}) − (x_w, y_w)‖ | r_w = true∧ {v_closest, w} ∈ E } ;
            { Ergänze den Rand um die Nachbarn von v_closest }
            for v ∈ {w ∈ V |{v_closest, w} ∈ E} do
                δ_v := min{δ_v, δ_{v_closest} + ‖(x_v, y_v) − (x_{v_closest}, y_{v_closest})‖};
            od;
        else
            {Graph ist nicht zusammenhängend }
        fi;
    od;
end;
```

Abbildung 3.36: Berechnung eines kürzesten Weges

Mit dem Algorithmus aus Abbildung 3.36 wird ein optimaler Weg im Wegegraphen bestimmt. Dabei zählt r die Anzahl der erreichten Knoten. Der Wert von r_v gibt an, ob ein Knoten v bereits erreicht ist. Die hier dargestellte Variante des Dijkstra-Algorithmus entspricht einer Version ohne explizite Speicherung der Randknoten und besitzt einen

Zeitaufwand von $O(|V|^2)$ Schritten. Dieses kann durch folgende Überlegung eingesehen werden. Die Initialisierung benötigt $O(|V|)$ Schritte. Die *while*-Schleife wird $O(|V|)$ mal durchlaufen, da in jedem Schritt r um 1 erhöht wird. Da jeder Knoten höchstens $|V|$ Nachbarn besitzt, werden innerhalb der *while*-Schleife ebenfalls $O(|V|)$ Schritte durchgeführt. Weitere Verfeinerungen dieses Algorithmus, die die Randknoten in einem Fibonacci-Heap abspeichern, erreichen sogar einen Zeitaufwand von nur $O(|E| + |V| \log |V|)$. Für weitere Einzelheiten dazu sei auf [OW90] verwiesen.
Als nächstes stellt sich nun die Frage, wie dieser optimale Weg in eine Bewegung des Fahrzeuges umgesetzt werden kann.
Befindet sich das Fahrzeug an der Position (x_{LAB}, y_{LAB}) und ist die Zielposition durch (x_{goal}, y_{goal}) gegeben, so werden zunächst solche Knoten im Wegegraphen bestimmt, die zu diesen Positionen einen minimalen Euklidischen Abstand besitzen. Diese werden hier mit $v_{LAB} \in V$ und $v_{goal} \in V$ bezeichnet.
Ein erster naiver Ansatz steuert das Fahrzeug direkt auf die Wegmarke v_{LAB} zu, um von dort entlang der Knotenfolge des optimalen Weges schließlich zu v_{goal} zu gelangen. Der Nachteil dieses Ansatzes besteht darin, daß das Fahrzeug zwischen v_{LAB} und einem zu v_{LAB} benachbarten Knoten stehen kann, der ebenfalls zu der optimalen Tour gehört. Diese Strategie führt dazu, daß der AMR einen Umweg über v_{LAB} fährt, obwohl der Nachbar von v_{LAB}, der auch auf der optimalen Tour liegt, ebenso direkt zu erreichen ist. Um dieses zu verhindern, wird zu Beginn nicht direkt v_{LAB} angesteuert, sondern der AMR zu einer Wegmarke $v^* \in \{v_{LAB}\} \cup \{v \in V | \{v, v_{LAB}\} \in E\}$ geführt, so daß die Entfernung zum Zielknoten v_{goal}

$$s = \left\| \begin{pmatrix} x_{v^*} \\ y_{v^*} \end{pmatrix} - \begin{pmatrix} x_{v_{LAB}} \\ y_{v_{LAB}} \end{pmatrix} \right\| + \delta_{v^*} \tag{3.26}$$

minimal ist. Es ist nicht sinnvoll, weitere Knoten als die Nachbarn von v_{LAB} in diese Suche mit einzubeziehen, da für solche Knoten nicht mehr gewährleistet werden kann, daß die Hindernisvermeidung den AMR problemlos dorthin steuert.
Wenn das Fahrzeug nun zu einem Knoten des Wegegraphen gesteuert wurde, wird durch δ_v jeweils der nächstbeste Knoten ermittelt und das Fahrzeug dorthin gesteuert. Auf diese Weise gelangt der Roboter schließlich zu v_{goal}. Von dort aus wird ohne weitere Kartennavigation versucht, direkt auf den Zielpunkt (x_{goal}, y_{goal}) zuzusteuern.
Das Ansteuern eines nahegelegenen Zielpunktes wird von der Hindernisvermeidung übernommen. Statt des Wandfolgeverhaltens wird dazu das Zielfolgeverhalten verwendet. Ein Etappenziel oder auch das Endziel wird dann als erreicht angesehen, wenn sich das Fahrzeug bis auf eine Toleranzentfernung genähert hat.
Schwieriger wird es, wenn die Umwelt nicht nur aus statischen Hindernissen besteht, deren Lage unveränderlich ist, sondern zusätzlich neue Hindernisse entstehen oder bestehende Hindernisse verschwinden. Neue Hindernisse können dazu führen, daß zuvor freie Wege versperrt werden und dadurch die Karte nicht mehr mit der Umwelt übereinstimmt. In diesem Falle versucht der Roboter, entlang einer nicht mehr gültigen Kante des Wegegraphen zu einer Wegmarke zu gelangen. Aufgrund der Hindernisvermeidung wird das Fahrzeug allerdings vom neuen Hindernis abgelenkt und erreicht in der Regel die anzusteuernde Wegmarke nicht. Für diesen Fall muß die Kante, die jetzt erfolglos verfolgt wird, gelöscht werden, um dann nach neuen Wegen im geänderten Wegegraphen zu suchen. Eine

Möglichkeit, solche Kanten zu löschen, besteht darin, während der Fahrt entlang einer Kante einen *Frustrationszähler* f zu erhöhen. Wenn dieser Zähler ein gewisses Limit[8] f_{Limit} erreicht, wird davon ausgegangen, daß das Etappenziel nicht zu erreichen ist. Die Kante wird dann gelöscht. Anschließend wird die Wegplanung bei gleichem Ziel aber geänderter Fahrzeugposition wiederholt.
Schwieriger ist mit Umweltänderungen umzugehen, die auf neu entstehenden Freiräumen basieren. Solche Freiräume, die evtl. kürzere Strecken ermöglichen, sind nur durch eine erneute Exploration der Umgebung zu finden, da die Zielverfolgung sich nicht an Wänden, sondern nur an bekannten Wegmarken orientiert.
Weitere Erweiterungsmöglichkeiten der Navigation durch die hier dargestellten Wegegraphen bestehen in einer nachträglichen Optimierung des Graphen. Darin werden Knotengruppen zusammengefaßt, die offensichtlich nicht durch Hindernisse getrennt sind und dicht beieinanderliegen.

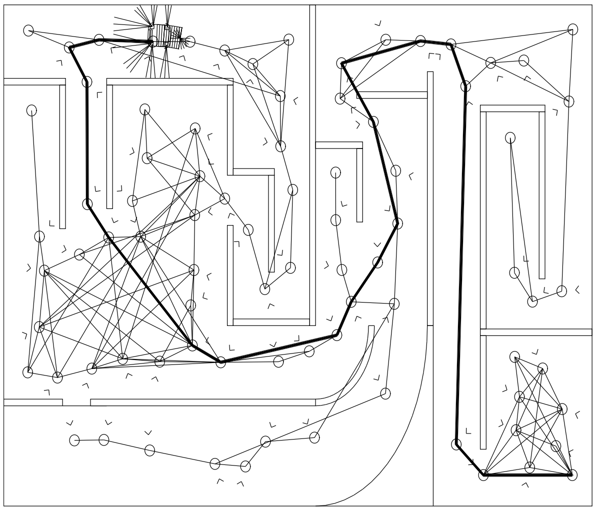


Abbildung 3.37: Planung eines Weges im Wegegraphen

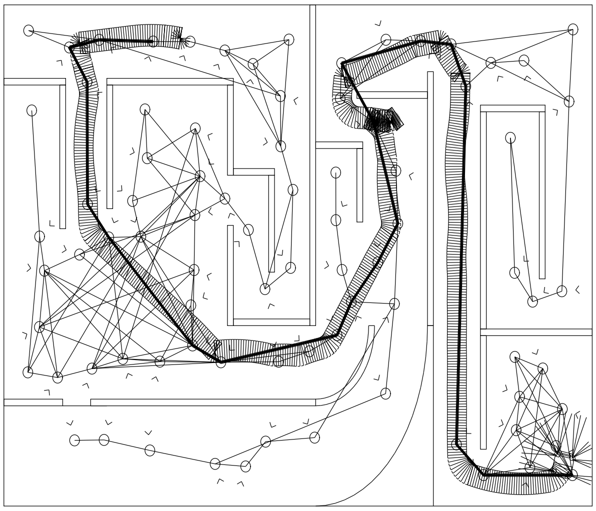


Abbildung 3.38: Verfolgung des geplanten Weges

Eine letzte Erweiterung, die sich als sinnvoll herausgestellt hat, ist die Neuorientierung nach Erreichen einer Wegmarkierung. Teilweise schließen zwei Kanten, die auf dem berechneten Weg liegen und mit einer Wegmarke inzidieren, einen derart spitzen Winkel ein, daß die Zielverfolgung nicht in der Lage ist, dem vorgegebenen Weg zu folgen, sondern durch Hindernisse abgetrieben wird. In diesem Fall ist bei Erreichen einer Wegmarke das Fahrzeug zu stoppen und in die Richtung der aus der Wegmarke auslaufenden Kante zu drehen. Die Fahrt kann dann fortgesetzt werden. In den Versuchen mit dem *LABMATE*-Simulator wurde eine Neuorientierung eingeleitet, wenn der Winkel zwischen den Kanten kleiner als

[8]Die Wahl von f_{Limit} hängt von dem zu erwartenden maximalen Abstand zwischen zwei benachbarten Wegmarken ab, da dem Fahrzeug die Möglichkeit erhalten bleiben muß, die Zielmarke zu erreichen. Bei einer beschränkten Umgebung (z. B. durch ein Rechteck mit Seitenlängen a und b), einer konstanten Geschwindigkeit v des Fahrzeuges und einer Zykluszeit Δt kann ein maximales Limit angegeben werden (z. B. durch $f_{Limit} = \left\lceil \frac{\sqrt{a^2+b^2}}{v \cdot \Delta t} \right\rceil$ im Falle einer rechteckigen Umgebungsbegrenzung).

45° war. Damit konnte der *LABMATE* in der Simulation beim seitlichen Passieren einer Zimmertür anhalten und in das Zimmer hineinfahren. Diese Neuorientierung ist ähnlich wie die nach einem Notfall durchzuführen. Dabei ergibt sich der Rotationswinkel ausschließlich aus dem eingeschlossenen Winkel der beiden benachbarten Kanten.
Die Abbildungen 3.37 und 3.38 zeigen die Wegplanung in einer zuvor erkundeten Umgebung. In Abbildung 3.37 wird dem Fahrzeug ein Zielpunkt in der unteren linken Ecke der Umgebung vorgegeben. Die dick dargestellte Linie zeigt das Ergebnis der Wegplanung. Entlang der markierten Kanten gelangt das Fahrzeug zum Zielpunkt. Es fährt dabei von Wegmarke zu Wegmarke und nutzt sein reaktives Hindernisvermeidungsverhalten.

Kapitel 4

Neuronale und Fuzzy-basierte Regelungssysteme

In Kapitel 3 wurde ein vollständiges Navigationssystem für einen autonomen mobilen Roboter auf der Basis statischer Verhaltensregeln angegeben. In den folgenden Kapiteln dieser Arbeit wird untersucht, inwieweit es möglich ist, den bisherigen durch [Mat92] motivierten verhaltensbasierten Steuerungsansatz aufrecht zu erhalten und in einen Ansatz zu überführen, dessen Regelbasis aus *weichen* Fuzzy-Regeln besteht, die sich zusätzlich in ein neuronales Netzwerk einbetten lassen. Dieses System soll aus eigener Erfahrung in der Lage sein, seine Verhaltensregeln selbständig zu optimieren und sich an ändernde Rahmenbedingungen anzupassen.
Dazu wird in diesem Kapitel eine Klasse neuronaler Netzwerke vorgestellt, die hier die Basis der zu entwerfenden Regler stellen soll. Diese werden im zweiten Abschnitt dieses Kapitels durch geringfügige Erweiterungen zu adaptiven Fuzzy-Reglern verallgemeinert.
Es wird gezeigt, welche enge Beziehungen zwischen Fuzzy-Regeln und den neuronalen Einheiten der hier verwendeten neuronalen Netzwerke besteht. Um diesen Zusammenhang auch in einen formalen Rahmen zu stellen, wird von der sonst üblichen Notation neuronaler Netzwerke abgewichen. Statt die Gewichtsvektoren eines neuronalen Netzes in Form von Matritzen anzugeben, wird die Zugehörigkeit der Gewichte zu einzelnen neuronalen Einheiten stärker betont.

4.1 Ein neuronaler Regler

4.1.1 Radiale-Basis-Funktions-Netzwerke

Als Grundlage des hier verwendeten neuronalen Netzwerkmodells dienen die sogenannten *Radialen-Basis-Funktions-Netzwerke* oder kurz RBF-Netzwerke, deren Vorteil gegenüber den häufig verwendeten mehrschichtigen Perceptron-Netzwerken (multi layer perceptron (MLP)) darin besteht, daß man den einzelnen neuronalen Einheiten des Netzwerkes Bedeutungen beimessen kann. Für die hier verfolgte Anwendung ist dieser Sachverhalt von besonderem Interesse, da hier einzelne neuronale Einheiten auch als Fuzzy-Regeln interpretiert werden sollen, und sich somit auf einfache Weise Vorwissen in solche RBF-Netzwerke einbinden läßt.

Ein *Radiales Basis-Funktions-Netzwerk (RBF)* ist ein zweischichtiges Netzwerk, dessen Ausgaben aus Linearkombinationen radialsymmetrischer Basisfunktionen gebildet werden (vgl. [HH93]).

Formal wird es im folgenden als Quintupel $\mathcal{R} = (I, O, U, \alpha, \delta_{\mathcal{R}})$ eingeführt. Dabei seien $n, m \in \mathbb{N}$ und $I \subseteq \mathbb{R}^n$ ein n-dimensionaler reeller *Eingaberaum* und $O \subseteq \mathbb{R}^m$ ein m-dimensionaler reeller *Ausgaberaum*.

Eine endliche Menge $U \subseteq I \times O$ enthält Paare von Ein- und Ausgabegewichten von Verarbeitungseinheiten, die auch kurz *Neuronen* oder *neuronale Einheiten* genannt werden. U wird *Neuronenmenge* genannt. Für $(c, w) \in U$ wird c als *Zentrum* oder *Eingabegewicht* und w als *Stützvektor* oder *Ausgabegewicht* des Neurons (c, w) bezeichnet. Für $u \in U$ werden Zentren bzw. Ausgabegewichte mit c_u bzw. w_u notiert.

$\delta_{\mathcal{R}} : I \to O$ bezeichnet im folgenden die *Evaluierungsfunktion* oder *Auswertung* des RBF-Netzwerkes $\mathcal{R}$. Sie ordnet jedem Eingabestimulus $x \in I$ eine Ausgabe $y = \delta_{\mathcal{R}}(x) \in O$ des Netzes $\mathcal{R}$ zu. Dazu gibt $\alpha : U \to [0,1]^I, u \mapsto \alpha_u$ eine Abbildung an, die jedem Neuron eine radialsymmetrische *Aktivierungsfunktion* $\alpha_u : I \to [0,1]$ zuordnet.

Zu einem *Eingabestimulus* $x \in I$ berechnet sich die Antwort von $\mathcal{R}$ durch

$$\begin{aligned} &\delta_{\mathcal{R}} : I \to O, x \mapsto y \quad \text{mit} \\ &y_j = \sum_{u \in U} (w_u)_j \cdot \alpha_u(x), \quad j \in \{1, \cdots, m\}. \end{aligned} \tag{4.1}$$

$\alpha_u(x)$ wird als *Aktivität* oder *Aktivierung* einer neuronalen Einheit $u \in U$ zum Eingabestimulus $x \in I$ bezeichnet.

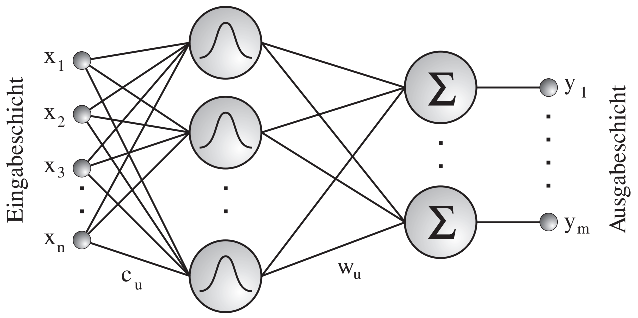


Abbildung 4.1: Modell eines Radialen-Basis-Funktions-Netzwerkes

Solche RBF-Netzwerke können als gute Funktionsapproximatoren eingesetzt werden. Die Approximation wird durch die Lage der Zentren der neuronalen Einheiten und durch die Wahl der Stützvektoren bestimmt. Anders als bei MLP-Netzwerken repräsentiert jedes Neuron einen Ein-Ausgabe-Prototypen. Als mögliche Aktivierungsfunktionen für ein Neuron $u \in U$ bieten sich beispielsweise folgende an:

$$\gamma_\sigma : I \to [0,1], \quad x \mapsto \exp\left(-\frac{\|x - c_u\|^2}{\sigma^2}\right) \quad \text{(Gaußsche Basisfkt.)}, \tag{4.2}$$

$$\rho_\sigma : I \to [0,1], \quad x \mapsto \frac{1}{1 + \sigma \cdot \|x - c_u\|^2} \quad \text{(rationale Basisfkt.)} \tag{4.3}$$

Der Wert $\sigma \in \mathbb{R}_{>0}$ gibt hier an, wie stark die Aktivierung eines einzelnen Neurons abnimmt, wenn sich der Eingabestimulus vom Zentrum eines Neurons entfernt. Daher bezeichnet man diesen Wert auch als *Aktivierungsradius* eines Neurons.

4.1.2 Erweiterungen von RBF-Netzwerken

Normalisierte RBF-Netzwerke

Während in RBF-Netzwerken eine Ausgabe nur durch Linearkombination der Aktivitäten der Neuronen bestimmt ist, kann die Approximationseigenschaft des Netzwerkes durch *Normalisierung* in vielen Anwendungen noch verbessert werden. Die Ausgabeberechnung des Netzwerkes ändert sich dann dahingehend, daß die Antwort des Netzwerkes zusätzlich durch die Summe der Aktivitäten geteilt wird. Zu einem RBF-Netzwerk $\mathcal{R} = (I, O, U, \alpha, \delta_\mathcal{R})$ erhält man durch

$$\tilde{\delta}_\mathcal{R} : I \to O, \quad x \mapsto \frac{1}{\sum\limits_{u \in U} \alpha_u(x)} \cdot \delta_\mathcal{R}(x), \tag{4.4}$$

ein *normalisiertes RBF-Netzwerk* $\tilde{\mathcal{R}} = (I, O, U, \alpha, \tilde{\delta}_\mathcal{R})$.
Für jedes Neuron $u \in U$ ist die Funktion $\alpha_u : I \to [0,1]$ eine in $\mathcal{R}$ verwendete Aktivierungsfunktion, deren *Träger* $\operatorname{supp}(\alpha_u) = \overline{\{x \in I | \alpha_u(x) \neq 0\}}$ mit dem gesamten Definitionsbereich $\operatorname{Def}(\alpha_u) = I$ übereinstimmt. Damit ist die normalisierte Evaluierungsfunktion überall definiert.
Die Abbildung 4.2 gibt ein Beispiel zur Normalisierung an. Die untersten drei Funktionen sind gewichtete Gaußsche Basisfunktionen, deren Maxima bei $(1,2)$, $(2,4)$ und $(4,1)$ liegen. Die Summe dieser drei Funktionen ergibt die Evaluierungsfunktion des zugehörigen RBF-Netzwerkes, die durch die Funktion mit dem größten Maximum dargestellt wird. Die normalisierte Evaluierungsfunktion wird dagegen durch die verbleibende Funktion angegeben.

Normierung des Eingaberaumes

Die Bereiche im Eingaberaum um das Zentrum eines Neurons, in denen eine Eingabe eine nennenswerte Aktivierung eines Neurons hervorruft, wird als *rezeptives Feld* eines Neurons bezeichnet [HH93]. Aufgrund der Verwendung der Euklidischen Norm zur Berechnung der Neuronenaktivierung erhält man radialsymmetrische rezeptive Felder.
In vielen Anwendungen ist der Eingaberaum beschränkt. Als Beispiel hierfür betrachte man einen zweidimensionalen Eingaberaum, der in $R = [-a, a] \times [-b, b]$ enthalten ist, wobei $a, b \in \mathbb{R}_{>0}$ mit $a \gg b$. Verwendet man als Basisfunktion beispielsweise γ_σ aus (4.2) mit $\sigma \approx 2b$, können mehrere Neuronen in R so verteilt werden, daß sich ihre rezeptiven Felder in Richtung der ersten Koordinatenachse kaum überlappen. Das ist aufgrund der Form des Eingaberaumes R für die zweite Koordinatenachse nicht möglich. Dadurch wird der Wunsch nach rezeptiven Feldern motiviert, die unterschiedliche Ausdehnungen des

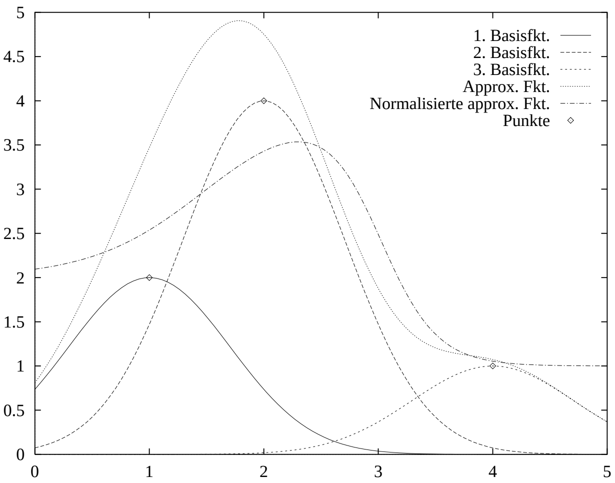


Abbildung 4.2: Approximation von drei Stützwerten: Basisfunktionen, gewichtete Linearkombination der Basisfunktionen und normalisierte Ausgabefunktion

Eingaberaumes entlang verschiedener Koordinatenachsen berücksichtigen. Einen möglichen Lösungsansatz erhält man durch Normierung der Eingaben auf einen Hyperwürfel mit Kantenlänge 1. Die Form der rezeptiven Felder entspricht damit keinen Kugeln mehr, sondern Ellipsoiden.
Ist $I \subseteq I\!R^n$ mit $n \in I\!N$ ein beschränkter reeller Eingaberaum und sind $l_1, l_2, \cdots, l_n$, $r_1, r_2, \cdots, r_n \in I\!R$ mit $l_i < r_i$ für alle $i \in \{1, 2, \cdots, n\}$, so daß der Eingaberaum I in den n-dimensionalen Hyperquader H eingebettet ist:

$$I \subseteq H := \bigtimes_{i=1}^{n} [l_i, r_i]. \tag{4.5}$$

Dann wird die Normierung von Eingaben aus I auf den Hyperwürfel $[0,1]^n$ durch die Abbildung

$$\eta_H : I \to [0,1]^n, \quad x \mapsto \begin{pmatrix} \frac{x_1 - l_1}{r_1 - l_1} \\ \vdots \\ \frac{x_n - l_n}{r_n - l_n} \end{pmatrix} \tag{4.6}$$

vorgenommen. Ist $\mathcal{R} = (I, O, U, \alpha, \delta_{\mathcal{R}})$ ein RBF-Netzwerk, so erhält man aus

$$\mathcal{R}_N = (I, O, \{(\eta_H(c), w) | (c, w) \in U\}, \alpha, \delta_{\mathcal{R}} \circ \eta_H) \tag{4.7}$$

ein *RBF-Netzwerk mit Eingabenormierung.* Durch die Normierung der Neuronenzentren und der Verknüpfung der Evaluierungsfunktion mit der Normierung bleibt die Approximationseigenschaft bis auf die Anpassung der rezeptiven Felder erhalten. Die Approximation ist dadurch aber unabhängig von der Ausdehnung des Eingaberaumes entlang verschiedener Koordinatenachsen.

Die meisten Lernalgorithmen für RBF-Netzwerke gehen für das überwachte Lernen von einer festen Trainingsmenge von Ein- und Ausgabebeispielen aus, mit denen meistens in

zwei Schritten erst die Zentren und dann die Ausgabegewichte justiert werden [HH93]. Bekannte Algorithmen sind der *K-Means-Cluster-Algorithmus* zur Adaption der Zentren und der *LMS-Algorithmus* (least mean square) zur Adaption der Ausgabegewichte.
Da die im folgenden zu entwerfenden Regler oftmals mit Trainingsdaten trainiert werden müssen, die durch ein dynamisches System während der Betriebsphase entstehen und nicht durch eine statische Trainingsmenge a priori gegeben sind, eignet sich diese Art von Lernverfahren nicht für die hier verfolgten Ziele.
Stattdessen werden die RBF-Netzwerke im nächsten Abschnitt mit einer zusätzlichen Verbindungsstruktur zwischen den einzelnen Neuronen ausgestattet, mit der zum Adaptieren der Zentren ein Lernverfahren verwendet wird, welches dem der Kohonen-Netzwerke ähnlich ist [Koh87].

4.1.3 Dynamische Zellstrukturen

Dynamische Zellstrukturen (*dynamic cell structures (DCS)*) [BS95] basieren auf den im vorherigen Abschnitt eingeführten RBF-Netzwerken und gehören in die Klasse der in [MS93] definierten topologierepräsentierenden Netzwerke (*topology representing networks (TRN)*).

Ähnlich wie in Kohonen-Netzwerken sind die neuronalen Einheiten des Netzwerkes untereinander verbunden. Durch diese Verbindungsstruktur werden Nachbarschaftsrelationen im Eingaberaum durch Nachbarschaftsrelationen der neuronalen Einheiten modelliert. Die Topologie des Eingaberaumes wird somit vollständig durch DCS-Netzwerke repräsentiert.
Formal werden DCS-Netzwerke jetzt als Erweiterungen von RBF-Netzwerken eingeführt. *Dynamische Zellstrukturen* oder *DCS-Netzwerke* sind 8-Tupel $\mathcal{D} = (I, O, U, L, \rho, \alpha, r, \delta_{\mathcal{D}})$, wobei $I \subseteq \mathbb{R}^n$, $O \subseteq \mathbb{R}^m$ Ein- und Ausgaberäume sind und $U \subseteq I \times O$ eine endliche Neuronenmenge wie bei den RBF-Netzwerken ist. $L \subseteq U^2 \setminus id_U$ ist zusätzlich eine Menge lateraler Neuronenverbindungen, so daß (U, L) ein gerichteter Graph ist. Ferner werden durch

$$\begin{aligned} &\rho : U^2 \to [0,1], \quad (u,v) \mapsto \begin{cases} c_{uv}, & (u,v) \in L \\ 0 & , sonst \end{cases} \\ &\alpha : U \to [0,1]^I, \quad u \mapsto \alpha_u, \\ &r : U \to \mathbb{R}_{\geq 0}, \quad u \mapsto r_u, \end{aligned}$$

Abbildungen bezeichnet. ρ ist eine Kantenbewertung für den Graphen (U, L), die je zwei Neuronen eine *Verbindungsstärke* zuordnet. Ein Paar nicht verbundener Neuronen erhält die Verbindungsstärke 0, während verbundene Neuronen u, v eine Stärke $0 < c_{uv} \leq 1$ erhalten.
Die Abbildung α ordnet wie bei RBF-Netzwerken jedem Neuron eine individuelle Aktivierungsfunktion zu.
Die Abbildung r wird *Resource-Abbildung* genannt und ordnet jedem Neuron einen reellen Wert zu, der als *Resource* dieses Neurons bezeichnet und später dazu verwendet wird, lokale Fehler des Netzwerkes an die neuronalen Einheiten zu heften.
Die Funktion $\delta_{\mathcal{D}}$ ist schließlich die *Evaluierungsfunktion* des DCS-Netzes $\mathcal{D}$ und ordnet jedem Eingabestimulus eine Antwort des Netzwerkes zu.

Als Repräsentant dieser Klasse von künstlichen neuronalen Netzwerken wird in [BS95] ein dynamisch wachsendes DCS-Netzwerk behandelt, welches auf Lernstrategien zurückgreift, wie sie in Fritzke's GCS-Netzwerken verwendet werden [Fri95]. Die in [BS95] angegebene Instanz eines DCS-Netzwerkes wird als GDCS-Netzwerk bezeichnet und soll im folgenden näher betrachtet und an Probleme des On-Line-Lernens angepaßt werden. Dazu wird ein DCS-Netzwerk $\mathcal{D} = (I, O, U, L, \rho, \alpha, r, \delta_{\mathcal{D}})$ für diesen Abschnitt als gegeben vorausgesetzt.

Lernen der Neuronenverbindungsstruktur

Als Teilklasse der in [MS93] eingeführten Klasse der topologierepräsentierenden Netzwerke kann die Verbindungsstruktur eines DCS-Netzwerkes mit Hilfe des in [MS93] angegebenen Hebb'schen Wettbewerbslernen aufgebaut werden. Diese Lernregel verbindet sukzessive zwei Neuronen miteinander, die durch eine Eingabe am stärksten aktiviert werden. Zwei solche Neuronen $bmu, second \in U$ werden durch[1]

$$\forall u \in U: \quad \|c_{bmu} - x\| \leq \|c_u - x\|, \tag{4.8}$$

$$\forall u \in U \setminus \{bmu\}: \quad \|second - x\| \leq \|c_u - x\|. \tag{4.9}$$

erklärt. Das Neuron $bmu \in U$ ist das zur Eingabe x *am besten passende Neuron* (best matching unit (bmu)) und $second \in U$ das *am zweitbesten passende Neuron*.
Die laterale Verbindungsstruktur wird gemäß der folgenden Regel adaptiert:

$$\Delta\rho(u,v) = \begin{cases} \alpha_{bmu}(x) \cdot \alpha_{second}(x), & \{u,v\} = \{bmu, second\} \\ 0 & , sonst \end{cases}, \tag{4.10}$$

wobei sich die neue Verbindungsstärke durch $\rho := \rho + \Delta\rho$ ergibt. Als laterale Verbindungen ergibt sich daraus die Menge $L = \{(u,v) | \rho(u,v) > 0\}$. Martinetz konnte in [MS93] beweisen, daß durch eine sukzessive Anwendung dieser Regel die Topologie des Eingaberaumes perfekt in die Topologie des Netzwerkes abgebildet wird, d. h. Neuronen, deren Zentren benachbarte Voronoi-Zellen besitzen, sind auch im Graphen (U, L) benachbart und umgekehrt. Auf diesen Beweis wird im Anhang B noch näher eingegangen. Dort wird eine allgemeinere Fassung des Satzes von Martinetz bewiesen.
In [BS95] wird eine Variante dieser Hebb'schen Wettbewerbsregel vorgeschlagen, die auch den Abbau von lateralen Neuronenverbindungen vorsieht. Dieses ist insbesondere dann sinnvoll, wenn nichtstationäre Eingabedaten vorliegen und sich dadurch die Nachbarschaftsrelationen im Eingaberaum im Laufe der Zeit ändern. Das Vergessen einer Verbindung wird in [BS95] durch Einführung einer Konstante $\alpha_c \in]0,1[$ und einem Schwellwert $\theta \in]0,1[$ nach folgender Adaptionsregel durchgeführt:

$$\rho(u,v) := \begin{cases} 1 & , \{u,v\} = \{bmu, second\} \\ 0 & , bmu \in \{u,v\} \wedge second \notin \{u,v\} \wedge \alpha_c \cdot \rho(u,v) < \theta \\ \alpha_c \cdot \rho(u,v), & bmu \in \{u,v\} \wedge second \notin \{u,v\} \wedge \alpha_c \cdot \rho(u,v) \geq \theta \\ \rho(u,v) & , sonst \end{cases} \tag{4.11}$$

[1] Die durch (4.8) und (4.9) definierten Neuronen müssen natürlich nicht eindeutig bestimmt sein. Sollten Mehrdeutigkeiten auftreten, so kann eine beliebige Auswahl getroffen werden.

Eine Verbindung altert also um den Faktor α_c, wenn sie nicht die beiden am besten passenden Neuronen miteinander verbindet, sondern nur auf einer Seite mit dem am besten passenden Neuron *bmu* inzidiert.

Unüberwachtes Lernen im DCS-Netzwerk

Für die bisher besprochene laterale Verbindungsstruktur der Neuronen wurde bisher nur das Erlernen der Nachbarschaftsstruktur einer vorgegebenen Menge von Neuronen behandelt. Als nächstes wird ein Lernverfahren aus [BS95] angegeben, welches die Zentren der Neuronen adaptiert, so daß der Eingaberaum möglichst gut überdeckt wird. Die Zentrenadaption wird wie bei Kohonens selbstorganisierenden Karten (*self-organizing maps (SOM)*) [Koh87] vorgenommen, wo die Zentren der am stärksten erregten neuronalen Einheit und ihrer Nachbarn von einer Eingabe angezogen werden. Darüberhinaus wächst während des Lernverfahrens die Anzahl der im Netzwerk befindlichen Neuronen wie bei Fritzke's GCS-Netzwerken [Fri95]. Im Gegensatz zu GCS und SOM ist es nicht nötig, eine feste Dimension des Eingaberaums anzunehmen. Stattdessen lernen DCS-Netzwerke nicht nur die Topologie und die Wahrscheinlichkeitsverteilung der Eingabedaten, sondern auch die Dimension des Eingaberaums.
Um das Adaptionsverfahren der Zentren angeben zu können, wird zunächst der Begriff der *Nachbarschaft* eines Neurons benötigt. Diese wird zu einem Neuron $u \in U$ durch die Menge

$$Nh(u) := \{v \in U | (u, v) \in L\} \tag{4.12}$$

gegeben. Wie bei Kohonens selbstorganisierenden Karten werden zu einer Eingabe $x \in I$ die Zentren des am besten passenden Neurons und seiner Nachbarn in Richtung der Eingabe verschoben.

$$\Delta c_{bmu} := \varepsilon_{bmu} \cdot (x - c_{bmu}), \tag{4.13}$$

$$\forall u \in Nh(bmu): \quad \Delta c_u := \varepsilon_{Nh} \cdot (x - c_u). \tag{4.14}$$

Die Konstanten $\varepsilon_{bmu}, \varepsilon_{Nh} \in \mathbb{R}_{>0}$ geben die Stärke der Verschiebung der Zentren in Richtung der Eingabe an.
Für das Einfügen neuer Neuronen wird wie bei [Fri95] jedes Neuron $u \in U$ mit einem Resourcewert r_u versehen. Diese Resourcewerte akkumulieren Fehler, die für ein Neuron entstehen, wenn es den Wettbewerb um eine Eingabe gewinnt und das am besten passende Neuron dazu darstellt. Als mögliche lokale Fehlergröße, die in dem Resourcewert akkumuliert wird, kann beispielsweise der quadratische Abstand zwischen der Eingabe und dem Zentrum von *bmu* verwendet werden. Die Verteilung der Resourcen über dem Eingaberaum gibt an, in welchen Gebieten die größten Fehler gemacht wurden. Um diese Fehler möglichst gleich über den Eingaberaum zu verteilen, werden in Regionen großer Resourcen neue Neuronen eingefügt.
Zu einer Eingabe $x \in I$ wird die Resource des am besten passenden Neurons gemäß

$$\Delta r_{bmu} = \|x - c_{bmu}\|^2 \tag{4.15}$$

adaptiert. Nachdem dem Netzwerk über $\lambda_{ins} \in I\!N$ Schritte Eingaben präsentiert und dabei die Resourcen angepaßt wurden, werden zwei Neuronen[2] $\ell, s \in U$ mit größten lokalen Fehlern ermittelt.

$$\forall u \in U: \quad r_\ell \geq r_u, \tag{4.16}$$
$$\forall u \in Nh(\ell): \quad r_s \geq r_u. \tag{4.17}$$

Das Zentrum eines neu einzufügenden Neurons richtet sich nach den Resourcen von ℓ und s. Dazu berechnet man

$$\bar{\gamma} = \frac{r_s}{r_\ell + r_s}, \tag{4.18}$$
$$\Delta r_\ell = \frac{1}{2}(1 - \bar{\gamma}) r_\ell), \quad \Delta r_s = \frac{1}{2}\bar{\gamma} r_s. \tag{4.19}$$

Auf diese Weise erhält man eine Schätzung der Resourcenverteilung für den Fall, daß das neue Neuron von Anfang an im Netzwerk enthalten gewesen wäre.

$$c_{new} = c_\ell + \bar{\gamma}(c_s - c_\ell), \quad r_{new} = \Delta r_\ell + \Delta r_s, \tag{4.20}$$
$$r_\ell = r_\ell - \Delta r_\ell, \quad r_s = r_s - \Delta r_s. \tag{4.21}$$

Dann gibt c_{new} das Zentrum und r_{new} die Resource des neu einzufügenden Neurons an. Die Verbindungen werden so geändert, daß das neue Neuron mit den beiden alten Neuronen verbunden wird, wobei deren Verbindungen gelöscht werden.

$$\rho(u_\ell, u_s) = \rho(u_s, u_\ell) = 0, \tag{4.22}$$
$$\rho(u_{new}, u_\ell) = \rho(u_\ell, u_{new}) = 1, \tag{4.23}$$
$$\rho(u_{new}, u_s) = \rho(u_s, u_{new}) = 1, \tag{4.24}$$
$$L := \{(u, v) | \rho(u, v) > 0\}. \tag{4.25}$$

Damit die Resourcen nicht zu groß werden, altern die Resourcen nach ebenfalls λ_{ins} Schritten. Für alle Neuronen $u \in U$ werden dazu die Resourcen durch $r_u := \beta \cdot r_u$ mit $\beta \in]0, 1[$ angepaßt.

Überwachtes Lernen im DCS-Netzwerk

Während im unüberwachten Lernverfahren die Anpassung der Neuronenzentren an die Eingabedaten und das Einfügen neuer Neuronen im Vordergrund steht, um eine gute Repräsentation des Eingaberaumes zu gewährleisten (adaptive Vektorquantisierung), werden nun die Ausgabegewichte der Neuronen betrachtet.
Das in [BS95] vorgestellte GDCS-Netzwerk berechnet zu einer gegebenen Eingabe $x \in I$ die Ausgabe gemäß

$$\delta_{\mathcal{D}}(x) = \sum_{u \in \{bmu\} \cup Nh(bmu)} \alpha_u(x) \cdot w_u. \tag{4.26}$$

[2] ℓ steht für *largest resource* und s steht für *second largest resource*.

Ist $y \in O$ die zu x gehörige gewünschte Ausgabe, so sollen die Ausgabegewichte so geändert werden, daß der quadratische Fehler $E_{\mathcal{D}}(x,y) = \|y - \delta_{\mathcal{D}}(x)\|^2$ minimiert wird. Dieses wird durch einfachen Gradientenabstieg erreicht und führt zur bekannten Delta-Regel:

$$\forall u \in \{bmu\} \cup Nh(bmu): \quad \Delta w_u = -\frac{\eta}{2} \nabla_{w_u} E_{\mathcal{D}}(x,y) = \eta \alpha_u(x)(y - \delta_{\mathcal{D}}(x)). \tag{4.27}$$

Dabei werden nur die Gewichte des am besten passenden Neurons und seiner Nachbarn adaptiert. Die Schrittweite des Gradientenverfahrens wird durch die Lernkonstante $\eta > 0$ festgelegt.
Für die Resourcen wird im überwachten Lernverfahren vorzugsweise statt (4.15) der quadratische Ausgabefehler des Netzwerkes verwendet. Das führt zu folgender Adaptionsregel:

$$\Delta r_{bmu} = \|y - \delta_{\mathcal{D}}(x)\|^2. \tag{4.28}$$

4.1.4 Erweiterungen für DCS-Netzwerke

Die bisher beschriebenen Strukturen und Lernverfahren der DCS-Netzwerke sind in [BS95] in zahlreichen Simulationen getestet worden.
Im folgenden sollen einige Ergänzungen zu den DCS-Netzwerken angegeben werden, die sich insbesondere für Probleme des On-Line-Lernens als hilfreich herausgestellt haben [ABS95].

Normalisierte DCS-Netzwerke

Ebenso wie für die Standard-RBF-Netzwerke Eingabenormierung und Normalisierung eingeführt wurde, kann dieses auch für DCS-Netzwerke vorgenommen werden.
Die Normierung der Eingabedaten ist vollständig analog zu den RBF-Netzwerken vorzunehmen und ändert außer der Form der rezeptiven Felder der Neuronen keine wesentlichen Eigenschaften der DCS-Netzwerke.
Die Normalisierung des Netzwerkes führt wie beim RBF-Netzwerk in vielen Anwendungen zu einer Verbesserung des Approximationsverhalten. Da sich dabei die Evaluierungsfunktion des Netzwerkes ändert, hat dieses auch Konsequenzen für die Adaption der Ausgabegewichte. Zu einer Eingabe $x \in I$ berechnet sich die Ausgabe gemäß

$$\tilde{\delta}_{\mathcal{D}}(x) = \frac{1}{\sum\limits_{u \in \{bmu\} \cup Nh(bmu)} \alpha_u(x)} \cdot \sum_{u \in \{bmu\} \cup Nh(bmu)} \alpha_u(x) \cdot w_u. \tag{4.29}$$

Da sich die Ausgabe nur um einen von den Ausgabegewichten unabhängigen Faktor ändert, verändert sich die Adaptionsregel der Ausgabegewichte entsprechend zu:

$$\forall u \in \{bmu\} \cup Nh(bmu): \quad \Delta w_u = \frac{\eta \alpha_u(x)(y - \delta_{\mathcal{D}}(x))}{\sum\limits_{u \in \{bmu\} \cup Nh(bmu)} \alpha_u(x)}. \tag{4.30}$$

Alternative Strategie zum Neuroneneinfügen

Im klassischen DCS-Ansatz werden neue Neuronen nach λ_{ins} Schritten an den Stellen eingefügt, wo die Resourcen am größten sind *(fehlergetriebenes Neuroneneinfügen)*. Dieses ist ein legitimes Vorgehen, wenn sich die Trainingsphase an einer festen Trainingsmenge T orientiert, die dem Netzwerk immer wieder von neuem präsentiert wird (z. B. mit $\lambda_{ins} = |T|$). Das hier verfolgte Ziel ist aber die Adaption eines neuronalen Reglers, der ein dynamisches System steuern soll. Die Trainingsdaten entstehen hier aus der Dynamik des Systems und stammen nicht aus einer festen endlichen Trainingsmenge. Befindet sich das System längere Zeit in ähnlichen Zuständen, so fallen auch die Trainingsdaten in lokal begrenzte Gebiete des Eingaberaumes. Dieses kann zu Schwierigkeiten beim *Entfalten* des Netzwerkes führen. Die Idee zur Behebung dieses Problems besteht in einer zusätzlichen Neuroneneinfügestrategie. Dazu wird ein Schwellwert $\delta \in \mathbb{R}_{>0}$ eingeführt, der den maximalen Abstand eines Eingabestimulus zum Zentrum des am besten passenden Neurons angibt. Ist $\|x - c_{bmu}\| > \delta$ für eine Eingabe $x \in I$ und enthält das Netzwerk weniger als eine maximal zulässige Neuronenanzahl $N_{max} \in \mathbb{N}$, so wird ein neues Neuron an der Stelle des aktuellen Trainingspaares $(x, y) \in I \times O$ eingefügt. Zusätzlich wird eine Verbindung zu dem letzten am besten passenden Neuron bmu hergestellt. Diese Einfügestrategie wird hier als *distanzgetriebenes Neuroneneinfügen* bezeichnet. Eine Kombination aus beiden Einfügestrategien ist genauso möglich und wird im folgenden *gemischte Neuroneneinfügestrategie* genannt.

Unsymmetrische laterale Verbindungsstruktur

Wenn die Verbindungsstruktur der Neuronen einem Alterungsprozeß unterworfen ist (vgl. (4.11)), ergibt sich ein weiteres Problem der DCS-Netzwerke. Durch das Vergessen von lateralen Neuronenverbindungen ist es möglich, daß ein nicht zusammenhängender Neuronenverbindungsgraph (U, L) entsteht. Zusammenhangskomponenten, die in Regionen liegen, in denen Eingabestimuli eine geringe Aufenthaltswahrscheinlichkeit besitzen, werden hier als *tote Neuronen* bezeichnet. Durch die fehlende Verbindung zu anderen Zusammenhangskomponenten können diese toten Neuronen nicht mehr in Bereiche des Eingaberaumes mit höherer Aufenthaltswahrscheinlichkeit gezogen werden und sind damit für die Funktionsapproximation verloren. Um dieses Problem zu beheben, wird die bisherige symmetrische Behandlung der lateralen Neuronenverbindungen fallengelassen. Zu dem am besten passenden Neuron bmu altern nur noch die Verbindungen, die von bmu ausgehen. Die Verbindung zwischen zwei Neuronen wird nur dann gelöscht, wenn die Verbindungsstärke in beide Richtungen unterhalb der Schwelle θ liegen. Als neue Adaptionsregel der Verbindungsstärken erhält man somit:

$$\rho(u,v) := \begin{cases} 1 & , \{u,v\} = \{bmu, second\} \\ 0 & , \alpha_c \cdot \rho(u,v) < \theta \wedge \rho(v,u) < \theta \wedge u = bmu \wedge v \neq second \\ 0 & , \alpha_c \cdot \rho(v,u) < \theta \wedge \rho(u,v) < \theta \wedge v = bmu \wedge u \neq second \\ \alpha_c \cdot \rho(u,v), & \alpha_c \cdot \rho(u,v) \geq \theta \vee \rho(v,u) \geq \theta \wedge u = bmu \wedge v \neq second \\ \rho(u,v) & , sonst \end{cases} \tag{4.31}$$

Fehlermodulierte Kohonen-Regel zur Zentrenadaption

Trainingsdaten, die durch ein dynamisches System erzeugt werden, sind oftmals nicht gleichmäßig im Eingaberaum verteilt, sondern beschreiben bedingt durch die Dynamik des Systems Trajektorien im Eingaberaum. Bei Verwendung der Kohonen-Regel zur Adaption der Neuronenzentren besteht die Gefahr, daß mehrere Neuronen diese Trajektorien im Eingaberaum verfolgen. Dadurch ist das Netzwerk nicht in der Lage sich zu stabilisieren und kann die zu erlernende Funktion nicht ausreichend beschreiben.
Abhilfe für dieses Problem schafft die Berücksichtigung der Fehlerverteilung bei der Zentrenadaption. Dazu wird für die Resourcen eines Neurons ein gleitender Mittelwert

$$r_{bmu} := \gamma_{res} \cdot r_{bmu} + (1 - \gamma_{res}) \cdot err, \quad \gamma_{res} \in [0,1] \tag{4.32}$$

bestimmt. Als Fehlersignal $err \in \mathbb{R}$ wird beispielsweise wieder der quadratische Abstand zwischen Eingabe und Zentrum oder der quadratische Abstand zwischen berechneter Ausgabe und Trainingsausgabe verwendet. Dadurch geben die an die Neuronen gehefteten Resourcen eine Verteilung der Fehler im Eingaberaum an.
Um eine Gleichverteilung der lokalen Fehlergrößen (Resourcen) im Eingaberaum gewährleisten zu können und weitgehend unabhängig von der Wahrscheinlichkeitsverteilung der Eingabedaten die Neuronenzentren so zu verschieben, daß einerseits die loklen Fehler reduziert werden und andererseits es nicht zu unerwünschten Häufungen von Neuronenzentren im Eingaberaum kommt, werden die Adaptionsregeln für die Zentren des am besten passenden Neurons und der Nachbarneuronen an den lokalen Fehlern orientiert.
Das Zentrum des am besten passenden Neurons wird nur dann von der Eingabe angezogen, falls der lokale Fehler des Neurons bmu größer als der mittlere lokale Fehler $\bar{r}$ der Nachbarn ist.
Umgekehrt werden die Zentren der Nachbarn nur dann von der Eingabe angezogen, falls ihre Resource größer als die der am besten passenden neuronalen Einheit ist.
Um die Zentrenverschiebung mit dem Verhältnis der lokalen Fehlergrößen gewichten zu können, wird eine monoton fallende Funktion $c : \mathbb{R}_{\geq 0} \rightarrow [0,1]$ mit $c(0) = 1$ und $c(1) = 0$ eingeführt. Mit ihr läßt sich die folgende fehlermodulierte Kohonen-Regel zur Zentrenadaption angeben:

$$\bar{r} = \frac{1}{|Nh(bmu)|} \cdot \sum_{u \in Nh(bmu)} r_u, \tag{4.33}$$

$$\Delta c_{bmu} = \varepsilon_{bmu} \cdot (x - c_{bmu}) \cdot c\left(\frac{\bar{r}}{r_{bmu}}\right), \tag{4.34}$$

$$\forall u \in Nh(bmu): \quad \Delta c_u = \varepsilon_{Nh} \cdot (x - c_u) \cdot c\left(\frac{r_u}{r_{bmu}}\right) \tag{4.35}$$

4.1.5 Anwendungsbeispiel eines DCS-Netzwerkes

In [BS95] sind zahlreiche Simulationen mit DCS-Netzwerken durchgeführt worden, die im Vergleich zu anderen Netzwerktypen gut abgeschnitten haben. In den dort durchgeführten Versuchen wurde jedoch ausschließlich von festen Trainingsmengen ausgegangen, die dem Netzwerk wiederholt und zusätzlich in gemischter Weise präsentiert wurden.

Da in dieser Arbeit ein neuronaler Regler erstellt werden soll, der anhand von Daten trainiert wird, die sich aus der Dynamik eines zu steuernden Systems während der Betriebsphase ergeben, kann weder von einer festen noch von einer gemischten Trainingsmenge ausgegangen werden.
Daher wurden die Erweiterungen des letzten Abschnitts angegeben, um DCS-Netzwerke auch auf das On-Line-Lernproblem anwenden zu können.
Um die Wirksamkeit dieser Vorschläge zu zeigen, wird an dieser Stelle ein praktisches Beispiel eingeschoben (vgl. [ABS95]). Das hier betrachtete Problem ist die *Stabbalance* in seiner einfachsten Version [RMS90]. Das dynamische System besteht aus einem Stab, der an seinem Fußpunkt auf einem in eine Richtung frei beweglichen Wagen befestigt ist. Durch Anlegen einer Kraft f am Fußpunkt des Stabes soll erreicht werden, daß der Stab aufrecht balanciert wird. Als Eingabegrößen für den Regler sind sowohl der Stabwinkel ϕ als auch die Winkelgeschwindigkeit $\dot{\phi}$ bekannt. Der Stab hat die Länge 1 m und die Masse 1 kg, die als punktförmig an der Spitze des Stabes angenommen wird. Der Wagen, auf dem der Stab befestigt ist, besitzt die Masse m_{cart}. Als Schwerebeschleunigung wird 1 $\frac{\mathrm{m}}{\mathrm{s}^2}$ verwendet. Dann wird die Dynamik des Systems durch die Differentialgleichung

$$(m_{cart} + \sin^2\phi)\ddot{\phi} + \frac{1}{2}\dot{\phi}^2\sin(2\phi) - (m_{cart}+1)\sin\phi = -f\cos\phi \tag{4.36}$$

beschrieben. Mit einem vereinfachten Runge-Kutta-Verfahren und einer Zeitschrittweite von Δt wird die Dynamik des Systems simuliert.
Einem DCS-Netzwerk soll nun durch überwachtes Lernen beigebracht werden, den Stab zu balancieren. Als Lehrersignal dient dabei die Funktion $f^L(\phi,\dot{\phi}) = 5\sin\phi + \dot{\phi}$. In der Simulation wird der Stab zufällig mit einem Startwinkel $\frac{\pi}{4} \le \phi \le \frac{\pi}{4}$ aus der Ruhelage losgelassen. Die Kraft, die auf das Pendel angewendet wird, ergibt sich dann aus

$$f = \beta(t_k)\delta_{\mathcal{D}}(\phi,\dot{\phi}) + (1-\beta(t_k))f^L(\phi,\dot{\phi}), \tag{4.37}$$

wobei $\beta : [0,T] \to [0,1]$ eine monoton steigende Funktion ist, die die Verläßlichkeit des Netzwerkes in Abhängigkeit der Zeit angibt. Mit zunehmender Zeit t_k tritt an die Stelle des Lehrersignals die durch das Netzwerk berechnete Kraft und wird auf den Stab angewendet. Das DCS-Netzwerk wird über k_{max} Schritte mit Hilfe der fehlermodulierten Kohonen-Regel und der gemischten Neuroneneinfügestrategie trainiert. Nach einem Fehlversuch ($|\phi| > \frac{\pi}{4}$) oder nach k_{try} erfolgreichen Schritten wird die Simulation angehalten und der Stab von neuem gestartet.
Die Parameter der Simulation waren: $k_{max} = 3000$, $k_{try} = 100$, $m_{cart} = 1.0$ kg, $\Delta t = 0.3$ s, $\varepsilon_{bmu} = 0.1$, $\varepsilon_{Nh} = 0.001$, $\eta = 0.001$, $\gamma_{res} = 0.8$, $\lambda_{ins} = 70$, $\alpha_c = 0.88$, $N_{max} = 70$, $\theta = 0.01$, $\delta = 0.08$. Als Basisfunktionen wurden Gaußsche Aktivierungsfunktionen mit $\sigma = 0.2$ verwendet. Ferner wurde eine normalisierte Funktionsauswertung und eine Eingabenormierung auf den Hyperwürfel $[0,1]^2$ vorgenommen. Die Verläßlichkeit des DCS-Netzwerkes wurde durch $\beta(t_k) = \frac{k}{k_{max}}$ bestimmt. Für die fehlermodulierte Kohonen-Regel wurde $c(x) = 2x^3 - 3x^2 + 1,\ x \in [0,1]$ und $c(x) = 0,\ x > 1$ verwendet.
Nach 3000 Lernschritte war das Netzwerk mit 70 Neuronen in der Lage, den Stab zu balancieren.

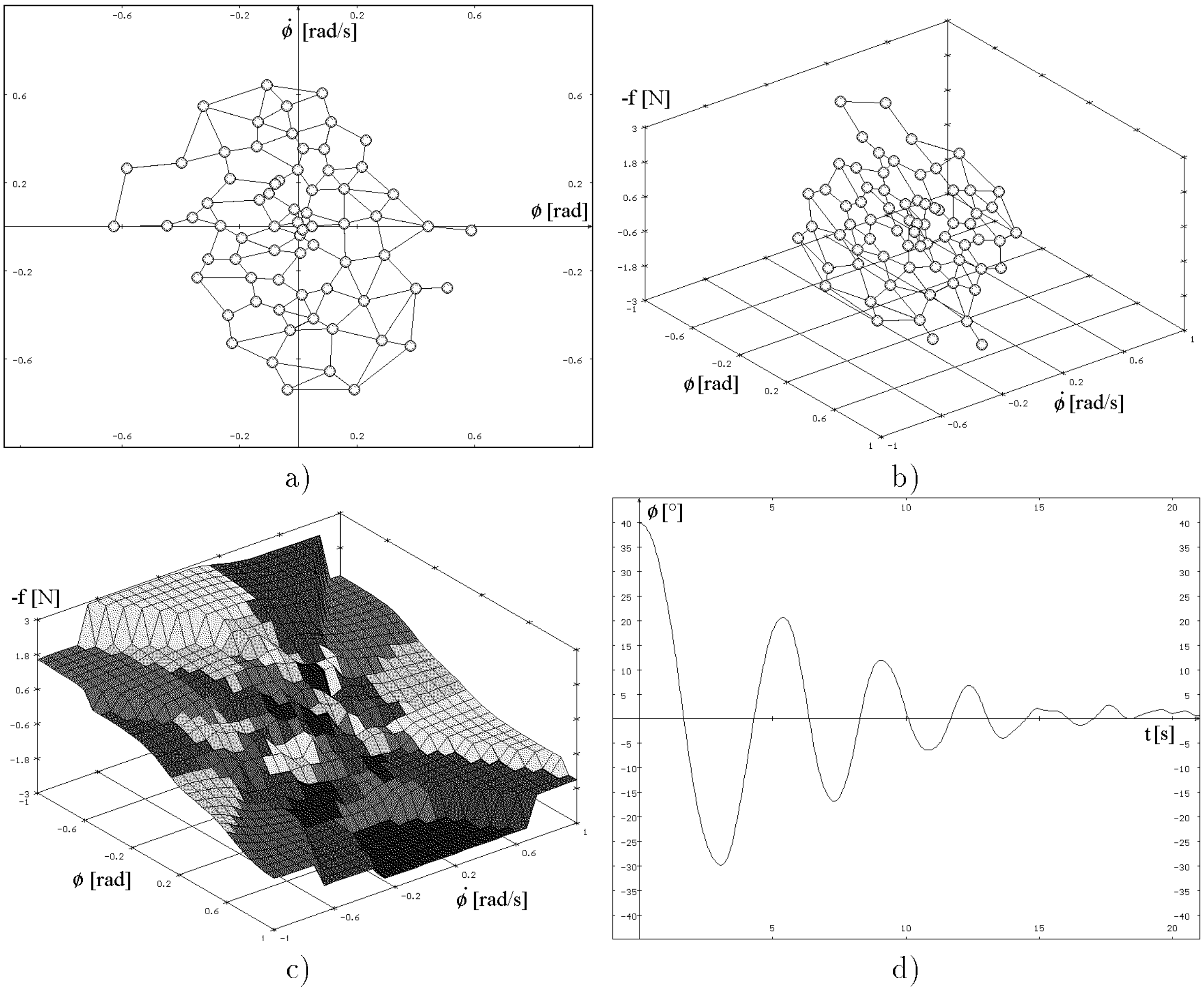


Abbildung 4.3: In a) und b) ist die Verteilung der Neuronen im Eingaberaum bzw. in $I \times O$ abgebildet. c) zeigt die approximierte Evaluierungsfunktion, die die negative Kraft zur Kontrolle des Stabes angibt. Die verschiedenen Grautöne kennzeichnen unterschiedliche Voronoi-Regionen einzelner Neuronen. In d) ist schließlich eine Schwingung $\phi(t)$ des Pendels während der Balance durch den belehrten Regler dargestellt.

4.2 Ein adaptiver Fuzzy-Regler

Nachdem bis zu dieser Stelle die Funktionsweisen der verschiedenen Varianten von RBF- und DCS-Netzwerken dargestellt wurden, wird jetzt eine Interpretation der einzelnen Neuronen als Fuzzy-Regeln gegeben, die zu einer erweiterten Klasse der DCS-Netzwerke führt.

Damit ist es möglich, die Vorteile eines Fuzzy-Reglers, wie sie beispielsweise auch in [Rei94] zur Hindernisvermeidung eines autonomen mobilen Roboters eingesetzt werden, mit den adaptiven Möglichkeiten eines neuronalen Netzes zu verbinden.
Unter einer *Fuzzy-Menge* von einem Grundbereich X, versteht man eine *Zugehörigkeitsfunktion* $\mu : X \rightarrow [0, 1]$, die jedem Element des Grundbereichs einen Zugehörigkeitsgrad zu der Menge zuordnet. Die klassische Mengenlehre ist in die Theorie der Fuzzy-Mengen

sinnvoll eingebettet, da sich eine klassische Menge als Fuzzy-Menge mit den Zugehörigkeitsgraden 0 und 1 darstellen läßt. Der Wert 1 hat dabei die Bedeutung der klassischen Mengenzugehörigkeit und der Wert 0 steht für die Nichtzugehörigkeit eines Elementes zu einer Menge. Eine klassische Menge M wird im Sinne der Fuzzy-Mengen also durch ihre charakteristische Funktion χ_M repräsentiert.
Darüberhinaus wird die Theorie der Fuzzy-Mengen auch auf die Prädikatenlogik erster Stufe übertragen. Statt den Wahrheitswerten $\mathbb{B} = \{0, 1\}$ ist die Fuzzy-Logik eine mehrwertige Logik, deren Aussagen Wahrheitswerte aus $[0, 1]$ annehmen können. Genauso wie ein Element zu einem bestimmten Grad in einer Fuzzy-Menge enthalten ist, kann eine Fuzzy-Aussage zu einem bestimmten Grad wahr sein. Damit bietet die Fuzzy-Theorie eine formale Grundlage für den Umgang mit vagen Daten, die häufig zur Beschreibung von Expertenwissen benötigt werden.
Bei der Hindernisvermeidung im vorherigen Kapitel wurde der Roboter durch Regeln gesteuert, die beispielsweise die Form

„Wenn der frontale Abstand klein ist, dann weiche scharf nach rechts aus“

hatten. Dabei treten zwei Kategorien auf, die gut durch Fuzzy-Mengen repräsentiert werden können, nämlich *klein* und *scharf*. In den meisten Fällen stammen Meßgrößen aus einem Kontinuum $[a, b]$. Während a mit Sicherheit als *klein* und b als *groß* bezeichnet werden kann, gehören Meßgrößen zwischen a und b mit bestimmten Zugehörigkeitsgraden zu beiden Kategorien. Auf der Basis solcher Interpretationen lassen sich vage Daten gut verarbeiten. In Fuzzy-Reglern findet die Theorie der Fuzzy-Mengen und Fuzzy-Logik ihre Anwendung. Hier wird Expertenwissen in Form der obigen Regel in eine Basis von Fuzzy-Regeln umgesetzt, die einen Fuzzy-Regler ein dynamisches System steuern lassen. Die bekanntesten Ansätze für Fuzzy-Regler sind die von Sugeno und Mamdani.
Für eine ausführliche Darstellung der Theorie der Fuzzy-Systeme und Fuzzy-Regler sei z. B. auf [KGK93], [Kos92] und [NKK94] verwiesen, wobei die letzten beiden außerdem auf den Zusammenhang zwischen Fuzzy-Systemen und neuronalen Netzen eingehen.

4.2.1 Interpretation von Neuronen als Fuzzy-Regeln

Eine mögliche Interpretation eines Neurons $u = (c, w) \in U$ eines RBF-Netzwerkes mit Neuronenmenge U ist eine Regel der Form:

„*Wenn* $x \approx c$ *gilt, dann ist* $y \approx w$“.

Ein spezielles Neuron wird somit als eine Fuzzy-Regel aufgefaßt. $x \in I$ ist eine Eingabe und $y \in O$ eine Ausgabe, die von einer solchen Regel vorgeschlagen wird, falls die Prämisse $x \approx c$ gilt.
Die Aktivierungsfunktion von u angewendet auf die Eingabe x liefert den *Grad der Erfüllung* $\alpha_u(x)$ der *Prämisse* $x \approx c$. Somit stellt eine Aktivierungsfunktion α_u eines Neurons u eine Fuzzy-Menge auf dem Eingaberaum I dar.
Die *Konklusion* $y \approx w$ einer solchen Regel wird mit dem Erfüllungsgrad $\alpha_u(x)$ gewichtet. Durch Summation der so gewichteten Konklusionen über alle Regeln (Neuronen) des Netzwerkes wird eine approximierte Ausgabe erhalten.

In praktischen Anwendungen werden die Prämissen meistens als konjunktive Verknüpfungen von vagen Aussagen über einzelne reellwertige Größen angegeben und nicht durch Angabe eines reellwertigen Vektors, wie ihn eigentlich ein Neuron repräsentiert. Unter bestimmten Voraussetzungen sind diese Vorgehensweisen jedoch äquivalent, so daß ein Neuron $u = (c, w)$ einer Fuzzy-Regel der Form

$$\textbf{if } (x_1 \approx c_1) \wedge \cdots \wedge (x_n \approx c_n) \textbf{ then } (y \approx w).$$

entspricht. Wird die Aktivierungsfunktion beispielsweise durch die Gaußsche Aktivierungsfunktion (4.2) $\alpha_u = \gamma_\sigma$ definiert und verwendet man als Fuzzy-Mengen die Funktionen

$$\mu_{c_i} : I\!R \to [0,1], x \mapsto \exp\left(-\frac{(x-c)^2}{\sigma^2}\right), \tag{4.38}$$

und wird die konjunktive Verknüpfung von Fuzzy-Aussagen durch die Multiplikation festgelegt, so folgt direkt aus der multiplikativen Homomorphie der Exponentialfunktion die einfache Beziehung

$$\alpha_u = \mu_{c_1} \wedge \mu_{c_2} \wedge \cdots \wedge \mu_{c_n} \tag{4.39}$$

für die Bestimmung des Erfülltheitsgrades der Prämisse $x \approx c$ der zu u gehörigen Fuzzy-Regel.

Für praktische Anwendungen dieser Fuzzy-Regeln zur Kodierung einer Regelbasis in RBF oder DCS-Netzwerken ergibt sich ein schwerwiegendes Problem, welches an folgendem Beispiel deutlich wird. Zur Steuerung eines Roboters gehe man von einem dreidimensionalen Eingaberaum I aus, dessen Eingabegrößen aus zwei Seitenabständen und einem Frontalabstand bestehen. Für Abstände werden jeweils $k \in I\!N$ Kategorien eingeführt, die mit wachsendem Index einen größeren Abstand kennzeichnen. Für jede Regel, mit der das Verhalten des Roboters beschrieben wird, muß die Prämisse Aussagen über jede Komponente des Eingaberaumes treffen. Die Modellierung von Regeln der Form

„Wenn der Frontalabstand klein ist, dann setze die Geschwindigkeit auf kleinen Wert“

ist bisher nicht möglich, da die Prämisse keine Aussagen über die beiden Seitenabstände macht. Dennoch werden solche Regeln für die Steuerung des Roboters benötigt, da beispielsweise für die Geschwindigkeit des Fahrzeuges nur der frontale Abstand, nicht aber der Seitenabstand wichtig ist. Um diese Regel mit den bisher zur Verfügung gestellten Mitteln zu modellieren, wären viele Regeln nötig, die jede Kombination von Kategorien des linken und rechten Seitenabstandes zusammen mit der Forderung *Frontalabstand ist klein* als Prämissen abdecken. Damit würden für die Modellierung einer solchen Regel k^2 Regeln benötigt werden. Da ein Regler sicher mehrere solche Regeln benötigt und oftmals der Eingaberaum auch höherdimensional ist, führt diese Vorgehensweise zu einer kombinatorischen Explosion der Regelanzahl.

4.2.2 FDCS-Netzwerke

Zur Lösung dieses Problems werden jetzt Neuronen zugelassen, die nicht vollständig mit der Eingabeschicht verbunden sind. Dabei geht die anschauliche Vorstellung eines Neurons als

Punkt im Raum $I \times O \subseteq \mathbb{R}^{n+m}$ verloren. Vielmehr gewinnt eine solche neuronale Einheit mit jeder Eingabekomponente, mit der sie nicht verbunden ist, einen Freiheitsgrad, so daß Neuronen in dieser Sichtweise besser als affine Teilräume von $\mathbb{R}^{n+m}$ aufzufassen sind. Und wenn Neuronen mit der Eingabeschicht nicht vollständig verbunden sein müssen, dann kann auch die Forderung nach der vollständigen Verbindung mit der Ausgabeschicht aufgehoben werden. Die oben beschriebenen Probleme bei der Modellierung einer Regel, deren Prämisse nicht über alle Komponenten des Eingaberaumes Aussagen trifft, sind damit behoben. Ein Neuron, welches eine derartige Regel modellieren soll, darf nur mit den Eingabekomponenten verbunden sein, die in der Prämisse der zugehörigen Regel auch vorkommen. Wegen der Verträglichkeit solcher Neuronen mit Fuzzy-Regeln wird der Begriff des FDCS-Netzwerkes (oder auch Fuzzy-DCS-Netzwerkes) eingeführt, welches aus einem DCS-Netzwerk durch Ergänzung um nicht vollständig verbundene Neuronen hervorgeht.
Dazu betrachte man wieder einen Eingaberaum $I \subseteq \mathbb{R}^n$ und einen Ausgaberaum $O \subseteq \mathbb{R}^m$. Als Neuronen werden nicht nur Elemente aus $I \times O$ zugelassen, sondern die Komponenten dürfen zusätzlich den *undefinierten Wert* $\perp$ besitzen. Neuronen, die ein undefiniertes Ein- oder Ausgabegewicht besitzen, sind in der entsprechenden Komponente nicht mit der Ein- oder Ausgabeschicht des Netzwerkes verbunden. Als neue Neuronenmenge erhält man somit eine endliche Menge U mit

$$U \subseteq \left\{ z \in (\mathbb{R} \cup \{\perp\})^{n+m} \middle| \exists x \in I \times O \quad \forall i \in \{1, \cdots, n+m\} : z_i \in \{x_i, \perp\} \right\}. \tag{4.40}$$

Die Elemente aus U werden hier als *verallgemeinerte Neuronen* bezeichnet. Ist zusätzlich $L \subseteq U^2 \setminus id_U$ wieder eine Menge von lateralen Neuronenverbindungen und sind

$$\begin{aligned} \rho : U^2 \to [0,1], \quad & \alpha : U \to [0,1]^I, \\ r : U \to \mathbb{R}_{\geq 0}, \quad & \delta_{\mathcal{F}} : I \to O \end{aligned}$$

Abbildungen wie im DCS-Netzwerk, so wird durch $\mathcal{F} = (I, O, U, L, \rho, \alpha, r, \delta_{\mathcal{F}})$ ein *FDCS-Netzwerk* definiert, falls zusätzlich die folgende Bedingung gilt:

$$\forall i \in \{1, \cdots, m\} \quad \exists u \in U : \quad (w_u)_i \neq \perp. \tag{4.41}$$

Sie besagt, daß zu jeder Ausgabekomponente ein Neuron existieren muß, das mit dieser Komponente verbunden ist. Dieses ist eine notwendige Forderung, damit die Evaluierungsfunktion $\delta_{\mathcal{F}}$ wohldefiniert ist.
Als Evaluierungsfunktionen kommen ähnliche Funktionen wie beim DCS-Netzwerk in Betracht. Es muß lediglich berücksichtigt werden, daß die Neuronen nun nicht mehr vollständig mit der Ein- und Ausgabeschicht verbunden sein müssen.
Wird die Evaluierungsfunktion durch Summation über alle Neuronen gemäß

$$\begin{aligned} & \delta_{\mathcal{F}} : I \to O, \quad x \mapsto y, \quad \text{wobei} \\ & y_i = \sum_{u \in U \wedge (w_u)_i \neq \perp} \alpha_u(x) \cdot (w_u)_i, \quad i \in \{1, \cdots, m\} \end{aligned} \tag{4.42}$$

erhalten, so ist die Forderung (4.41) sogar hinreichend, damit $\delta_{\mathcal{F}}$ wohldefiniert ist.

Wird dagegen die Evaluierungsfunktion, wie es bei DCS-Netzwerken üblich ist, durch

$$\begin{aligned} &\delta_{\mathcal{F}} : I \to O, \quad x \mapsto y, \quad \text{wobei} \\ &y_i = \sum_{u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_i \neq \perp} \alpha_u(x) \cdot (w_u)_i, \quad i \in \{1, \cdots, m\} \end{aligned} \tag{4.43}$$

definiert, so muß die Forderung (4.41) durch

$$\forall v \in U : \quad \forall i \in \{1, \cdots, m\} \quad \exists u \in \{v\} \cup Nh(v) : \quad (w_u)_i \neq \perp. \tag{4.44}$$

ersetzt werden.
Ebenso ist eine normalisierte Evaluierungsfunktion für FDCS-Netzwerke möglich.

$$\begin{aligned} &\tilde{\delta}_{\mathcal{F}} : I \to O, \quad x \mapsto y, \quad \text{wobei für } i \in \{1, \cdots, m\} \\ &y_i = \frac{1}{\sum_{u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_i \neq \perp} \alpha_u(x)} \cdot \sum_{u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_i \neq \perp} \alpha_u(x) \cdot (w_u)_i. \end{aligned}$$

Zusätzlich müssen auch die Aktivierungsfunktionen an die unvollständigen Verbindungen mit der Eingabeschicht angepaßt werden. Für ein Neuron $u \in U$ mit Aktivierungsradius $\sigma \in I\!R_{>0}$ werden durch

$$\alpha_u : I \to [0,1], \quad x \mapsto \exp\left(- \sum_{i \in \{1, \cdots, n\} \wedge (c_u)_i \neq \perp} \frac{(x_i - (c_u)_i)^2}{\sigma^2} \right), \tag{4.45}$$

$$\alpha_u : I \to [0,1], \quad x \mapsto \frac{1}{1 + \sum_{i \in \{1, \cdots, n\} \wedge (c_u)_i \neq \perp} \sigma \cdot (x_i - (c_u)_i)^2}. \tag{4.46}$$

zwei mögliche Aktivierungsfunktionen für den Gebrauch in FDCS-Netzwerken angegeben. Die Lernverfahren für DCS-Netzwerke lassen sich problemlos an FDCS-Netzwerke anpassen. Die Adaption der Ausgabegewichte wird weiterhin durch die Deltaregel vorgenommen, wobei natürlich nur die Gewichte $(w_u)_i \neq \perp$ adaptiert werden.
Auch die Kohonen-Regel zur Adaption der Zentren wird ebenso übertragen. Der einzige Punkt für den sich etwas wesentliches geändert hat, ist die Bestimmung der zu einer Eingabe am besten passenden Neuronen *bmu* und *second*. Statt über die Euklidische Norm den Abstand zwischen Neuronenzentren und Eingaben zu bestimmen, muß ein neues Abstandsmaß eingeführt werden, das den Abstand einer Eingabe $x \in I$ vom Zentrum c_u eines verallgemeinerten Neurons $u \in U$ bestimmt:

$$d(x, c_u) := \sqrt{\sum_{i \in \{1, \cdots, n\} \wedge (c_u)_i \neq \perp} (x_i - (c_u)_i)^2} \tag{4.47}$$

Mit diesem Abstandsmaß anstelle der Euklidischen Norm lassen sich entsprechend zu (4.8) und (4.9) die beiden zu einer Eingabe am besten passenden Neuronen *bmu* und *second* definieren.

Eine entscheidende Eigenschaft der DCS-Netzwerke ist die Zugehörigkeit zur Klasse der von Martinetz eingeführten topologierepräsentierenden Netzwerke. Durch die Verallgemeinerung der DCS-Netzwerke zu FDCS-Netzwerken ist es allerdings fraglich, ob die Eigenschaft der perfekten Toplogieerhaltung trotz der Verallgemeinerung erhalten bleibt. Da Neuronen nun nicht mehr als Punkte im Raum $I \times O$ aufzufassen sind und der klassische Begriff der Voronoi-Zellen nicht mehr zur Verfügung steht, sind FDCS-Netzwerke nicht mehr in dem Sinne perfekt topologieerhaltend, wie es in [MS93] definiert ist. Allerdings lassen sich alle dort vorkommenden Begriffe sinnvoll erweitern, so daß die theoretischen Resultate von Martinetz auch auf FDCS-Netzwerke übertragbar sind. Diese Fragestellung wird ausführlich im Anhang B aufgegriffen, wo die Resultate von Martinetz auf FDCS-Netzwerke übertragen werden.

4.2.3 Interpolation mit FDCS-Netzwerken

Die berechnete Ausgabe aller bisher besprochenen Netzwerktypen stellt eine Approximation einer Funktion dar, deren Funktionswerte an den Neuronenzentren durch die Ausgabegewichte der Neuronen gegeben sind. Ist der Träger der Aktivierungsfunktionen gleich dem gesamten Eingaberaum, wie dieses beispielsweise bei den Gaußschen Basisfunktionen der Fall ist, so ist die Aktivierung eines Neurons am Zentrum eines Nachbarneurons echt größer als Null. Daher liefert die Linearkombination der mit den Neuronenaktivitäten gewichteten Ausgabegewichten an den Stellen der Neuronenzentren eine Ausgabe, die von dem vorgegebenem Stützvektor abweicht. Das Netzwerk leistet somit also lediglich eine Approximation der durch die Neuronen vorgegebenen Stützvektoren aber keine Interpolation, da bei der Interpolationsaufgabe gefordert ist, daß die erzeugte Funktion an den Stützstellen die durch die Ausgabegewichte vorgegebenen Stützwerte exakt hält.
Für klassische neuronale Netzwerke spielt diese Eigenschaft keine Rolle, da durch die sukzessive Adaption der Gewichte der Approximationsfehler minimiert wird. Da FDCS-Netzwerke aber zusätzlich die Möglichkeit der Kodierung von Vorwissen in Form von Fuzzy-Regeln besitzen, ist es wünschenswert, die vorgegebenen Stützstellen nicht nur zu approximieren, sondern auch zwischen ihnen zu interpolieren. Die Interpolation ist dann notwendig, wenn das Vorwissen bestimmte Randbedingungen der zu approximierenden Funktion kodiert, die auf jeden Fall erfüllt werden sollen. Am Beispiel der Hindernisvermeidung betrachtet, könnte dieses eine Regel sein, die die Geschwindigkeit des Fahrzeuges auf jeden Fall auf Null setzt, wenn der frontale Abstand $\approx 200\ mm$ beträgt. Diese Regel ist für die Sicherheit des Fahrzeuges von so großer Bedeutung, daß sie auch während einer Adaption des Reglers nicht geändert und auch durch Approximation mit anderen Neuronen nicht überschrieben werden darf.
Als Lösung dieses Problems genügt es, die Evaluierungsfunktion des Netzwerkes anzupassen und die Neuronen auszuzeichnen, deren Ausgabegewichte an den Neuronenzentren exakt gehalten werden sollen. Es ist klar, daß für diese Art von Neuronen eine vollständige Verbindung mit der Ausgabeschicht vorausgesetzt werden muß.
Für eine formale Definition solcher Netzwerke betrachte man ein FDCS-Netzwerk $(I, O, U, L, \rho, \alpha, r, \delta)$. Die Neuronen, deren Stützvektoren an ihren Zentren exakt gehalten werden sollen, werden als *exakte Neuronen* definiert. Um sie zu zu kennzeichnen wird eine Teilmenge $\Pi \subseteq U$ eingeführt, die aus diesen Neuronen besteht.

Formal wird dann das 9-Tupel $\mathcal{F} = (I, O, U, L, \Pi, \rho, \alpha, r, \delta_{\mathcal{F}})$ als *π-FDCS-Netzwerk* definiert, falls $\delta_{\mathcal{F}}$ eine neue Evaluierungsfunktion ist und zusätzlich folgende Bedingung gilt:

$$\forall u \in \Pi : \quad w_u \in O. \tag{4.48}$$

Durch sie wird gefordert, daß die Ausgabegewichte der exakten Neuronen vollständig definiert sind.
Anschaulich kann ein π-FDCS-Netzwerk wie in Abbildung 4.4 dargestellt werden.

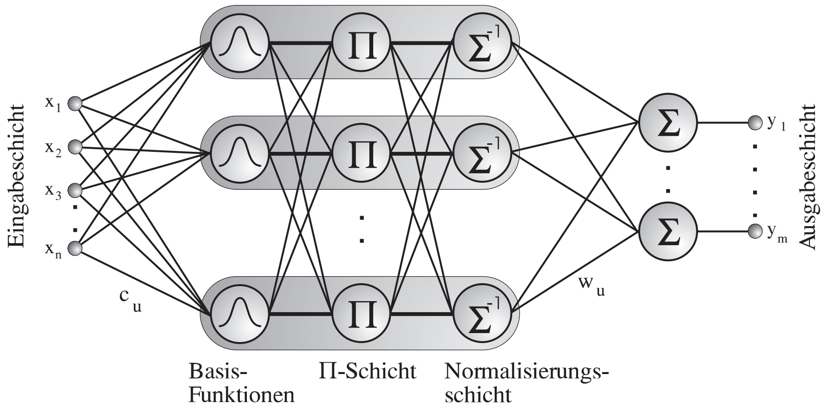


Abbildung 4.4: Modell eines π-FDCS-Netzwerkes

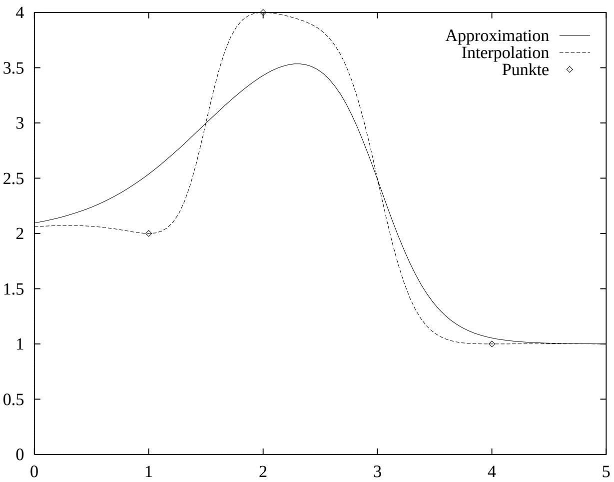


Abbildung 4.5: Interpolation in π-FDCS-Netzwerken.

Die Interpolation durch die exakten Neuronen wird durch die Evaluierungsfunktion $\delta_{\mathcal{F}}$ gewährleistet, für die die Normalisierung anders als bei FDCS-Netzwerken zwingend benötigt wird.

$$\delta_{\mathcal{F}} : I \to O, \quad x \mapsto y, \quad \text{wobei für } i \in \{1, \cdots, m\} \text{ gilt:}$$

$$y_i = \frac{1}{\sum\limits_{u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_i \neq \perp} \bar{\alpha}_u(x)} \cdot \sum_{u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_i \neq \perp} \bar{\alpha}_u(x) \cdot (w_u)_i. \quad (4.49)$$

$$\text{Dabei ist } \bar{\alpha}_u(x) = \alpha_u(x) \cdot \prod_{v \in \Pi \setminus \{u\}} (1 - \alpha_v(x)). \quad (4.50)$$

Jedes Neuron besteht aus einer Basisfunktion zur Berechnung der Aktivität, einer Verarbeitungseinheit, die die Aktivität mit dem Produkt der Komplemente der Nachbaraktivitäten multipliziert und einer Verarbeitungseinheit, die die Normalisierung berechnet.
In Abbildung 4.5 wird die Interpolationseigenschaft der π-FDCS-Netzwerke veranschaulicht. Die eine Kurve zeigt die Evaluierungsfunktion eines normalisierten RBF-Netzwerkes zu den Neuronen $(1, 2)$, $(2, 4)$ und $(4, 1)$. Die andere Kurve zeigt den Verlauf der Evaluierungsfunktion eines RBF-Netzwerkes mit exakten Neuronen. Die gegebenen Stützwerte werden von der Funktion exakt gehalten.
Ein formaler Nachweis, daß π-FDCS-Netzwerke die Interpolationseigenschaft erfüllen, wird in Anhang B gegeben.

Kapitel 5

Fuzzy-basierte Hindernisvermeidung

In diesem Kapitel werden die im vorherigen Abschnitt entworfenen Regler auf das Problem der Hindernisvermeidung eines autonomen mobilen Roboters angewendet.
Dazu werden zwei Architekturen vorgestellt. In der ersten Architektur wird versucht, den verhaltensbasierten Ansatz so weit wie möglich aufrecht zu erhalten. Die dabei auftretenden Probleme führen dann zu einer zweiten Architektur, die eine stärkere Vorverarbeitung der Ultraschalldaten benötigt.

5.1 Verhaltensbasierter Fuzzy-Ansatz zur Hindernisvermeidung

In Kapitel 3 wurde ein verhaltensbasierter Ansatz zur Hindernisvermeidung eines autonomen mobilen Roboters angegeben, bei dem das Gesamtverhalten aus hierarchisch angeordneten Teilverhalten zusammengesetzt ist.
Dabei wurde das Fahrzeug zunächst durch eine konstante Geschwindigkeit und fünf ausgewählte Rotationsbewegungen gesteuert. Diese sehr eingeschränkten Kontrollmöglichkeiten erlauben nur eine relativ geringe Fahrtgeschwindigkeit des AMR's (ca. 200 $\frac{mm}{s}$) und führen bei höheren Geschwindigkeiten zu starken Oszillationen des Fahrzeuges.
Das Fahrverhalten konnte dann durch kontinuierliche Regelung der Fahrtgeschwindigkeit über eine 3-Sekunden-Regel verbessert werden. Die Kontrolle der Rotationsgeschwindigkeit wurde dabei ebenfalls an die Fahrtgeschwindigkeit angepaßt.
Ferner wurden die Meßwerte der Ultraschallsensoren nicht direkt zur Steuerung des Systems verwendet, sondern die Meßwerte wurden in zwei Klassen unterteilt und die Steuerung nur von der Klassenzugehörigkeit der gemessenen Werte abhängig gemacht. Auch durch diese schwache Ausnutzung der Sensordaten werden Oszillationen hervorgerufen.
Ein möglicher Lösungsansatz ist die Verwendung eines Reglers zur Steuerung der AMR-Bewegung, der sowohl kontinuierliche[1] Motorstellgrößen liefert und ebenso von kontinuierlichen Meßwerten seine Ausgaben abhängig macht.

[1] Mit *kontinuierlich* ist hier nicht wie in der Regelungstechnik üblich, eine analoge Steuerung gemeint, sondern eine Kontrolle, deren digitale Ein- und Ausgangsgrößen als Fließkommazahlen verarbeitet werden.

Ein möglicher Regler, der diese Forderungen erfüllt, ist der im vorherigen Abschnitt vorgestellte FDCS-Regler, der durch seine Konzeption als neuronales Netz zudem die Möglichkeit der Adaptivität bietet.
Zunächst wird nicht auf die Lernfähigkeit eines solchen Reglers eingegangen. In diesem Kapitel wird erst gezeigt, wie ein auf FDCS-Reglern basierendes System in der Lage ist, das Problem der Hindernisvermeidung mit vorgegebenen Fuzzy-Regeln zu lösen.

5.1.1 FDCS-basierte Architektur zur Hindernisvermeidung

In einem ersten Versuch wurde direkt die in Kapitel 3 vorgestellte verhaltensbasierte Architektur so abgeändert, daß bis auf die beiden untersten Verhaltensebenen (Entkomme und Vermeide Gefahrensituation) alle Teilverhalten durch separate FDCS-Regler ausgetauscht wurden. Damit bleibt zum einen der verhaltensbasierte Ansatz vollständig erhalten und zum anderen wird somit jedes Teilverhalten durch einen Fuzzy-Regler realisiert (vgl. [Rei94]). Ein dabei auftretendes Problem schränkt diesen Fuzzyfizierungsansatz stark ein. Während im verhaltensbasierten Ansatz aus Kapitel 3 die Auswahl der aktiven Verhaltensweisen sehr einfach durchgeführt wird, ist sie hier nicht mehr so einfach zu realisieren. Im Mataric-Ansatz schlägt jedes Teilverhalten gemäß seiner Hierarchie eine Aktion vor oder gibt die Kontrolle an die nächst höhere Ebene ab. Da im Fuzzy-Ansatz jedes Teilverhalten durch die Evaluierungsfunktion des Fuzzy-Reglers ausgedrückt wird, schlägt jedes Teilverhalten grundsätzlich zu jedem Eingabezustand eine Aktion vor. Somit ist die Auswahl aktiver Verhaltensweisen also explizit von außen vorzugeben. Eine mögliche Verhaltensauswahl besteht darin, für jedes Teilverhalten eine Zustandsbewertung zu definieren, die den Grad der Zuständigkeit eines Teilverhaltens aus der aktuellen Eingabe berechnet.
Liegt die Bewertung unterhalb eines vorgegeben Schwellwertes, so ist das betreffende Teilverhalten zuständig und die berechnete Ausgabe wird als aktuelle Motorstellgröße verwendet. Sonst wird die Kontrolle an das nächsthöhere Verhalten abgegeben. Die oberste Verhaltensebene benötigt keine solche Zuständigkeitsklärung mehr. Ihre Ausgaben werden immer an die Motorik weitergeleitet, wenn die Zustandsbewertungen aller unteren Verhaltensebenen oberhalb gewisser Schwellwerte liegen.
Da die untersten Verhaltensebenen im Ansatz aus Kapitel 3 komplexere Bewegungsabläufe wie das Zurücksetzen und die Neuorientierung des Fahrzeuges einleiten, bleiben diese erhalten und werden nicht als Fuzzy-Regler implementiert.
Dieser erste Versuch der Fuzzyfizierung hat sich bereits im Simulator als nicht sinnvoll herausgestellt, da mit ihm folgende Probleme verbunden sind. Durch die Verhaltensauswahl mit einer expliziten Zuständigkeitsberechnung der Teilverhalten wird der Vorteil der glatten Steuerung der Fuzzy-Regler wieder aufgehoben. Das Umschalten zwischen zwei Verhaltensebenen findet nicht auf eine weiche Art statt, sondern führt zu einer sehr abrupten Verhaltensänderung des AMR's, woraus teilweise starke Oszillationen im Fahrverhalten resultieren können.
Um dieses zu verhindern, ist die Auswahl der aktuellen Verhaltensweise so zu implementieren, daß auch der Wechsel zwischen zwei in der Hierarchieordnung aufeinanderfolgenden Teilverhaltensweisen auf eine glatte Art realisiert wird. Diese Idee wird von dem folgenden Fuzzy-basierten Ansatz berücksichtigt.

Man betrachte zwei π-FDCS-Netzwerke $\mathcal{A}$ und $\mathcal{B}$, die zwei hierarchisch aufeinanderfolgende Teilverhalten repräsentieren. $\mathcal{A}$ sei dabei das tiefer liegende und somit das für die Sicherheit des AMR wichtigere Verhalten. $\delta_{\mathcal{A}} : I_{\mathcal{A}} \rightarrow O$ und $\delta_{\mathcal{B}} : I_{\mathcal{B}} \rightarrow O$ seien die zugehörigen Evaluierungsfunktionen, wobei die Eingaberäume auch verschieden sein dürfen, um jedem Netzwerk nur die für das jeweilige Verhalten relevanten Merkmale aus den Sensordaten als Eingabe zu präsentieren. Eine *Zustandsbewertung*

$$eval : I_{\mathcal{A}} \rightarrow [a, b], \quad a, b \in \mathbb{R} \tag{5.1}$$

gibt an, wie gefährlich der momentane Zustand für das Roboterfahrzeug ist und liefert damit einen Grad der Zuständigkeit des Teilverhaltens $\mathcal{A}$ um die drohende Gefahr vom AMR abzuwenden. Diese Zustandsbewertung ist bei biologischen Systemen mit einem *Schmerz* vergleichbar, der bei genügend großer Intensität einen Verhaltenswechsel des Lebewesens hervorruft. Entsprechend wird gemäß der Zustandsbewertung ein *Schmerzsignal*

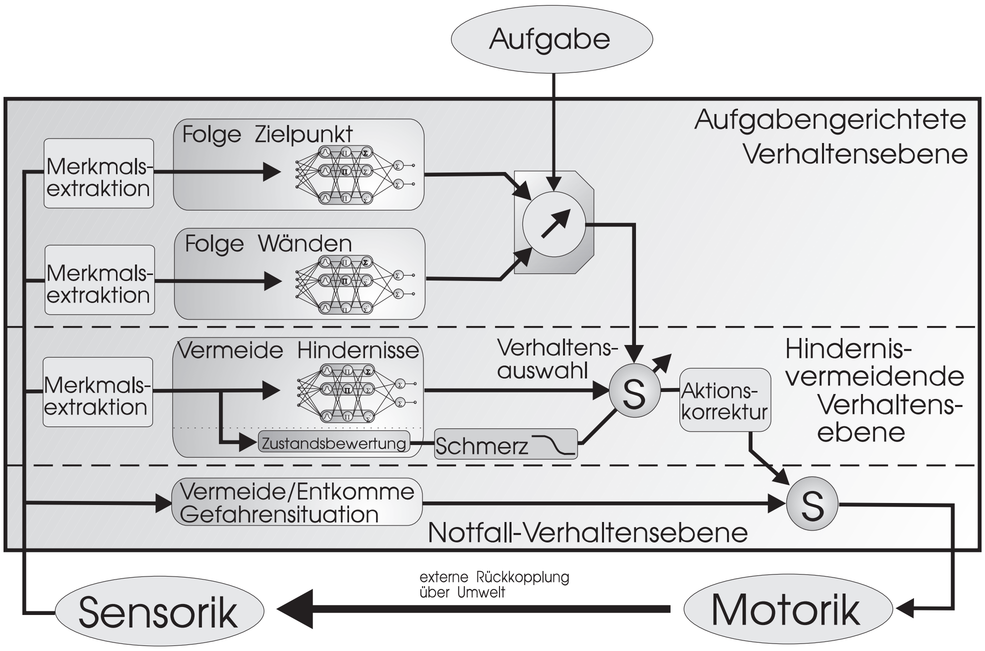


Abbildung 5.1: Verhaltensbasierte Fuzzy-Architektur für die Hindernisvermeidung

$p : [a, b] \rightarrow [0, 1]$ erzeugt, das angibt wie stark das Verhalten $\mathcal{A}$ an der Reaktion des Roboters beteiligt sein soll. Ist p eine stetige, monoton fallende Funktion mit $p(a) = 1$ und $p(b) = 0$, so wird durch die Konvexkombination

$$y = p(eval(x_{\mathcal{A}})) \cdot \delta_{\mathcal{A}}(x_{\mathcal{A}}) + (1 - p(eval(x_{\mathcal{A}})) \cdot \delta_{\mathcal{B}}(x_{\mathcal{B}}) \tag{5.2}$$

eine Aktion $y \in O$ zu den Eingaben $x_{\mathcal{A}} \in I_{\mathcal{A}}$ und $x_{\mathcal{B}} \in I_{\mathcal{B}}$ berechnet. Bei einer geringen Zustandsbewertung $eval(x_{\mathcal{A}})$ wird hauptsächlich das niedrigere Teilverhalten die Aktion bestimmen, während bei höheren Zustandsbewertungen das höhere Verhalten die Aktionsberechnung dominiert.

Die Größen a und b geben die kleinste und größte Zustandsbewertung an. Als ein Beispiel für ein Schmerzsignal betrachte man die folgende Funktion, die man durch Überlagerung einer sigmoiden und einer linearen Funktion erhält:

$$p(z) = \varphi(z) - \varphi(b) + \frac{\varphi(a) - \varphi(b) - 1}{b - a} \cdot (z - b), \quad \text{wobei} \tag{5.3}$$

$$\varphi(z) = \frac{1}{1 + \exp(\varrho \cdot (x - m))}. \tag{5.4}$$

Der Parameter $m \in [a, b]$ ist die Stelle des Wendepunktes und $\varrho \in I\!R$ ein Parameter, der die Steilheit der sigmoiden Funktion angibt. Mit $\varrho = 0$ erhält man für p eine lineare Funktion und mit $\varrho \to \infty$ erhält man für p eine Schwellwertfunktion mit $p(z) = 1$ für $z < m$ und $p(z) = 0$ für $z \geq m$. In Abbildung 5.2 ist eine Schmerzfunktion p dargestellt.

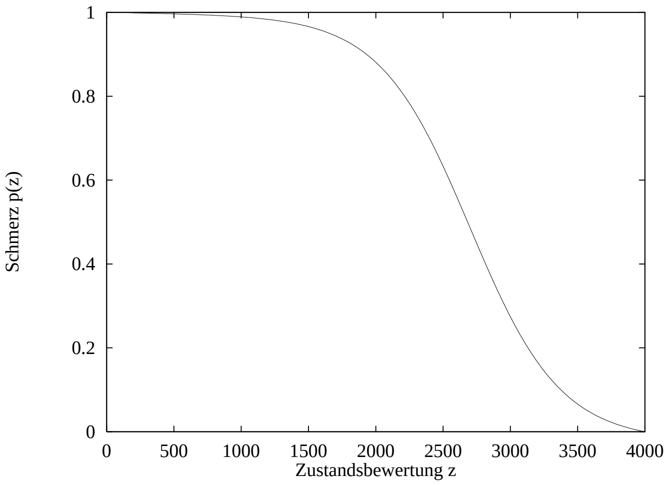


Abbildung 5.2: Schmerzfunktion für $a = 0$, $b = 4000$, $m = 2700$ und $\varrho = 0.003$

Auf diese Weise werden zwei Verhaltensebenen miteinander verknüpft. Die Aufteilung der Verhaltensweisen wird daher gegenüber der Aufteilung aus Kapitel 3 abgeändert.
Die unterste Ebene besteht nach wie vor aus der Notfallverhaltensebene, in der die Behandlung von Notfallsituationen auf klassische Weise durchgeführt wird. Die Verhaltensweisen der Vermeidung von frontalen und seitlichen Hindernissen und des Passierens von Korridoren werden dagegen zusammengefaßt und durch einen FDCS-Regler in der hindernisvermeidenden Verhaltensebene realisiert. Das Wandfolgeverhalten bzw. Zielverfolgungsverhalten bildet die aufgabengerichtete Verhaltensebene (vgl. Abb. 5.1). Zwischen der hindernisvermeidenden und der aufgabengerichteten Verhaltensebene findet die über die Schmerzsignal getriebene Auswahl von Verhaltensebenen statt.

Merkmalsextraktion

Jedem FDCS-Regler der Hindernisvermeidung ist eine eigene *Merkmalsextraktion* vorgeschaltet, in der die für das jeweilige Teilverhalten wichtigen Merkmale aus der durch die

Sensorik gelieferten momentanen Zustandsbeschreibungen aufbereitet werden. Eine Beschreibung des momentanen Zustands des Fahrzeuges besteht aus allen acht Sensorwerten, der aktuellen Fahrtgeschwindigkeit und der überlagerten Rotationsgeschwindigkeit des AMR's. Nicht alle diese Daten sind für jedes Teilverhalten wichtig.
Für den FDCS-Regler der hindernisvermeidenden Verhaltensebene besteht der Eingaberaum aus allen acht Sensormeßwerten.
Der Eingaberaum des Wandfolgeverhaltens besteht dagegen nur aus zwei reellen Größen, die die Orientierung des Fahrzeuges zur linken und rechten Wand angeben. Verwendet man eine der Sensorkonfigurationen aus Abb. 3.2, so können diese Merkmale durch

$$\alpha_{right} = \arctan\left(\frac{sensor[5] - sensor[4]}{d_v}\right) \cdot \frac{180°}{\pi}, \tag{5.5}$$

$$\alpha_{left} = \arctan\left(\frac{sensor[7] - sensor[6]}{d_v}\right) \cdot \frac{180°}{\pi} \tag{5.6}$$

aus den Sensorwerten extrahiert werden. Die Größe d_v gibt den vertikalen Abstand der Seitensensoren an.
Die Merkmalsextraktion für die Zielverfolgung liefert schließlich die Abstände Δx und Δy des Fahrzeuges in x und in y-Richtung zum Zielpunkt im egozentrischen Roboterkoordinatensystem als Eingabegrößen des zugehörigen FDCS-Reglers.
Neben den hier beschriebenen extrahierten Merkmalen sind noch weitere Merkmale denkbar. Analog zur Bestimmung der Fahrzeugneigung zu seitlichen Hindernissen kann ebenso die Orientierung eines frontalen Hindernisses mit den Sensoren # 1 und # 2 bestimmt werden. Möchte man ferner in der Prämisse einer Fuzzy-Regel zwei Sensorwerte miteinander vergleichen (z. B. $s_i < s_j$), so kann man deren Differenz $s_i - s_j$ als weiteres Merkmal verwenden. Auf eine stärkere Merkmalsextraktion und Aufbereitung der Sensorwerte wird im Abschnitt 5.2 näher eingegangen.

Aktionskorrektur

Als zusätzliche Sicherheitskomponente wird in dieser Hindernisvermeidungsarchitektur eine *Aktionskorrektur* verwendet, die alle Aktionen, die durch die FDCS-Regler berechnet werden, daraufhin überprüft, ob sie für den Roboter gefährlich sind. Der Einsatz einer solchen Komponente ist ratsam, da durch die Approximation der FDCS-Regler nicht mit Sicherheit vorhergesagt werden kann, wie groß die berechneten Stellgrößen der Motorik ausfallen. Eine Abschätzung der Beträge der zu erwartenden Stellgrößen wird insbesondere dann problematisch, wenn die FDCS-Regler um die Lernfähigkeit ergänzt werden. Um das System vor Gefahren zu schützen, werden durch die Aktionskorrektur zu große Rotations- und Fahrtgeschwindigkeiten unterdrückt. Negative Fahrtgeschwindigkeiten werden ebenfalls abgefangen. In diesen Fällen wird eine berechnete Aktion durch eine maximal zulässige Aktion ersetzt. Die durch die Aktionskorrektur weitergeleitete Aktion berechnet sich durch

$$\tilde{a} = \left\{ \begin{array}{ccl} a & : & a \in [l_a, r_a] \\ l_a & : & a < l_a \\ r_a & : & a > r_a \end{array} \right. . \tag{5.7}$$

Die Größe $a \in \{v, \omega\}$ gibt eine vorläufige durch den FDCS-Regler berechnete Aktion an und $[l_a, r_a]$ ist der für die jeweilige Stellgröße zulässige Bereich. Der Wert $\tilde{a}$ gibt schließlich die tatsächlich weitergeleitete Aktion an. Dabei bezeichnet v die zu regelnde Fahrtgeschwindigkeit und ω die zu regelnde Rotationsgeschwindigkeit.
Eine Aktionskorrektur, die jederzeit garantieren kann, daß die Fahrtgeschwindigkeit nur so groß ist, daß eine Annäherung des Fahrzeuges bis auf eine Entfernung von d_{stop} an ein frontales Hindernis mindestens eine Zeit T in Anspruch nimmt, wurde bei [BB+95] in Form einer 2-Sekunden-Regel ($T = 2\ s$) angegeben. Auch für die variable Geschwindigkeitskontrolle in Kapitel 3 wurde diese Regel bereits verwendet. Gibt d_{min} den minimalen frontalen Abstand zu einem Hindernis an, so berechnet sich die maximal zulässige Geschwindigkeit durch

$$r_v = \frac{d_{min} - d_{stop}}{T}. \tag{5.8}$$

5.1.2 Regelbasis für die FDCS-Regler

Im folgenden werden die Fuzzy-Regelbasen der drei FDCS-Regler angegeben. Statt linguistischen Termen werden für die Regeln direkt Zahlenwerte verwendet, die in der Prämisse den Eingabegewichten und in den Konklusionen den Ausgabegewichten eines Neurons bzw. einer Fuzzy-Regel entsprechen. Das Zeichen $\approx$ wird in der Konklusion dort verwendet, wo keine exakten Neuronen verwendet werden. Für den Gebrauch von exakten Neuronen wird stattdessen das Zeichen $=$ in der Konklusion verwendet. Dadurch wird deutlich, wann die Regeln approximiert und wann sie interpoliert werden. Statt der Notation $sensor[i]$ für einen Sensormeßwert wird die kürzere Schreibweise s_i verwendet.
Für die implementierten π-FDCS-Regler wird die Regelbasis in einer an die aus Tabelle 5.1 angelehnten Notation angegeben. Aus den Zahlenwerten in den Fuzzy-Regeln werden direkt Gewichte für die neuronalen Einheiten gewonnen.

Hindernisvermeidende Verhaltensebene

Die hindernisvermeidende Verhaltensebene faßt mehrere Verhaltensweisen zusammen, die in dem Ansatz aus Kapitel 3 noch getrennt realisiert wurden. Dennoch lassen sich diese Teilverhaltensweisen als Gruppen von Fuzzy-Regeln in diesem FDCS-Regler wiederfinden. Durch die Möglichkeit der nicht vollständig mit den Ein- und Ausgabeschichten verbundenen Neuronen lassen sich die Steuerung von Fahrtgeschwindigkeit und Rotationsgeschwindigkeit durch unterschiedliche Neuronengruppen realisieren. Die erste Regelgruppe bestimmt im wesentlichen die Fahrtgeschwindigkeit in Abhängigkeit der durch die Frontsensoren gemessenen Abstände.
Die Abbildungen 5.3 a) und b) zeigen die zu dieser Regelbasis gehörige Evaluierungsfunktion.

Aufgabengerichtete Verhaltensebene

Die Tabelle 5.4 zeigt die Regelbasis für die Wand- und Zielverfolgung. In den Konklusionen der Regeln für die Zielverfolgung wurden exakte Neuronen verwendet, damit gewährleistet

if $(s_1 \approx 250)$ **then** $v \approx 0$
if $(s_2 \approx 250)$ **then** $v \approx 0$

if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 500) \wedge (s_3 \approx 500)$ **then** $v \approx 500$
if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 1000) \wedge (s_3 \approx 1000)$ **then** $v \approx 600$
if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 2000) \wedge (s_3 \approx 2000)$ **then** $(v = 1000) \wedge (\omega = 0)$

if $(s_1 \approx 1500) \wedge (s_2 \approx 1500)$ **then** $v \approx 400$
if $(s_1 \approx 1000) \wedge (s_2 \approx 1000)$ **then** $v \approx 300$
if $(s_1 \approx 500) \wedge (s_2 \approx 500)$ **then** $v \approx 150$

Tabelle 5.1: Regelbasis zur Kontrolle der Fahrtgeschwindigkeit

ω	s_0				
s_3	0	300	700	1200	1800
0	0	30	30	30	30
300	-30	0	30	30	30
700	-30	-30	0	30	30
1200	-30	-30	-30	0	30
1800	-30	-30	-30	-30	0

ω	s_1				
s_2	0	300	700	1200	1800
0	0	30	30	30	30
300	-30	0	30	30	30
700	-30	-30	0	30	30
1200	-30	-30	-30	0	30
1800	-30	-30	-30	-30	0

Tabelle 5.2: Regelbasis zur Vermeidung frontaler Kollisionen

ω	s_4			
s_5	0	300	700	1200
0	30			
300	30	30		
700	30	30	30	
1200	30	30	30	30

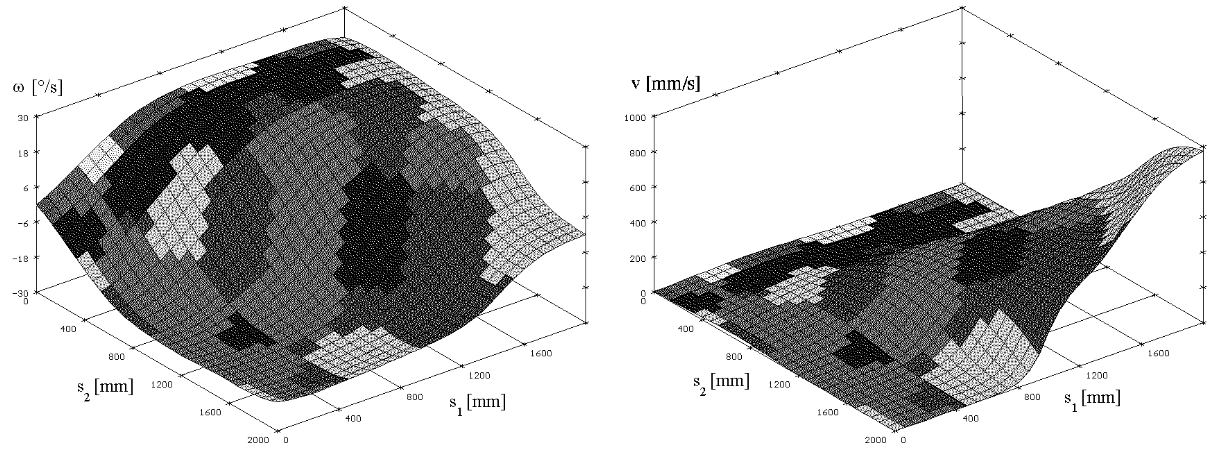
ω	s_7			
s_6	0	300	700	1200
0	-30			
300	-30	-30		
700	-30	-30	-30	
1200	-30	-30	-30	-30

Tabelle 5.3: Regelbasis zur Vermeidung seitlicher Kollisionen

wird, daß das Fahrzeug beim Erreichen des Zielpunktes wirklich die Geschwindigkeit 0 erhält.

5.1.3 Simulationen

In der Simulation hat sich der FDCS-basierte Ansatz zur Hindernisvermeidung gegenüber dem Ansatz aus Kapitel 3 in wesentlichen Punkten als überlegen herausgestellt. Versuche mit dem realen Roboter konnten aufgrund zu geringer Rechenleistung des Laptops nicht durchgeführt werden.
Während die verhaltensbasierte Hindernisvermeidung aus Kapitel 3 nur geringe Fahrtge-



a) b)

Abbildung 5.3: Evaluierungsfunktion des FDCS-Reglers zur Hindernisvermeidung mit $s_0 = s_3 = \cdots = s_7 = 2000\ mm$.

Zielverfolgung			
v/ω	Δx		
Δy	-2000	0	2000
-2000	0/-30	0/30	0/30
0	0/-30	0/0	0/30
2000	400/-30	800/0	400/30

Wandverfolgung			
v/ω	α_{left}		
α_{right}	-50	0	50
-50	800/-30	800/-30	
0	800/-30	800/0	800/30
50		800/30	800/30

Tabelle 5.4: Regelbasis zur Wand- und Zielverfolgung

schwindigkeiten bei akzeptabler Fahrweise des AMR zuläßt (ca. 0.2 $\frac{m}{s}$), ist die FDCS-basierte Hindernisvermeidung in der Lage, den Roboter mit seiner Maximalgeschwindigkeit von 1 $\frac{m}{s}$ durch seine Umgebung zu navigieren. Die aktuelle Fahrtgeschwindigkeit des AMR's hängt hierbei von der Entfernung zu Hindernissen in der Umgebung ab, so daß die Maximalgeschwindigkeit nur in Umgebungen mit weit entfernten Hindernissen erreicht wird. In Regionen, bei denen das Fahrzeug nah an die Hindernisse herankommt, wird die Geschwindigkeit entsprechend reduziert, so daß im Extremfall das Fahrzeug stehen bleibt. Dann wird nur eine rotatorische Bewegung ausgeführt, um in eine ungefährlichere Umgebung zu gelangen.

Außerdem wird mit dem FDCS-basierten Ansatz eine weitgehende Vermeidung von Oszillationen erreicht. Aufgrund der kontinuierlichen Steuerung der Rotationsgeschwindigkeit, ist das Fahrzeug in der Lage, sich in Korridoren in der Mitte zu bewegen und an den Wänden parallel auszurichten. Damit wird eine nahezu oszillationsfreie Fahrt ermöglicht. In Abbildung 5.4 ist die gefahrene Strecke des AMR's in die Umgebung eingezeichnet. Die Abstände der auf der gefahrenen Tour eingezeichneten Striche geben die gefahrene Geschwindigkeit an. In größeren Freiräumen erreicht der AMR eine Geschwindigkeit bis zu 0.8 $\frac{m}{s}$ und ist damit bis zu viermal schneller als der Ansatz aus Kapitel 3 mit konstanter Geschwindigkeit. Nähert sich das Fahrzeug einem frontalen Hindernis, wird die

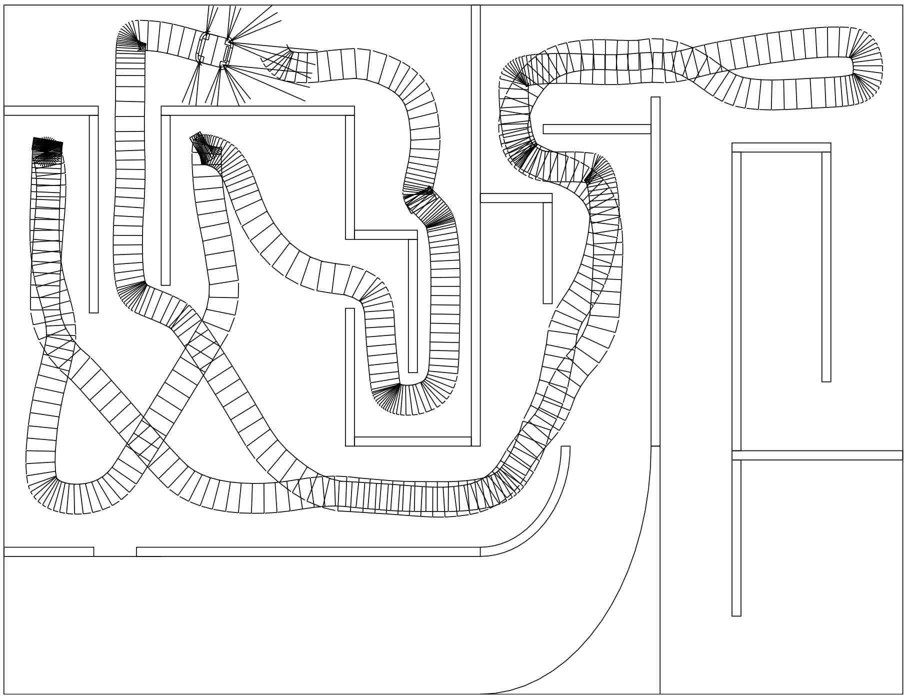


Abbildung 5.4: FDCS-gesteuerte Fahrt des AMR's

Geschwindigkeit allmählich reduziert und die Rotationsbewegung entsprechend verstärkt. Für alle FDCS-Regler wurden in dieser Simulation die Fuzzy-Regeln als Neuronen mit Gaußaktivierung und einem Aktivierungsradius von $\sigma = 0.2$ repräsentiert. Die Netzwerkauswertung wurde bei Normierung des Eingaberaumes, Normalisierung und Interpolation über alle Neuronen der Regler durchgeführt. Als Zustandsbewertung wurde die Abbildung

$$eval(s_0, \cdots, s_7) = \sqrt{s_0^2 + s_1^2 + s_2^2 + s_3^2} \tag{5.9}$$

verwendet. Die maximal meßbare Entfernung der Sensoren wurde auf 2 m gesetzt, so daß die Zustandsbewertung zwischen $a = 0$ und $b = 4000$ liegt. Als Parameter für das Schmerzsignal wurden schließlich $m = 2700$ und $\varrho = 0.003$ gesetzt und als Zeitschrittweite der Simulation wurden 0.4 s gewählt.

Trotz der Vorteile der Fuzzy-basierten Hindernisvermeidung ist dieser Ansatz in einem Punkt der Hindernisvermeidung aus Kapitel 3 unterlegen. In dem früheren Ansatz findet eine sequentielle Aktionsauswahl nicht nur zwischen den Verhaltensebenen, sondern auch innerhalb einer Teilverhaltensweise statt. Dieses liegt an der sequentiellen Untersuchung des Zustandes des AMR's durch die Verwendung von *if-then* Konstrukten, bis in einem Zweig schließlich eine Aktion bestimmt wird. Alle folgenden Verzweigungen innerhalb der Verhaltensebene werden schließlich unterdrückt. Durch diese hierarchische Zustandsauswertung innerhalb einer Verhaltensebene ist es möglich, bestimmten Reaktionen Vorrang vor anderen zu geben. Wird eine Situation abgefragt, die zu einer Linksrotation führen wird, bevor eine symmetrische Situation abgefragt wird, die zu einer Rechtsrotation des

Fahrzeuges führt, so haben bei dieser Anordnung Linkskurven grundsätzlich vor Rechtskurven Vorrang. Das so entworfene Verhalten ist also *unsymmetrisch*. Beim FDCS-Ansatz ist dagegen völlige Symmetrie einer Verhaltensebene gegeben, da alle Fuzzy-Regeln bzw. Neuronen einer Verhaltensebene gleichberechtigt sind und somit keine Regel im Netzwerk bevorzugt wird.

Dieser Sachverhalt hat schwerwiegende Folgen, wenn das Fahrzeug in einen symmetrischen Zustand gelangt, wie er sich beispielsweise ergibt, wenn der AMR in eine rechtwinklige Ecke hineinfährt. Zur Vereinfachung betrachte man nur die Sensoren # 0 und #4, die beide schräg nach vorne gerichtet die Umwelt abtasten. Der AMR bewegt sich nun in eine rechtwinklige Ecke. Zeigen beide Sensoren den gleichen Abstand an, entscheidet sich die Hindernisvermeidung für keine Ausweichrichtung, sondern reduziert nur die Geschwindigkeit. Zeigt der linke Sensor einen geringeren Abstand als der rechte Sensor an, so wird das Fahrzeug nach rechts ausweichen, woraufhin im nächsten Moment der rechte Sensor einen kleineren Wert als der linke Sensor angibt. Durch die symmetrische Behandlung von linken und rechten frontalen Hindernissen fährt der AMR geradeaus in die Ecke (vgl Abb. 5.5). Da in der Realität Messungen mit den Ultraschallsensoren gewissen Schwankungen unterworfen sind, sind exakt symmetrische Messungen sehr unwahrscheinlich. Um dieses zu berücksichtigen, wurden in der Simulation zu den Sensormeßwerten ein normalverteiltes Rauschen mit einer Varianz von ca. 1000 mm addiert. Trotzdem stellt sich in der Simulation das oben beschriebene Phänomen ein.

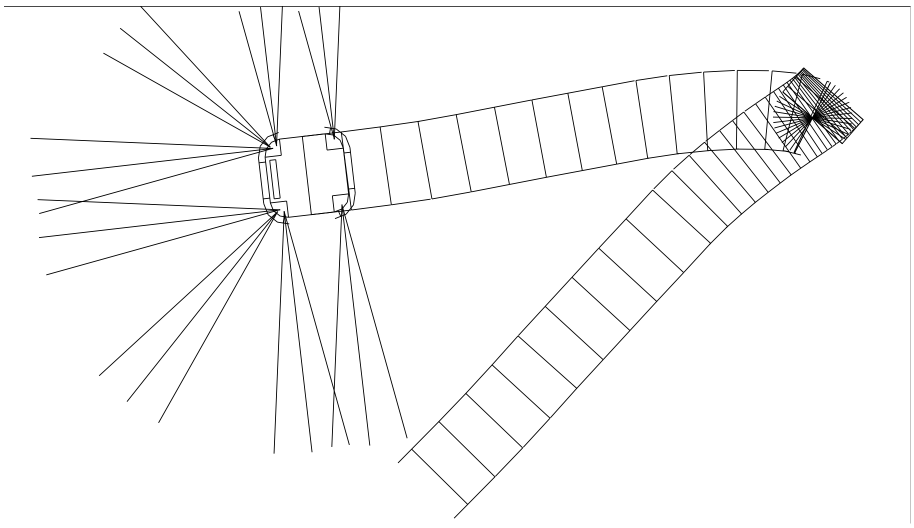


Abbildung 5.5: Probleme aufgrund der Symmetrie des Fuzzy-Reglers

Dieser Fehler der Hindernisvermeidung wird durch den Ansatz aus Kapitel 3 nicht begangen, da dort eine Ausweichrichtung Vorrang vor einer anderen Richtung hat. Da ferner die Reaktionen nicht direkt von den Sensorwerten abhängen, sondern nur davon, ob die Sensorwerte im Gefahren- oder Sicherheitsbereich liegen, führt eine solche Hindernisvermeidung das Fahrzeug aus der Ecke heraus.

5.2 Weitergehende Merkmalsextraktion aus Ultraschallmeßwerten

Um das Problem des vorherigen Abschnitts zu lösen und dennoch die Vorteile der FDCS-basierten Hindernisvermeidung zu nutzen, werden in diesem Abschnitt weitergehende Merkmale aus den Ultraschallmeßwerten extrahiert. Die Hauptaufgabe der Merkmalsextraktion besteht darin, dem Regelungssystem Eingabegrößen zu liefern, die für die Hindernisvermeidung genutzt werden können und keine symmetrischen Zustände mehr ergeben.
Ein möglicher Lösungsansatz wird durch die in [BB+95] verwendete Hindernisvermeidung motiviert. In begrenztem Umfang wird in [BB+95] die Umwelt modelliert und zur Berechnung der nächsten Aktion eine optimale Trajektorie durch die modellierte Umwelt bestimmt. Die Trajektorie wird aus Kreisbahnen ausgewählt, die durch den Ursprung des egozentrischen Roboterkoordinatensystems verlaufen. Die Tangenten dieser Kreisbahnen im Ursprung des Roboterkoordinatensystems sind dabei grundsätzlich parallel zur aktuellen Fahrtrichtung. Aufgrund des Radius der optimalen Kreisbahn wird dann eine Rotationsgeschwindigkeit für den AMR abgeleitet.
Da hier die motorische Kontrolle ohne explizite Trajektorienberechnung auskommen soll, wird nur eine wesentliche Idee dieses Ansatzes aufgegriffen. Das wesentliche Merkmal einer solchen Trajektorie ist die Richtung eines Freiraums in der Umgebung. Wird aus den Ultraschallmeßwerten eine Freiraumrichtung in der Umgebung abgeleitet, so wird das obige Problem der symmetrischen Zustände dadurch gelöst, daß die Regler die Freiraumrichtung als Eingabemerkmal erhalten.
Zur Extraktion eines Freiraums in der Umgebung wird im folgenden ein ähnlicher Ansatz wie in [BB+95] beschritten. Da aus den aktuellen Meßwerten von nur acht Sensoren kaum ein verläßlicher Freiraum in der Umgebung erhalten wird, werden die gemessenen Sensorwerte innerhalb eines rechteckigen Bereiches um das Fahrzeug in einer Liste abgespeichert. Durch die so gespeicherten Meßwerte wird die Umgebung des AMR's grob abgebildet. Damit ist es dann möglich, einen Freiraum in der Umgebung zu erkennen und dieses Merkmal an die Hindernisvermeidung zu übergeben.
Durch die Speicherung der Sensormeßwerte in einem Kurzzeitgedächtnis ist das Verhalten des Roboters im strengen Sinne nicht mehr reaktiv. Allerdings dient die Speicherung der Meßwerte nur für eine Erweiterung der Sensorik. Die Steuerungseinheit selbst, die hier aus dem FDCS-Regler besteht, kommt nach wie vor ohne ein Kurzzeitgedächtnis aus. Daher kann diese Vorgehensweise in einem weiteren Sinne als reaktiv betrachtet werden.

5.2.1 Speicherung der Sensormeßwerte

In guter Näherung an die Charakteristik eines Ultraschallsensors wird eine Ultraschallkeule durch einen Kegel approximiert. Wird eine Entfernung zu einem Hindernis gemessen, so gibt diese die Höhe des Kegels an. Unter Ausnutzung des bekannten charakteristischen Öffnungswinkel[2] α des Ultraschallkegels kann die Ausdehnung der Kegelgrundfläche bestimmt werden. In der Grundfläche dieses Kegels muß sich das erkannte Hindernis befinden. Da alle Ultraschallsensoren in einer Ebene auf dem Fahrzeug angebracht sind, kann

[2] Die US-Sensoren des *LABMATE* haben einen charakteristischen Öffnungswinkel von ca. $\alpha \approx 18°$.

die Geometrie der Ultraschallkegel auf zwei Dimensionen reduziert werden. In der Ebene der Ultraschallsensoren modelliert man die Kegel als gleichschenklige Dreiecke und die Grundfläche der Kegel geht in die Grundseite des Dreiecks über.

Aus der Position und der Orientierung eines Sensors im egozentrischen Roboterkoordinatensystem erhält man die Koordinaten der Eckpunkte der Grundseite des gleichschenkligen Dreiecks. Damit ergibt sich zu jeder Ultraschallmessung eine Strecke im egozentrischen Koordinatensystem, die für ein Hindernis steht. Diese Strecken werden in einer *Hindernisliste* abgespeichert. Bei jeder durchgeführten Bewegung des AMR's müssen dabei die Koordinaten von zuvor gespeicherten Hindernisstrecken an die Bewegung angepaßt werden, so daß die relative Lage der Hindernisstrecken zum Fahrzeug immer richtig bleibt.

Ebenso wie die Grundseite eines gleichschenkligen Dreiecks eines Sensorkegels ein Hindernis in der Umgebung angibt, stellt das innere dieses Dreiecks einen hindernisfreien Raum dar. Alte Hindernisstrecken, die ganz oder auch nur teilweise innerhalb eines solchen Dreiecks liegen, werden daher aus der Liste gelöscht oder zumindest verkürzt. Da nur die Umgebung innerhalb eines begrenzten Bereiches um das Fahrzeug für eine Freiraumdetektion von belang ist, können Hindernisstrecken, die aus diesem Bereich herausfallen, ebenfalls aus der Hindernisliste gelöscht werden. Somit gibt die Liste der Hindernisstrecken eine Repräsentation einer lokalen Fahrzeugumgebung an. Aus dieser Liste können alle Merkmale abgeleitet werden, die einem FDCS-basierten Hindernisvermeidungsmodul als Eingabe dienen sollen.

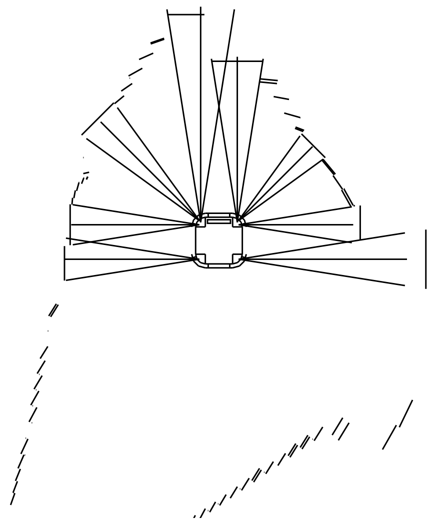


Abbildung 5.6: Beispiel einer Umgebungsrepräsentation mit der Hindernisliste

Die Abbildung 5.6 zeigt ein Beispiel der in einer Hindernisliste aufgezeichneten Hindernisstrecken. Für den praktischen Einsatz zur Merkmalsextraktion reicht eine Umgebungsbegrenzung, die sich an der maximal meßbaren Entfernung der Ultraschallsensoren orientiert.

5.2.2 Merkmalsextraktion mit Hilfe der Hindernisliste

Im Anhang C wird dargestellt, wie die Hindernisliste erstellt und weitere Merkmale abgeleitet werden können. Mögliche extrahierte Merkmale sind beispielsweise:

- Die Richtung eines Freiraums, die am wenigsten von der Richtung eines Zielpunktes abweicht (Zielverfolgung).
- Die Richtung eines Freiraums, die am wenigsten von der Geradeausfahrt abweicht (freies Herumfahren).
- Der Abstand zu frontalen Hindernissen.
- Der Abstand zu seitlichen Hindernissen.
- Die Orientierung von frontalen Hindernissen.
- Die Orientierung von seitlichen Hindernissen (z. B. Wände).

Obwohl insbesondere die letzten vier Merkmale auch ohne eine Hindernisliste direkt aus den Sensorwerten zu gewinnen sind, hat die Umgebungsrepräsentation durch die Hindernisliste den Vorteil, daß sie gegenüber der direkten Sensordateninterpretation eine viel höhere Auflösung der Umgebung liefert. Damit sind sogar Sensorkonfigurationen mit sich überschneidenden Ultraschallkeulen möglich. Die Auflösung der einzelnen Sensoren wird dadurch stark erhöht.

5.3 FDCS-basierte Hindernisvermeidung mit Sensordatenvorverarbeitung

Mit der im letzten Abschnitt behandelten Vorverarbeitung von Ultraschallsensordaten und anschließender Merkmalsextraktion wird jetzt das zuvor angegebene Fuzzy-basierte Hindernisvermeidungsmodul verbessert. Damit berechnet das System auch in symmetrischen Zuständen eine sinnvolle Motorstellgröße.
Im folgenden wird neben den acht Ultraschallmeßwerten als zusätzliches Eingabemerkmal die Richtung eines erkannten Freiraums verwendet, der zu der jeweils angestrebten Aufgabe am besten paßt. Wie in den bisherigen Hindernisvermeidungsansätzen sollen hier zwei Aufgaben verfolgt werden, nämlich die Orientierung an Wänden, um eine Exploration der Umgebung zu ermöglichen und die Verfolgung von Zielpunkten, um innerhalb eines kompletten Navigationssystems (vgl. Kapitel 3) Verwendung zu finden.
Diese beiden Aufgaben wurden bisher durch zwei getrennte Verhaltensebenen realisiert, die ihrerseits über den hindernisvermeidenden Verhaltensebenen angeordnet waren. Mit Hilfe der Freiraumrichtungen ist diese Trennung nicht mehr notwendig, da es ausreicht, wenn die auszuführende Aufgabe die Wahl der Freiraumrichtung bestimmt.

5.3.1 Architektur der FDCS-basierten Hindernisvermeidung mit Freiraumextraktion

Die Merkmalsextraktion mit Hilfe der Hindernisliste ist in der Lage, eine Menge von Freiraumrichtungen zu berechnen, die in solche Regionen der unmittelbaren Umgebung des AMR's zeigen, in die das Fahrzeug ausgehend vom Ursprung des egozentrischen Roboterkoordinatensystems fahren könnte.
Da natürlich nur eine Freiraumrichtung vom AMR verfolgt werden kann, ist eine Richtung aus den Freiraumrichtungen auszuwählen.

Zielverfolgung: Soll der AMR einen bestimmten Zielpunkt erreichen, so wird die Freiraumrichtung zur Verfolgung ausgewählt, die am wenigsten von der Richtung des Zielpunktes abweicht. Damit kann eine Zielverfolgung in Umgebungen ohne Sackgassen vorgenommen werden.

Exploration: Soll der AMR dagegen einfach nur seine Umgebung erkunden, so genügt es, dem Freiraum zu folgen, der die geringste Abweichung von der Fahrtrichtung hat. Fährt der AMR in Richtung einer Wand, so wird er vor ihr ausweichen, bis das Fahrzeug sich parallel zur Wand befindet. Wird dem Freiraum in Fahrtrichtung gefolgt, so ergibt sich ein Verhalten, das dem Wandfolgen entspricht.

Es ist also nicht mehr nötig, die aufgabenorientierte Verhaltensebene in zwei verschiedene Ebenen aufzuteilen. Stattdessen genügt ein Verhalten, das nur noch eine Verfolgung der Freiraumrichtung anstrebt. Die unterschiedlichen Aufgabenstellungen der Exploration und Zielverfolgung fließen nur noch in die Merkmalsextraktion ein.

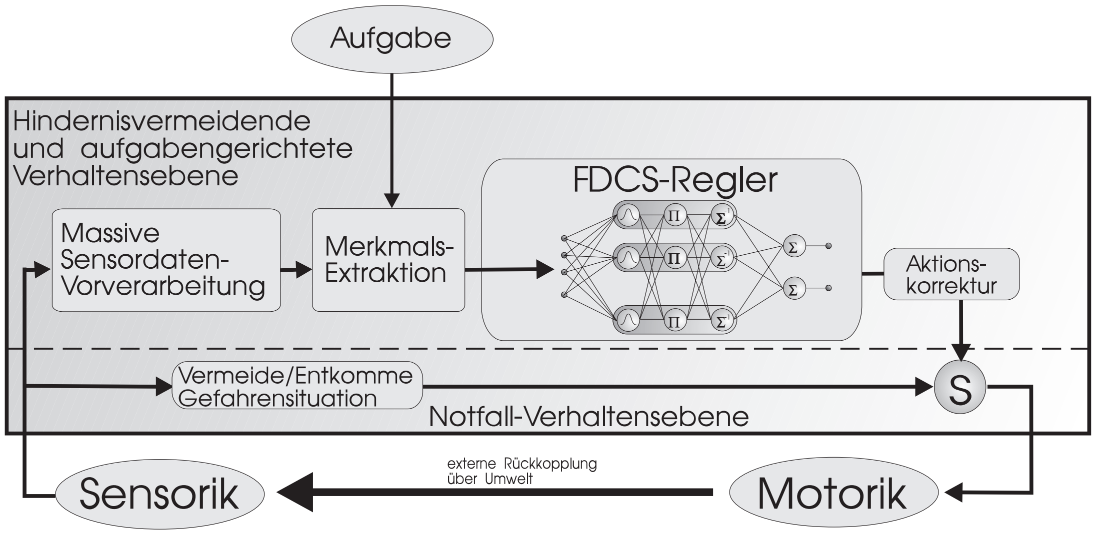


Abbildung 5.7: FDCS-basierte Hindernisvermeidung mit Sensordatenvorverarbeitung

Im ersten FDCS-Ansatz wurden bereits mehrere Teilverhalten innerhalb eines einzigen FDCS-Reglers durch bestimmte Neuronengruppen realisiert. Diese Vereinfachung wird in

diesem Ansatz noch erweitert, indem nur noch ein einziger FDCS-Regler verwendet wird. Die Architektur der Hindernisvermeidung stellt sich somit wie in Abbildung 5.7 dar.
Durch die Verschmelzung von aufgabenorientierter und hindernisvermeidender Verhaltensebene entfällt die schmerzsignalgetriebene Verhaltensauswahl. Die unterste Verhaltensauswahl durch die Notfallverhaltensebene bleibt dagegen bestehen. Ebenso wird hinter den FDCS-Regler wieder eine Aktionskorrektur geschaltet.

5.3.2 Regelbasis für die FDCS-Regler

Die Tabellen zeigen die Regelbasis des FDCS-Reglers für die Hindernisvermeidung mit Sensordatenvorverarbeitung. Als Merkmale werden alle acht Sensormeßwerte $s_0, \cdots, s_7$ sowie die Richtung α_{best} der für die jeweilige Aufgabe am besten geeigneten Freiraumrichtung verwendet. Neben den Regeln der folgenden Tabellen wurden außerdem die Regeln aus Tabelle 5.3 mit in die Regelbasis aufgenommen. Die Regeln in Tabelle 5.8 wurden durch exakte Neuronen realisiert.

v/ω	α_{best}					
	-150	-100	-50	50	100	150
$s_1 \approx 100:$	0/-30	0/-30	0/-30	0/30	0/30	0/30
$s_2 \approx 100:$	0/-30	0/-30	0/-30	0/30	0/30	0/30

Tabelle 5.5: Regelbasis zur Behandlung von Gefahrensituationen

if $(s_1 \approx 3000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 600) \wedge (s_3 \approx 600)$ **then** $v \approx 600$
if $(s_1 \approx 3000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 1300) \wedge (s_3 \approx 1300)$ **then** $v \approx 750$
if $(s_1 \approx 3000) \wedge (s_2 \approx 3000) \wedge (s_0 \approx 3000) \wedge (s_3 \approx 3000)$ **then** $(v = 1000) \wedge (\omega = 0)$
if $(s_1 \approx 1500) \wedge (s_2 \approx 1500)$ **then** $v \approx 400$
if $(s_1 \approx 1000) \wedge (s_2 \approx 1000)$ **then** $v \approx 300$
if $(s_1 \approx 500) \wedge (s_2 \approx 500)$ **then** $v \approx 150$

Tabelle 5.6: Regelbasis zur Kontrolle der Fahrtgeschwindigkeit

5.3.3 Simulationen

Experimente im Simulator mit der FDCS-basierten Hindernisvermeidung mit massiver Sensordatenvorverarbeitung zeigen, daß einerseits die Vorteile der Fuzzy-Steuerung des ersten FDCS-Ansatzes erhalten bleiben und auch das Problem der symmetrischen Zustände durch die Hinzunahme einer Freiraumrichtung zu den Eingabegrößen gelöst wird.
Die Abbildung 5.8 b) zeigt ein erfolgreiches Ausweichmanöver eines AMR's in einer Situation, in welcher der erste Fuzzy-Ansatz gescheitert ist. In Abbildung 5.8 a) wird die Hindernisliste und der extrahierte Freiraumvektor dargestellt. Damit gelingt es, das Fahrzeug aus der Ecke in der Umgebung herauszuführen.

$\alpha_{best} \approx 0$:					
ω	s_0				
s_3	0	300	700	1200	1800
0		30	30	30	30
300	-30		30	30	30
700	-30	-30		30	30
1200	-30	-30	-30		30
1800	-30	-30	-30	-30	

$\alpha_{best} \approx 0$:					
ω	s_1				
s_2	0	300	700	1200	1800
0		30	30	30	30
300	-30		30	30	30
700	-30	-30		30	30
1200	-30	-30	-30		30
1800	-30	-30	-30	-30	

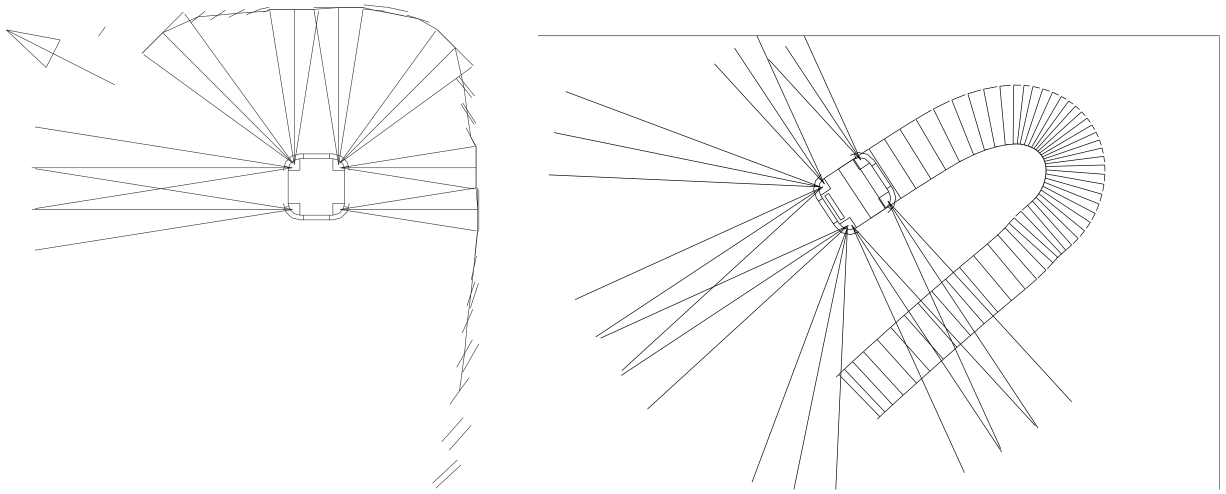
ω	α_{best}					
$s_0 = s_3$	-150	-100	-50	50	100	150
0	-30	-30	-30	30	30	30
300	-30	-30	-30	30	30	30
700	-30	-30	-30	30	30	30
1200	-30	-30	-30	30	30	30
1800	-30	-30	-30	30	30	30

ω	α_{best}					
$s_1 = s_2$	-150	-100	-50	50	100	150
0	-30	-30	-30	30	30	30
300	-30	-30	-30	30	30	30
700	-30	-30	-30	30	30	30
1200	-30	-30	-30	30	30	30
1800	-30	-30	-30	30	30	30

Tabelle 5.7: Regelbasis zur Vermeidung frontaler Kollisionen

ω	α_{best}					
	-150	-100	-50	50	100	150
$s_1 \approx s_2 \approx 3000$:	-30	-30	-30	30	30	30

Tabelle 5.8: Regelbasis zur Freiraumverfolgung



a) b)

Abbildung 5.8: a) Extrahierter Freiraum b) Ausweichmanöver eines AMR's

Kapitel 6

Adaptive Hindernisvermeidung

In diesem Kapitel wird die FDCS-basierte Hindernisvermeidung um die Lernfähigkeit ergänzt. Der AMR soll selbständig einen Regler erlernen, der das Fahrzeug durch eine unbekannte Umgebung kollisionsfrei fahren läßt (vgl. [DC94], [MT92], [Thr95], [Dig93], [PM92] und [RN94]).

Zunächst stellt sich jedoch die Frage, weshalb überhaupt lernfähige Systeme benötigt werden, wenn die Aufgabe auch mit statischen Methoden lösbar ist. Im vorherigen Kapitel wurde ein derartiges statisches System auf der Basis fester Fuzzy-Regeln entworfen. Obwohl die Aufgabe der Hindernisvermeidung auf den ersten Blick nicht so schwer erscheint, macht es dennoch viel Mühe, diese durch einen Fuzzy-Regler zu implementieren. Dabei ist anzunehmen, daß der entworfene Regler noch kein optimales Verhalten darstellt. Eine Optimierung eines vorgegebenen Fuzzy-Reglers durch ein selbständig lernendes System stellt eine wünschenswerte Ergänzung dar.

Des weiteren ändern sich während der Lebensdauer des Roboters ständig die Randbedingungen unter denen ein solcher Regler entworfen wird. Zum Beispiel durch Verschleiß der Antriebstechnik oder durch Änderung des Untergrundes können sich Randbedingungen gegenüber den Bedingungen zur Zeit der Entwurfsphase eines statischen Reglers ändern, die die Qualität der Hindernisvermeidung mindern. Ist dagegen das System lernfähig und während des Einsatzes ständig auf der Suche nach einem optimalen Verhalten, so paßt sich der adaptive Regler ändernden Randbedingungen an.

Prinzipiell wird der hier dargestellte Ansatz zum Erlernen eines Verhaltens zur Hindernisvermeidung sowohl in der Lage sein, durch Fuzzy-Regeln vorkodiertes Wissen zu verfeinern, als auch völlig selbständig ohne jegliches Vorwissen einen solchen Regler zu erlernen. Da die letztere Aufgabe zweifellos die schwierigere ist und die Verfeinerung von Fuzzy-Vorwissen als Spezialfall enthält, wird sich im folgenden auf das Erlernen eines Hindernisvermeidungsverhaltens ohne Vorwissen beschränkt. Dennoch sollte der Vorteil, eine a priori gegebene Wissensbasis adaptiv verbessern zu können, nicht unterschätzt werden. In vielen technischen Anwendungen liegt bereits ein großes Expertenwissen vor, das auch für lernfähige Regler als Vorwissen genutzt werden sollte.

6.1 Reinforcement-Lernen

Obwohl es prinzipiell möglich ist, das Hindernisvermeidungsverhalten durch überwachtes Lernen mit einem Lehrer zu erwerben, wird hier eine weitaus schwierigere überwachte Lernaufgabe behandelt, bei der zu einer Eingabe eines neuronalen Netzwerkes keine korrekte Ausgabe bekannt ist. Vielmehr wird nur ein Kritiksignal aus der Umwelt des Systems abgeleitet. Diese gibt im Extremfall nur darüber Auskunft, ob die berechnete Ausgabe *gut* oder *schlecht* ist. Eine solche Situation ist auch beim Erlernen eines hindernisvermeidenden Verhaltens eines AMR's gegeben. Für den AMR ist zu einer gegebenen sensorischen Wahrnehmung eben keine korrekte Aktion a priori bekannt, sondern es wird nach einer ausgeführten Aktion nur festgestellt, ob diese gut oder schlecht war und eventuell auch wie gut diese Aktion war. Das Lernen aufgrund von Kritiksignalen wird als *Verstärkungslernen* oder *Reinforcement-Lernen* bezeichnet und gehört in die Klasse der überwachten Lernverfahren.

Beim Einsatz des Reinforcement-Lernens zum Erwerb von Roboterverhalten wird grundsätzlich nach einer Abbildung gesucht, die für den Roboter (*Agenten*) optimale *Aktionen* eines Aktionsraumes $\mathcal{A}$ angibt. Die Aktionsauswahl ist von der Situation (*Zustände eines Zustandsraumes* $\mathcal{S}$) abhängig, in der sich der Agent befindet. Eine solche zu erlernende Abbildung

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

wird als *Politik* bezeichnet. Sie bestimmt neben den äußeren Umwelteinflüssen während der Lebensdauer eines Agenten eine Zustandsfolge der Form

$$s_0 \xrightarrow[r_0]{\pi(s_0)} s_1 \xrightarrow[r_1]{\pi(s_1)} s_2 \xrightarrow[r_2]{\pi(s_2)} s_3 \xrightarrow[r_3]{\pi(s_3)} s_4 \longrightarrow \quad \cdots \quad \longrightarrow s_j \xrightarrow[r_j]{\pi(s_j)} s_{j+1} \longrightarrow \quad \cdots$$

Dabei ist $s_j \in \mathcal{S}$, $j \in I\!N_0$ ein Zustand und $\pi(s_j)$ eine durch die Politik berechnete und ausgeführte Aktion, die den Agenten durch Wechselwirkung mit der Umwelt in den Folgezustand s_{j+1} versetzt. Die Wechselwirkung mit der Umwelt wird meistens als Markov-Prozeß modelliert, der von Übergangswahrscheinlichkeiten $p(s, s', a)$ abhängt. Dieser Wert gibt die Wahrscheinlichkeit an, durch Ausführung der Aktion a vom Zustand s in den Folgezustand s' zu gelangen.

Für solche Zustandsübergänge erhält der Agent eine *direkte Belohnung* $r_j \in I\!R$, die auch als Kritik, Verstärkungs- oder Reinforcement-Signal bezeichnet wird.

Das Ziel besteht darin, eine Politik π^* zu finden, die die zu erwartenden Belohnungen maximiert.

Ein Überblick über gängige Reinforcement-Lernansätze wird von Richard Sutton in [Sut90] angegeben.

6.1.1 Die policy-only-Architektur

Die einfachste Form des Reinforcement-Lernen bildet die sogenannte *policy-only*-Architektur. In dieser Architektur ist die Politik die einzige modifizierbare Datenstruktur.

Die Politik wird hier ausschließlich durch die Verstärkungssignale trainiert. Ferner wird davon ausgegangen, daß diese Signale um den Basiswert 0 verteilt sind, und daß Belohnungen positive und Bestrafungen negative Verstärkungssignale liefern.
Oft sind die Verstärkungssignale allerdings nicht um den Basiswert 0 verteilt, sondern um einen von 0 verschiedenen Basiswert, der zusätzlich vom Zustand des Systems abhängt. Dann ist ein Verstärkungssignal in einigen Zuständen als Belohnung aufzufassen, während das gleiche Verstärkungssignal in anderen Zuständen als Bestrafung anzusehen ist.

6.1.2 Die reinforcement-comparison-Architektur

Eine möglicher Ausweg aus dieser Problematik stellt die *reinforcement-comparison*-Architektur dar, die nicht direkt das Belohnungssignal r_j für die Adaption der Politik heranzieht, sondern ein zweites Verstärkungssignal r_j^s (secondary reinforcement) einführt. Dieses wird durch einen Vergleich der direkten Belohnung mit einem *Basiswert* b (baseline-value) erhalten.

$$r_j^s = r_j - b. \tag{6.1}$$

Dieser Basiswert gibt an, um welchen Wert die Belohnungen im mittel verteilt sind. Da die Verstärkungssignale oft vom Zustand abhängig um unterschiedliche Basiswerte verteilt sind, werden die Basiswerte vom jeweiligen Zustand s_j abhängig gemacht. Der Basiswert wird daher ebenfalls adaptiv gehalten, wobei in Abhängigkeit des Zustandes die mittlere zu erwartende direkte Belohnung erlernt und als Basiswert verwendet wird.

6.1.3 AHC-Lernen

Da sich Aktionen sowohl auf die direkte Belohnung aber auch auf den Folgezustand auswirken, haben sie auch Einfluß auf die zu erwartenden Belohnungen der folgenden Zustandsübergänge. Der Agent kann durch eine Aktion in einen Zustand gebracht werden, von dem aus langfristig nur eine geringe Belohnung zu erwarten ist, selbst wenn die direkte Belohnung der ausgeführten Aktion einen größtmöglichen direkten Wert erzielt. Es ist für zahlreiche Anwendungen daher nicht ausreichend, eine Politik zu suchen, die die direkten Belohnungen maximiert. Oftmals ist eine Politik zu suchen, die die für die Zukunft zu erwartende akkumulierte Belohnung maximiert. Das Problem, daß direkte Belohnungen nicht nur den momentanen Zustandsübergang, sondern auch frühere Zustandsübergänge bewerten, wird als das Problem der zeitlichen Zuordnung von Bewertungen (*temporal credit assignment problem*) bezeichnet. Die Architektur der *adaptiven heuristischen Kritik* (*adaptive heuristic critic (AHC)*) löst dieses Problem [BSA83].
Statt nur die direkten Belohnungen zu maximieren, wird beim AHC-Lernen eine Politik π^* gesucht, die

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s \wedge \forall t \in \mathbb{N}_0 : s_t \underset{r_t}{\overset{\pi(s_t)}{\longrightarrow}} s_{t+1}\right] \tag{6.2}$$

unter allen Politiken π für alle Zustände $s \in \mathcal{S}$ maximiert. Eine solche optimale Politik π^* existiert immer und kann bei bekannten Übergangswahrscheinlichkeiten $p(s, s', a)$ mit Methoden der *dynamischen Programmierung* [BB93] bestimmt werden.

Die Abbildung $V_\pi : \mathcal{S} \to I\!R$ wird hier als *Ertragsfunktion* bezeichnet und ist jeweils von der zugrundeliegenden Politik abhängig. Die Konstante $\gamma \in]0,1[$ heißt *Diskontierungsrate* und sichert die Konvergenz[1] von (6.2). Ferner wird durch γ festgelegt, wie weit die Zukunft durch V_π berücksichtigt werden soll. $\gamma \approx 1$ führt zu einer Betrachtung großer Zeiträume in die Zukunft, während $\gamma \approx 0$ die Zukunft weitgehend unberücksichtigt läßt und die Gegenwart dafür stärker gewichtet.
Beim AHC-Lernen lernt man neben der Politik π eine Schätzung V für die Ertragsfunktion V_π mit der Aufdatierungsregel

$$\Delta V(s_j) = \beta \cdot (r_j + \gamma V(s_{j+1}) - V(s_j)) . \tag{6.3}$$

Sagt die geschätzte Ertragsfunktion V die akkumulierte Belohnung korrekt voraus, so gilt:

$$V(s_j) = \sum_{t=0}^{\infty} \gamma^t r_{j+t} = r_j + \gamma \cdot \sum_{t=0}^{\infty} \gamma^t r_{(j+1)+t} = r_j + \gamma \cdot V(s_{j+1}). \tag{6.4}$$

Also folgt

$$r_j = V(s_j) - \gamma \cdot V(s_{j+1}) \tag{6.5}$$

und somit gibt

$$b = V(s_j) - \gamma \cdot V(s_{j+1}) \tag{6.6}$$

einen Basiswert an, der wie bei reinforcement-comparison-Architekturen zur Adaption der Politik verwendet wird.

6.1.4 Q-Lernen

Ein weiteres Reinforcement-Lernverfahren stellt das *Q-Lernen* dar (vgl. [Wat89], [Wat92], [Lin93]). Hier wird die Ertragsfunktion um die Abhängigkeit von Aktionen ergänzt. Die daraus resultierende Abbildung

$$Q_\pi(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s \underset{r_0}{\overset{a}{\longrightarrow}} s_1 \wedge \forall t \in I\!N : s_t \underset{r_t}{\overset{\pi(s_t)}{\longrightarrow}} s_{t+1}\right] \tag{6.7}$$

wird als *Q-Funktion* bezeichnet und gibt zu einem Zustand $s \in \mathcal{S}$ und einer Aktion $a \in \mathcal{A}$ den zu erwartenden Ertrag der akkumulierten Belohnungen an, wenn im Zustand s die Aktion a ausgeführt wird und für die folgenden Zustände die Politik π die ausgeführten Aktionen berechnet. Da die optimale Politik π^* die Ertragsfunktion maximiert, gilt im Falle eines endlichen Aktionsraumes

$$V_{\pi^*}(s) = \max_{a \in \mathcal{A}} Q_{\pi^*}(s,a), \quad s \in \mathcal{S}. \tag{6.8}$$

Ähnlich wie beim AHC-Lernen wird auch beim Q-Lernen die Funktion Q_π durch eine Funktion Q geschätzt, die durch die Aufdatierungsregel

$$\Delta Q(s_j, a_j) = \beta_j \cdot \Big(r_j + \gamma \max_{a \in \mathcal{A}} Q(s_{j+1}, a) - Q(s_j, a_j)\Big) \tag{6.9}$$

[1] Gilt für alle $i \in I\!N$ $|r_i| < c$ für eine Konstante $c \geq 0$, so folgt die Konvergenz direkt über den Konvergenzsatz von Cauchy-Hadamard für Potenzreihen.

erlernt wird.
Falls Q die Q-Funktion der optimalen Politik vollständig erlernt hat, so gilt

$$Q(s,a) = Q_{\pi^*}(s,a) \text{ für alle } s \in \mathcal{S}, a \in \mathcal{A}. \tag{6.10}$$

Umgekehrt gilt dann für die optimale Politik

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s,a), \tag{6.11}$$

so daß bereits durch die erlernte Q-Funktion die Politik implizit festgelegt wird und nicht wie beim AHC-Lernen explizit gelernt werden muß. Die durch die Q-Funktion implizit definierte Politik ist eine *gierige Politik* (greedy policy).
In [Wat92] konnte das folgende theoretische Resultat gezeigt werden. Ist der zugrundeliegende stochastische Prozeß Markovsch und ist $\mathcal{S} \times \mathcal{A}$ endlich, so daß Q in einer Tabelle gespeichert werden kann, und wird die Q-Funktion für jedes Zustands-Aktions-Paar unendlich oft adaptiert, so konvergiert die Q-Funktion und damit auch die durch sie definierte Politik gegen eine optimale Politik, falls die Q-Funktion in jedem Zeitschritt mit (6.9) adaptiert wird, und dabei $\beta_j > 0$ eine mit $j \to \infty$ gegen Null konvergierende Lernkonstante ist.
Um in der Praxis möglichst viele Zustands-Aktions-Paare zu treffen, werden von der Politik vorgeschlagene Aktionen nicht direkt an die Motorik des Roboters weitergeleitet. Die Aktionen werden vielmehr zufällig ausgewählt, wobei Aktionen mit höherem Q-Wert wahrscheinlicher sind als solche mit kleinerem Q-Wert. Für diese Exploration wird vielfach die Boltzmann-Verteilung verwendet, die mit fortschreitender Zeit T die durch die Q-Funktion festgelegte Politik immer wahrscheinlicher werden läßt.

$$P(a|s_j) = \frac{\exp(Q(s_j,a)/T)}{\sum_{\bar{a} \in \mathcal{A}} \exp(Q(s_j,\bar{a})/T)} \tag{6.12}$$

gibt dazu die Wahrscheinlichkeit zum Zeitpunkt T an, daß die Aktion a im Zustand s_j ausgeführt wird.
Ein Vorteil des Q-Lernens gegenüber dem AHC-Lernen ist der, daß unter bestimmten Voraussetzungen für das Q-Lernen die Konvergenz zugesichert werden kann, was für das AHC-Lernen bisher nicht bekannt ist. Ferner wird durch die Q-Funktion direkt eine Politik angegeben, während beim AHC-Lernen die Politik zusätzlich erlernt werden muß.
Die theoretischen Resultate hinsichtlich der Konvergenz des Q-Lernens sind allerdings nur eingeschränkt aussagefähig, da für viele praktische Anwendungen der stochastische Prozeß nicht Markovsch ist. Ferner ist es in den seltensten Fällen zu gewährleisten, daß alle möglichen Zustands-Aktions-Paare während der Lebensdauer des Agenten unendlich oft vorkommen. Letzteres ist besonders dann nicht zu erreichen, wenn bestimmte Zustände die Sicherheit des Agenten gefährden können. Solche Zustände sind auch während der Lernphase unbedingt zu vermeiden. Des weiteren ist die Forderung nach der Konvergenz der Lernrate ($\beta_j \to 0$) unerwünscht, da dadurch die Elastizität der erlernten Politik aufgegeben wird. Diese ist jedoch für den Einsatz des Roboters in einer sich ändernden Umwelt unumgänglich.

6.1.5 Stochastisches reellwertiges Reinforcement-Lernen

Ein weiterer Nachteil des Q-Lernens besteht in der Forderung nach einer endlichen[2] Aktionsmenge. Da der im letzten Abschnitt entworfene FDCS-Regler zur Hindernisvermeidung keine endliche Aktionsmenge, sondern einen kontinuierlichen Aktionsraum besitzt, kann die ursprüngliche Version des Q-Lernens nicht verwendet werden.
Reinforcement-Lernansätze, die auch mit kontinuierlichen Aktionsräumen zurecht kommen, wurden von [Wil92] und [GF94] vorgeschlagen. Diese Reinforcement-Ansätze werden auch als *stochastische reellwertige Reinforcement-Lernansätze* (stochastic real-valued-reinforcement) bezeichnet. Hier dürfen der Zustands- und Aktionsraum als überabzählbare Mengen $\mathcal{S} \subseteq I\!R^n$ und $\mathcal{A} \subseteq I\!R^m$, $n, m \in I\!N$ angenommen werden.
In diesem Ansatz ist die Politik anders als bei den vorherigen Reinforcement-Ansätzen keine deterministische Funktion mehr, sondern eine Abbildung, die in Abhängigkeit des Zustandes die Parameter einer Zufallsverteilung angibt. Die auszuführenden Aktionen der Politik ergeben sich damit als Zufallsexperiment gemäß der durch die Politik gegebenen Verteilung. Dazu sei $\mathcal{P} \subseteq I\!R^k$, $k \in I\!N$ ein *Parameterraum* und

$$\pi : \mathcal{S} \to \mathcal{P}$$

die Politik. Zu einem Zustand $s \in \mathcal{S}$ liefert $a(s) \in \mathcal{A}$ eine zufällige Aktion, deren Verteilung durch die von $\pi(s)$ berechneten Parameter festgelegt ist.
Ist der Aktionsraum $\mathcal{A}$ diskret, so wird die Verteilung mit Hilfe der Wahrscheinlichkeitsfunktion

$$g_i(x, \pi(s)) = P(a(s)_i = x \mid \pi(s)), \quad i \in I\!N_{\leq m} \tag{6.13}$$

festgelegt. Ist der Aktionsraum dagegen kontinuierlich, so gibt $g_i(\cdot, \pi(s))$ die Wahrscheinlichkeitsdichte zu $a(s)_i$ in Abhängigkeit der durch $\pi(s)$ berechneten Parameter an.
Williams führt in [Wil92] den Begriff des *REINFORCE*-Algorithmus[3] ein, indem die Gewichte w_k der Politik, die die Parameter der Wahrscheinlichkeitsdichte g_i festlegen, gemäß

$$\Delta w_k = \beta \cdot (r - b) \cdot \frac{\partial}{\partial w_k} (\ln \circ g_i)\,(a(s)_i, \pi(s)), \quad i \in I\!N_{\leq m} \tag{6.14}$$

adaptiert werden. Dabei ist β eine Lernkonstante und b ein Basiswert wie bei der reinforcement-comparison-Architektur. Der Wert von r gibt die aktuelle direkte Belohnung an.
Williams konnte zeigen, daß der mittlere Aufdatierungsvektor im Parameterraum in Richtung des Gradienten des mittleren direkten Belohnungssignals nach dem Parametervektor zeigt, so daß die REINFORCE-Algorithmen im Mittel einem Gradientenverfahren entsprechen und die mittleren direkten Belohnungen maximieren.

$$\nabla_w E\,[r|w] = \beta \cdot E\,[\Delta w|\,w]\,. \tag{6.15}$$

[2] Mit *endlich* ist in der Praxis eine wirklich kleine Zahl von Aktionen zu verstehen. Da man das Maximum der Q-Werte über der Aktionsmenge bestimmen muß und die Q-Funktion entweder als Tabelle oder als neuronales Netz darstellt, wächst der Zeit- und Platzaufwand mindestens linear mit der Anzahl der Aktionen.

[3] REINFORCE ist eine Abkürzung für *REward Increment = Nonnegative Factor* $\times$ *Offset Reinforcement* $\times$ *Characteristic Eligibility*.

Da der Basiswert wie bereits zuvor adaptiv gehalten werden kann, können die REINFORCE-Algorithmen ebenfalls so erweitert werden, daß verzögerte Belohnungen berücksichtigt werden. Während beim Q-Lernen die Exploration explizit durchgeführt werden muß, ist sie im REINFORCE-Algorithmus von vornherein enthalten, da die Aktionen hier zufällig gewählt werden.

6.2 Prädiktion mit TD(λ)-Algorithmen

Sowohl beim Q-Lernen als auch beim AHC-Lernen werden Funktionen erlernt, die eine Prädiktion über die zu erwartende akkumulierte Belohnung angeben. Auch bei den REINFORCE-Algorithmen könnte eine Ertragsfunktion wie beim AHC-Lernen herangezogen werden, um den Basiswert b in (6.14) gemäß (6.6) zu bestimmen.
Um derartige Vorhersagen zu erlernen, bieten sich die Algorithmen der TD(λ)-Familie an [Sut88].
Dazu betrachte man eine Folge $(s_j)_{j\in\mathbb{N}}$ von aufeinanderfolgenden Zuständen und Kosten $(c_j)_{j\in\mathbb{N}}$. Die Kosten c_j fallen beim Zustandsübergang von s_j nach s_{j+1} an. Gesucht wird eine Prädiktion der akkumulierten Kosten

$$R_j = \sum_{k=0}^{\infty} \gamma^k c_{j+k+1}. \tag{6.16}$$

$\gamma \in]0,1[$ ist wieder eine Diskontierungsrate.
Zur Prädiktion wird eine Funktion $P_t : \mathcal{S} \to \mathbb{R}$ mit variablen Parametern $w_t \in \mathbb{R}^N$ zum Zeitpunkt $t \in \mathbb{N}$ verwendet. Um die Prädiktion zu verbessern, führt Sutton eine Aufdatierungsregel ein, die als *TD-Lernen* (temporal differencing) bezeichnet wird:

$$\Delta w_t = \beta \cdot (c_{t+1} + \gamma \cdot P_t(s_{t+1}) - P_t(s_t)) \cdot \sum_{k=1}^{t} \lambda^{t-k} \nabla_{w_k} P_k(s_k). \tag{6.17}$$

Δw_t ist die Aufdatierung des Parametervektors zum Zeitpunkt t. Statt die Gradientensumme explizit zu berechnen, bestimmt man sie iterativ durch

$$\begin{aligned} e_0 &= 0 \\ e_{t+1} &= \lambda \cdot e_t + \nabla_{w_{t+1}} P_{t+1}(s_{t+1}). \end{aligned} \tag{6.18}$$

Womit man die endgültige Aufdatierungsregel des TD-Lernens erhält:

$$\Delta w_t = \beta \cdot (c_{t+1} + \gamma \cdot P_{t+1}(s_{t+1}) - P_t(s_t)) \cdot e_t. \tag{6.19}$$

Dabei gibt $\beta > 0$ wieder eine Lernrate an. Je kleiner der Parameter $\lambda \in [0,1]$ ist, um so geringer werden Prädiktionen korrigiert, die zeitlich weiter zurückliegen. Der Wert e_t wird als *Spur* bezeichnet. Für $\lambda = 0$ geht die Spur in den einfachen Gradienten zum Zeitpunkt t über. Den Term $c_{t+1} + \gamma \cdot P_{t+1}(s_{t+1}) - P_t(s_t)$ bezeichnet man als *TD-Fehler*.
Eine Anwendung des TD-Lernens auf die Q-Funktion des Q-Lernens führt zu Erweiterungen des Q-Lernens die in [RN94] beschrieben sind. Eine Anwendung des TD-Lernens in Zusammenhang mit REINFORCE-Algorithmen findet sich bei [SE94] und [PM92].

6.3 Reinforcement-Lernen mit FDCS-Netzwerken

Nachdem einige gängige Verfahren zum Reinforcement-Lernen erörtert wurden, wird daraus mit DCS- bzw. FDCS-Netzwerken ein funktionsfähiges System zur Lösung der Reinforcement-Lernaufgabe angegeben.
In Anlehnung an die kontinuierlichen Fuzzy-Regler, die in dem vorherigen Abschnitt besprochen wurden, werden hier nur reellwertige Reinforcement-Ansätze mit DCS-Netzwerken entwickelt. Für weitere konnektionistische Lösungsansätze zum Reinforcement-Lernen und insbesondere für das Q-Lernen sei auf [And90], [BSA83], [Wat92] und [Lin93] verwiesen.
Im folgenden werden zwei Ansätze untersucht. Der erste Ansatz ist eine direkte Umsetzung der REINFORCE-Algorithmen unter Verwendung des TD-Lernens zur adaptiven Gewinnung eines Basiswertes. Der zweite Ansatz ist eine neue reellwertige Variante des Q-Lernens.

6.3.1 Der FDCS-REINFORCE-Ansatz zum Reinforcement-Lernen

Als erster Ansatz zum stochastischen reellwertigen Reinforcement-Lernen werden die REINFORCE-Algorithmen mit FDCS-Netzwerken realisiert.
Die Grundidee des folgenden FDCS-REINFORCE-Ansatzes besteht daraus, daß durch ein FDCS-Netzwerk in Abhängigkeit des aktuellen Zustandes sowohl die Parameter der Verteilung zur zufälligen Auswahl von Aktionen als auch eine Schätzung der Ertragsfunktion V zur Bestimmung eines Basiswertes berechnet werden.
Dazu betrachte man einen Zustandsraum $\mathcal{S} \subseteq \mathbb{R}^n$, einen Aktionsraum $\mathcal{A} \subseteq \mathbb{R}^m$, einen Parameterraum $\mathcal{P} = \mathbb{R}^{2m}$ und einen Raum $\mathcal{B} = \mathbb{R}$ für Basiswerte. Als Eingaberaum für ein neuronales Netzwerk wird $I = \mathcal{S}$ und als Ausgaberaum $O = \mathcal{P} \times \mathcal{B}$ verwendet. Auf diesen Räumen wird das folgende π-FDCS-Netzwerk $\mathcal{R} = (I, O, U, L, \Pi, \rho, \alpha, r, \delta_{\mathcal{R}})$ betrachtet (vgl. Abbildung 6.1).
Die Aktionen $a(s) \in \mathcal{A}$ werden als normalverteilte Zufallsvariablen realisiert. Die Parameter ihrer Verteilung werden durch die Evaluierungsfunktion $\delta_{\mathcal{R}}$ des Netzwerkes berechnet. Die ersten $2m$ Komponenten der Evaluierungsfunktion geben also die Politik an

$$\pi(s) = (\delta_{\mathcal{R}}(s)_1, \cdots, \delta_{\mathcal{R}}(s)_{2m}). \tag{6.20}$$

Zu einem Zustand $s \in \mathcal{S}$ gibt $\delta_{\mathcal{R}}(s)_{2i-1} = \mu_i(s) \in \mathcal{P}$ den Erwartungswert und $\delta_{\mathcal{R}}(s)_{2i} = \sigma_i(s) \in \mathcal{P}$ die Standardabweichung der Aktion $a(s)_i$ für $i \leq m$ an.
Für die Wahrscheinlichkeitsdichte zur Aktion $a(s)_i$ gilt also:

$$g_i(x, \pi(s)) = \frac{1}{\sqrt{2\pi} \cdot \sigma_i(s)} \cdot \exp\left(-\frac{(x - \mu_i(s))^2}{2\sigma_i(s)^2}\right). \tag{6.21}$$

Neben der Politik erlernt das FDCS-Netzwerk gleichzeitig eine adaptive Kritik nach dem AHC-Lernansatz. Dabei wird durch $\delta_{\mathcal{R}}(s)_{2m+1}$ eine Schätzung der Ertragsfunktion $V(s)$

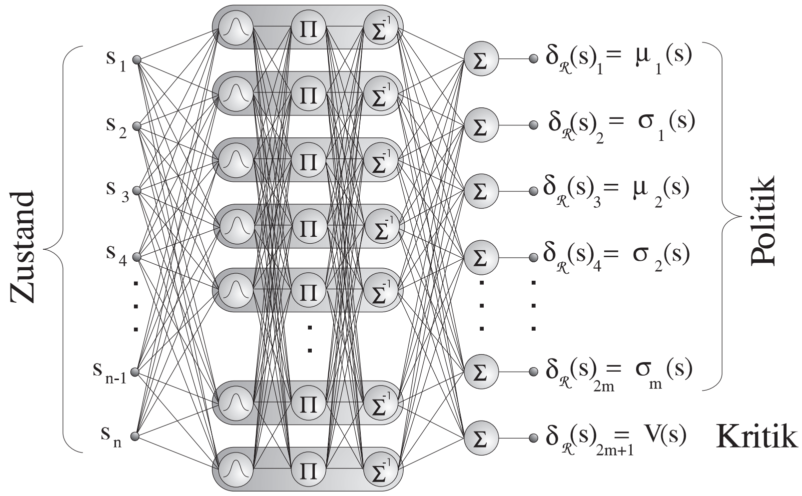


Abbildung 6.1: π-FDCS-REINFORCE-Netzwerk

(vgl. (6.2)) angegeben. Mit dieser Kritik wird ein Basiswert gewonnen, der für eine Berücksichtigung zeitlicher Verzögerungen bei den Belohnungen und unterschiedliche Belohnungsniveaus in Abhängigkeit des Zustandes sorgt.

$$V(s) = \delta_{\mathcal{R}}(s)_{2m+1} \tag{6.22}$$

Adaption des FDCS-REINFORCE-Netzwerkes

Da das FDCS-REINFORCE-Netzwerk sowohl für die Politik als auch für die Kritik dieselben zustandskodierenden Neuronen verwendet, bietet sich für die Adaption der Neuronenzentren die Kohonen-Regel bzw. die fehlermodulierte Kohonen-Regel an. Damit erreicht man, daß die Neuronenzentren den Eingaberaum möglichst vollständig überdecken. Die Schicht der Eingabegewichte des FDCS-Netzwerkes stellt somit einen adaptive Vektorquantisierer dar, der die Eingabevektoren auf die Aktivitäten der Neuronen abbildet.
Wie in FDCS-Netzwerken üblich, muß dazu die laterale Verbindungsstruktur der Neuronen durch Hebb'sches Wettbewerbslernen aufgebaut werden.
Neu ist jetzt die Adaption der Ausgabegewichte der Neuronen. Die Ausgabegewichte der Neuronen nehmen hier die Rolle des Parametervektors w ein, von dem die Politik π abhängt. Dazu betrachte man die Adaption eines Ausgabegewichtes w_u eines Neurons $u \in U$.
Gemäß (6.14) wird ein solches Ausgabegewicht durch

$$\Delta(w_u)_j = \beta \cdot (r - b) \cdot \frac{\partial}{\partial (w_u)_j} (\ln \circ g_i)(a_i, \pi(s)), \quad i \leq m, \quad j \in \{2i, 2i-1\} \tag{6.23}$$

adaptiert, wobei r die letzte direkte Belohnung, b ein Basiswert und a_i die $i$te Komponente der zuletzt ausgeführten Aktion ist.

Für $i \leq m$ und $j \in \{2i, 2i-1\}$ gilt dann

$$\frac{\partial}{\partial(w_u)_j}(\ln \circ g_i)(a_i, \pi(s)) = \frac{1}{g_i(a_i, \pi(s))} \cdot \frac{\partial}{\partial(w_u)_j} g_i(a_i, \pi(s)) \tag{6.24}$$

und somit gilt wegen $\mu_i(s) = \delta_{\mathcal{R}}(s)_{2i-1}$ und $\sigma_i(s) = \delta_{\mathcal{R}}(s)_{2i}$ durch Anwendung der Kettenregel

$$\frac{\partial}{\partial(w_u)_{2i-1}}(\ln \circ g_i)(a_i, \pi(s)) = \frac{1}{g_i(a_i, \pi(s))} \frac{\partial g_i}{\partial \mu_i(s)}(a_i, \pi(s)) \frac{\partial}{\partial(w_u)_{2i-1}} \mu_i(s), \tag{6.25}$$

$$\frac{\partial}{\partial(w_u)_{2i}}(\ln \circ g_i)(a_i, \pi(s)) = \frac{1}{g_i(a_i, \pi(s))} \frac{\partial g_i}{\partial \sigma_i(s)}(a_i, \pi(s)) \frac{\partial}{\partial(w_u)_{2i}} \sigma_i(s). \tag{6.26}$$

Da hier für g_i die Normalverteilung zugrundegelegt wird, erhält man weiterhin

$$\frac{\partial g_i}{\partial \mu_i(s)}(a_i, \pi(s)) = g_i(a_i, \pi(s)) \cdot \frac{a_i - \mu_i(s)}{\sigma_i(s)^2} \tag{6.27}$$

und

$$\frac{\partial g_i}{\partial \sigma_i(s)}(a_i, \pi(s)) = g_i(a_i, \pi(s)) \cdot \frac{(a_i - \mu_i(s))^2 - \sigma_i(s)^2}{\sigma_i(s)^3}. \tag{6.28}$$

Schließlich gilt für $j \leq 2m$

$$\frac{\partial}{\partial(w_u)_j} \delta_{\mathcal{R}}(s) = \begin{cases} \dfrac{\bar{\alpha}_u(s)}{\sum\limits_{v \in \{bmu\} \cup Nh(bmu) \wedge (w_v)_j \neq \perp} \bar{\alpha}_v(s)}, & u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_j \neq \perp \\ 0 & , sonst \end{cases} \tag{6.29}$$

falls die Evaluierungsfunktion wie bei π-FDCS-Netzwerken mit Normalisierung berechnet wird. Dabei gibt bmu wieder das am besten zur Eingabe s passende Neuron an.
Für andere Berechnungsarten (z. B. ohne Normalisierung oder ohne exakte Neuronen) vereinfachen sich die Gradienten entsprechend. Damit kann die Adaptionsregel der Neuronenausgabegewichte nun angegeben werden.

Adaption der Ausgabegewichte der Politik:

Für alle $u \in U$:

Für alle $i \leq m$:

$$\Delta(w_u)_{2i-1} = \beta_\mu \cdot (r_t - b_t) \cdot \frac{a_i^t - \mu_i(s_t)}{\sigma_i(s_t)^2} \cdot \frac{\partial}{\partial(w_u)_{2i-1}} \mu_i(s_t) \tag{6.30}$$

$$\Delta(w_u)_{2i} = \beta_\sigma \cdot (r_t - b_t) \cdot \frac{(a_i^t - \mu_i(s_t))^2 - \sigma_i(s_t)^2}{\sigma_i(s_t)^3} \cdot \frac{\partial}{\partial(w_u)_{2i}} \sigma_i(s_t) \tag{6.31}$$

Dabei ist r_t die zuletzt erhaltene direkte Belohnung und b_t der dazugehörige Basiswert. Der Vektor $a^t = (a_1^t, \cdots, a_m^t)$ stellt die tatsächlich zuletzt ausgeführte Aktion dar. Ferner geben β_μ und β_σ positive Lernkonstanten zur Adaption der Mittelwerte und der Standardabweichungen an.

Die Prädiktion der Ertragsfunktion V wird durch das TD-Lernen adaptiert. In jedem Zeitschritt t wird eine Summe von Gradienten (vgl. (6.18)) für jedes Neuron berechnet. Daher wird das bisherige π-FDCS-Netzwerk um eine Abbildung $\tau : U \rightarrow \mathbb{R}$ erweitert, die jedem Neuron $u \in U$ durch $\tau(u)$ eine solche Gradientensumme zuordnet. Diese Summe wird häufig auch als *Spur* (trace) bezeichnet. Wird diese Spur mit der Schrittzahl der Adaptionsschritte indiziert, so ergibt sich gemäß (6.18) die folgende Adaptionsregel für die Spur:

Adaption der Spur:

$$\text{Für alle } u \in U: \quad \tau(u)_0 = 0 \quad \text{(Initialisierung)} \tag{6.32}$$

$$\text{Für alle } u \in U: \quad \tau(u)_t = \lambda \cdot \tau(u)_{t-1} + \frac{\partial}{\partial (w_u)_{2m+1}} V(s_t) \quad \text{(Iteration)} \tag{6.33}$$

Die Konstante $\lambda \in]0,1[$ bezeichnet eine Diskontierungsrate und s_t den Zustand zum Zeitpunkt t.
Nachdem in jedem Schritt die Spuren aktualisiert wurden, werden schließlich auch die Ausgabegewichte der adaptiven Kritik V aktualisiert.

Adaption der Ausgabegewichte der Kritik V :

Für alle $u \in U$:

$$\Delta(w_u)_{2m+1} = \beta_V \cdot (r_t + \gamma \cdot V(s_{t+1}) - V(s_t)) \cdot \tau(u)_t \tag{6.34}$$

Mit Hilfe der Kritik V kann schließlich auch der Basiswert b_t berechnet werden, der zur Adaption der Ausgabegewichte der Politik nach dem Prinzip der reinforcement-comparison-Architektur benötigt wird (vgl. (6.6), (6.30) und (6.31)).

$$b_t = V(s_t) - \gamma \cdot V(s_{t+1}). \tag{6.35}$$

Entfaltung des FDCS-Netzwerkes

Bevor eine Adaption der Gewichte des FDCS-Netzwerkes vorgenommen wird, müssen zunächst Neuronen in das Netzwerk eingefügt werden. Dieses wird sowohl nach der distanzgetriebenen als auch nach der fehlergetriebenen Einfügestrategie vorgenommen.
Für die fehlergetriebene Einfügestrategie wird ein Fehlermaß für die Resourcen der Neuronen benötigt. Möchte man wie beim unüberwachten Lernen durch die Positionierung der Neuronenzentren im Eingaberaum einfach eine adaptive Vektorquantisierung vornehmen, die den Eingaberaum möglichst gut überdeckt, so bietet sich als Fehlermaß für die Resourcen der quadratische Euklidische Abstand zwischen der letzten Eingabe und dem Zentrum der am besten passenden neuronalen Einheit an, so daß der Resource-Wert gemäß

$$\Delta r(bmu_t) = \gamma_{res} \cdot r(bmu_t) + (1 - \gamma_{res}) \cdot \|s_t - c_{bmu_t}\|^2, \quad \gamma_{res} \in]0,1[\tag{6.36}$$

aktualisiert wird. Eine weitere Möglichkeit, die Resourcen der Neuronen zu aktualisieren, besteht darin, den Fehler zu verwenden, der bei dem TD-Lernen der Kritik V anfällt (TD-Fehler).

$$\Delta r(bmu_t) = \gamma_{res} \cdot r(bmu_t) + (1 - \gamma_{res}) \cdot |r_t + \gamma \cdot V(s_{t+1}) - V(s_t)|, \quad \gamma_{res} \in]0,1[. \tag{6.37}$$

Bei der Verwendung dieses Fehlermaßes wird nicht so sehr auf eine gute Vektorquantisierung Wert gelegt, sondern vielmehr versucht, den TD-Fehler der Kritik zu minimieren. Das ist insofern wichtig, als daß eine gute Kritik V das Lernen der Ausgabegewichte der Politik beschleunigt. Problematisch ist jedoch, daß die Neuronenzentren nicht nur die Eingabegewichte der Kritik, sondern auch der Politik darstellen. Eine Trennung zwischen diesen beiden Komponenten in unterschiedliche FDCS-Netzwerke löst zwar dieses Problem, führt aber auch zu einer komplexeren Architektur.
Wird ein Neuron aufgrund der distanzgetriebenen Einfügestrategie bei s_t eingefügt, so werden geeignete Ausgabegewichte für das neue Neuron benötigt. Die einfachste Lösung ist hier, die berechnete Ausgabe des Netzwerkes zum Zustand s_t als Ausgabegewichtsvektor für das neue Neuron zu verwenden. Als neues Neuron wird

$$u_{new} = (s_t, \pi(s_t), V(s_t)) \tag{6.38}$$

in das Netzwerk $\mathcal{R}$ eingefügt. Eine Alternative, die die aktuelle Belohnung für die Kritik stärker berücksichtigt, stellt das Einfügen des folgenden Neurons dar:

$$u_{new} = (s_t, \pi(s_t), r_t + \gamma \cdot V(s_{t+1})). \tag{6.39}$$

Der Lernalgorithmus

Damit sind alle Einzelheiten der Adaption eines FDCS-REINFORCE-Netzwerkes vorgestellt. Der komplette Lernalgorithmus wird in dem Algorithmus aus Abbildung 6.2 zusammengefaßt.

$t := 0$;
Beobachte Startzustand s_0;
for $u \in U$ **do** $\tau(u)_0 = 0$;
do
    Bestimme bmu_t und $second_t$ zu s_t;
    $a^t := \pi(s_t)$;
    Führe Aktion a^t aus, gelange in Zustand s_{t+1} und erhalte direkte Belohnung r_t;
    for $u \in U$ **do**
        $\tau(u)_t := \lambda \cdot \tau(u)_{t-1} + \frac{\partial}{\partial (w_u)_{2m+1}} V(s_t)$;
    if neues Neuron bei s_t einfügen **then**
        $u_{new} := (s_t, \pi(s_t), V(s_t))$;
        $\tau(u_{new}) := 0$;
        $r(u_{new}) := 0$;
        $\rho(u_{new}, bmu_t) := \rho(bmu_t, u_{new}) := 1$;
        $\rho(u_{new}, second_t) := \rho(second_t, u_{new}) := 1$;
        $L := L \cup \{(u_{new}, bmu_t), (bmu_t, u_{new}), (u_{new}, second_t), (second_t, u_{new})\}$;
        $U := U \cup \{u_{new}\}$;
    else
        $err_t := r_t + \gamma \cdot V(s_{t+1}) - V(s_t)$;
        for $u \in U$ **do** {Adaptieren der Kritik}
            $(w_u)_{2m+1} := (w_u)_{2m+1} + \beta_V \cdot err_t \cdot \tau(u)_t$;

for $u \in \{bmu_t\} \cup Nh(bmu_t)$ **do** {Adaptieren der Politik}
  for $i \in \{1, \cdots, m\}$ **do**

$$(w_u)_{2i-1} = (w_u)_{2i-1} + \beta_\mu \cdot err_t \cdot \frac{a_i^t - \mu_i(s_t)}{\sigma_i(s_t)^2} \cdot \frac{\partial}{\partial (w_u)_{2i-1}} \mu_i(s_t);$$

$$(w_u)_{2i} = (w_u)_{2i} + \beta_\sigma \cdot err_t \cdot \frac{(a_i^t - \mu_i(s_t))^2 - \sigma_i(s_t)^2}{\sigma_i(s_t)^3} \cdot \frac{\partial}{\partial (w_u)_{2i}} \sigma_i(s_t);$$

  od;
od;
Adaptiere laterale Verbindungsstruktur und Resourcen von $\mathcal{R}$;
Adaptiere Neuronenzentren von $\mathcal{R}$ nach fehlermodulierten Kohonen-Regel;
fi;
$t := t + 1$;
loop;

Abbildung 6.2: Lernalgorithmus für FDCS-REINFORCE-Netzwerke

6.3.2 FDCS-REINFORCE-Architektur zur adaptiven Hindernisvermeidung

In diesem Abschnitt wird der FDCS-REINFORCE-Ansatz zum Erlernen eines Hindernisvermeidungsverhaltens eines AMR's eingesetzt.
Die Lernaufgabe besteht daraus, ein autonomes mobiles Roboterfahrzeug, das in einer unbekannten Umgebung ausgesetzt ist, durch Reinforcement-Lernen selbständig ein Hindernisvermeidungsverhalten erlernen zu lassen.
Als grundlegende Kontrollarchitektur wird die letzte FDCS-basierte Hindernisvermeidungskomponente verwendet, bei welcher der normale FDCS-Regler durch das FDCS-REINFORCE-Netzwerk ersetzt wird.
Ein ähnlicher Versuch wurde bereits in [PM92] durchgeführt. Dort wurden alle Aktionen und die Kritik jeweils in einer eigenen CMAC-Tabelle gespeichert.
In [SE94] wurde dagegen ein REINFORCE-Ansatz mit zwei mehrschichtigen Perzeptron-Netzwerken implementiert und auf das Problem der Stabbalance angewendet. Auch hier wurden Kritik und Politik in getrennten MLP-Netzwerken realisiert.
Hier wird der zuvor eingeführte FDCS-REINFORCE-Ansatz anhand der Hindernisvermeidung getestet. Dieser Ansatz kommt im Gegensatz zu [SE94] und [PM92] mit nur einem FDCS-Netzwerk aus und vereint Politik und Kritik in einem einzigen Netzwerk.

Die Simulation

Als Eingabemerkmale des Zustandsraumes werden hier acht direkt eingelesene Ultraschallmeßwerte verwendet. Obwohl die FDCS-basierte Architektur zur Hindernisvermeidung eine weitgehende Vorverarbeitung der Ultraschallmeßwerte vorsieht, wird an dieser Stelle darauf explizit verzichtet. Damit wird die Lernaufgabe möglichst schwierig gestaltet. Als Sensorkonfiguration wird hier die gleiche Anordnung verwendet, die auch schon beim FDCS-basierten Ansatz verwendet wurde.
Wie auch in einigen anderen Robotersimulationen ([RN94], [PM92]) zum Lernen von Verhalten werden auch hier die Ultraschallsensoren zunächst nur durch einzelne *Meßstrahlen* simuliert. Da dieses keine adäquate Beschreibung der Ultraschallsensoren darstellt, sondern

eher einem Laserentfernungssensor entspricht, wird schließlich untersucht, wie robust das Lernverfahren ist. Um dieses zu untersuchen, werden in einem zweiten Versuch die Sensormeßwerte mit einem Rauschen versehen und die Charakteristik der Ultraschallsensoren wird durch drei Meßstrahlen realistisch simuliert.

Zu Beginn der Simulation steht der AMR in einem Raum mit zufällig ausgewählten Hindernissen. Das Fahrzeug hat nur das Vorwissen, mit einer bestimmten Geschwindigkeit v_0 geradeaus zu fahren. Sowie das Fahrzeug gegen ein Hindernis fährt oder sich bis auf 20 cm einem Hindernis nähert, wird der Versuch als erfolglos abgebrochen. Der AMR wird dann mittels eines fest implementierten Neuorientierungsverhalten in eine neue Ausgangsposition gebracht, so daß die Frontsensoren innerhalb von 2 m kein Hindernis erkennen. Dann wird ein neuer Versuch innerhalb desselben Raumes mit den gleichen Hindernissen eingeleitet. Dieses Vorgehen hat den Vorteil, daß ein solcher Versuch prinzipiell auch mit dem realen AMR in einer festen Umgebung durchgeführt werden kann. Dieses ist beispielsweise bei [RN94] nicht möglich, da dort das Fahrzeug nach jedem Fehlversuch in einer zufällig neu konstruierten Umgebung plaziert wird.

Als Aktionen stehen für den hier verwendeten AMR wieder die Fahrtgeschwindigkeit v und die Rotationsgeschwindigkeit ω zur Verfügung. Die Politik, die durch das FDCS-Netzwerk berechnet wird, liefert dementsprechend zu jedem Zustand einen Mittelwert und eine Standardabweichung für die Fahrt- und Rotationsgeschwindigkeit $(\mu_v, \sigma_v, \mu_\omega, \sigma_\omega)$. Damit zu Beginn der Simulation überhaupt eine Aktion ausgeführt wird, hat das FDCS-Netzwerk genau eine Fuzzy-Regel als Vorwissen:

$$\begin{aligned}&\textbf{if } (s_0 \approx 5000) \wedge \cdots \wedge (s_7 \approx 5000) \textbf{ then}\\ &\qquad (\mu_v(s) \approx v_0) \wedge (\sigma_v(s) \approx \sigma_v^0) \wedge (\mu_\omega(s) \approx 0) \wedge (\sigma_\omega(s) \approx \sigma_\omega^0) \wedge (V(s) \approx 0).\end{aligned}$$

Dabei bezeichnen v_0 die Fahrtgeschwindigkeit mit der anfangs geradeaus gefahren wird und σ_v^0, σ_ω^0 die Standardabweichungen mit denen die Aktionen anfangs um die vorgegebenen Mittelwerte verrauscht sind.

Belohnungen

Ein Zustand des AMR's besteht ohne Vorverarbeitung der Sensoren aus acht Ultraschallmeßwerten $s_t = (s_t^0, \cdots, s_t^7)$. Wird eine Aktion a^t vom AMR ausgeführt, so gelangt das Fahrzeug in eine neuen Zustand s_{t+1}. Um aus diesem Zustandsübergang eine direkte Belohnung für die Aktion a^t abzuleiten, wird eine Zustandsbewertung $\psi : \mathcal{S} \rightarrow I\!R$ eingeführt, die jedem Zustand s_t eine Bewertung $\psi(s_t)$ zuordnet. Häufig wird diese Zustandsbewertung als direkte Belohnung verwendet:

$$r_t = \psi(s_{t+1}). \tag{6.40}$$

Als Belohnung für die Ausführung der Aktion a^t im Zustand s_t wird also die Bewertung des Folgezustandes s_{t+1} verwendet. Obwohl dieser Ansatz vielfach verwendet wird (z. B. in [SE94]), besitzt er den Nachteil, daß der Ausgangszustand s_t weitgehend unberücksichtigt bleibt.

Eine Alternative, die sowohl Ausgangs- als auch Folgezustände berücksichtigt, verwendet als Belohnungssignal die Differenz aus beiden Zustandsbewertungen:

$$r_t = \psi(s_{t+1}) - \psi(s_t). \tag{6.41}$$

Da eine Politik, die in einem Zustand immer die Aktion auswählt, die eine maximale Zustandsbewertung der Folgezustände bewirkt, zu einer Politik äquivalent ist, die in einem Zustand immer die Aktion auswählt, die eine maximale Verbesserung (Differenz) der Zustandsbewertungen bewirkt, können diese Differenzen von Zustandsbewertungen selbst dann als Belohnungssignal verwendet werden, wenn die akkumulierten Zustandsbewertungen maximiert werden sollen.

Sind $s_i \in I\!R$, $i \in I \subseteq \{1, \cdots, n\}$ Komponenten eines Zustandes $s \in \mathcal{S}$, also z. B. Sensormeßwerte oder aber auch aufbereitete Merkmale wie z. B. die Orientierung des Fahrzeuges zu einer Wand, so sind daraus geeignete Zustandsbewertungen zu gewinnen. Soll der Wert einer Komponente durch den Lernvorgang maximiert werden, so kann

$$\bar{s_i} = \sqrt{\frac{s_i - p_i}{q_i - p_i}} \tag{6.42}$$

gesetzt werden, falls in jedem Zustand $s_i \in [p_i, q_i]$ für $q_i, p_i \in I\!R$ gilt. Soll dagegen der Betrag einer Komponente durch den Lernvorgang minimiert werden, so setzt man beispielsweise

$$\bar{s_i} = 1 - \sqrt{\frac{|s_i| - \min\{|p_i|, |q_i|\}}{\max\{|p_i|, |q_i|\} - \min\{|p_i|, |q_i|\}}}. \tag{6.43}$$

Durch Konvexkombination der Bewertung einzelner Zustandskomponenten erhält man eine geeignete Zustandsbewertung:

$$\psi(s) = \frac{1}{\sum\limits_{i \in I} c_i} \cdot \sum_{i \in I} c_i \cdot \bar{s_i}. \tag{6.44}$$

Die Koeffizienten c_i, $i \in I$ gewichten dabei die einzelnen Merkmale und geben damit an, wie wichtig die einzelnen Merkmale für das zu erlernende Verhalten sind.

Durch die Verwendung der Normierung in (6.42), (6.43) und (6.44) wird erreicht, daß $\psi(s) \in [0, 1]$ gilt.

Die Wurzel wird bei den Bewertungen der einzelnen Komponenten verwendet, um einen nichtlinearen Zusammenhang zwischen den einzelnen Merkmalen herzustellen.

Um dieses zu verdeutlichen betrachte man folgendes Beispiel. Für einen AMR geben s_0 und s_1 den linken und rechten Seitenabstand zu Hindernissen an. Beide Abstände sollen maximiert werden. Verwendet man $\psi(s) = s_0 + s_1$ als Zustandsbewertung, so ist ψ eine lineare Abbildung. Angenommen das Fahrzeug fährt durch einen Korridor der Breite b, so gilt $\psi(s) = b - d$, wenn d zusätzlich die Fahrzeugbreite angibt. Die Zustandsbewertung ist also konstant, unabhängig davon, ob der AMR in der Mitte des Ganges oder sehr dicht an nur einer Wand fährt.

Definiert man dagegen die Zustandsbewertung durch eine nichtlineare Funktion $\psi(s) = \sqrt{s_0} + \sqrt{s_1}$, so nimmt ψ bei $s_0 = s_1 = \frac{b-d}{2}$ den Maximalwert an, da im Korridor $s_0 + s_1 = b - d$ gilt.

Die Verwendung der nichtlinearen Zustandsbewertung führt also zu einer optimale Politik, die den AMR in der Mitte durch einen Gang lenkt.
Zusätzlich zu den Belohnungen, die direkt aus den Zuständen gewonnen werden, wird eine Belohnung bzw. Bestrafung am Ende eines Versuches vergeben. Stößt der AMR gegen ein Hindernis oder fährt er zu nah an ein Hindernis heran, so wird der Versuch abgebrochen und mit einer sehr niedrigen Belohnung r_{final} bestraft.

Simulationen

In den folgenden Simulationen wird die Effektivität des FDCS-REINFORCE-Lernverfahrens am Beispiel der Hindernisvermeidung gezeigt.
Aus den acht Sensormeßwerten, die den Zustandsraum für das FDCS-Netzwerk bilden, werden die Meßwerte der Sensoren #0 - #3 für die Gewinnung einer Zustandsbewertung herangezogen. Da diese Werte maximiert werden sollen, wird die Zustandsbewertung nach (6.44) und (6.42) mit den Koeffizienten $c_0 = \cdots = c_3 = 1$ und $I = \{0, \cdots, 3\}$ berechnet.
Das Ziel ist eine möglichst kollisionsfreie Fahrt durch die Umwelt des AMR's. Ein Versuch wird beendet, wenn das Fahrzeug gegen ein Hindernis fährt oder sich einem Hindernis bis auf 20 cm nähert. Ein Versuch wird ebenfalls beendet, wenn der AMR 200 Schritte nacheinander kollisionsfrei durch die Umgebung gefahren ist. Nach jedem Fehlversuch wird das Fahrzeug so neu orientiert, daß es innerhalb von 2 m kein frontales Hindernis erkennt. Erst dann wird ein neuer Versuch gestartet. Hat sich das Fahrzeug in seiner Umgebung festgefahren, so daß keine Neuorientierung möglich ist, wird der AMR auf seine anfängliche Startposition zurückgebracht. Nach jedem erfolgreichen Versuch (200 Schritte ohne Kollision) wird das Fahrzeug zufällig neu orientiert. Dann wird es wie nach einem Fehlversuch so orientiert, daß die Frontsensoren innerhalb von 2 m kein Hindernis entdecken. Erst dann wird mit einem neuen Versuch fortgesetzt. Die zufällige Neuorientierung nach einem erfolgreichen Versuch ist notwendig, damit das Fahrzeug gezwungen wird, neue Regionen der Umgebung zu befahren. Sonst ist es möglich, daß das Fahrzeug in nur einer bestimmten Region der Umgebung fährt und dadurch sein Verhalten speziell an diese Region anpaßt.

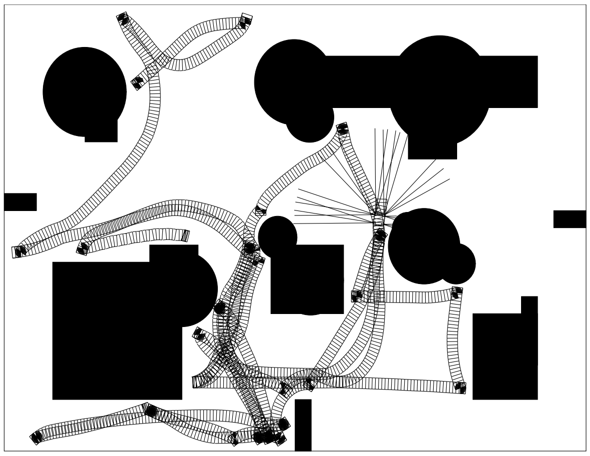


Abbildung 6.3: Beginn der Lernphase

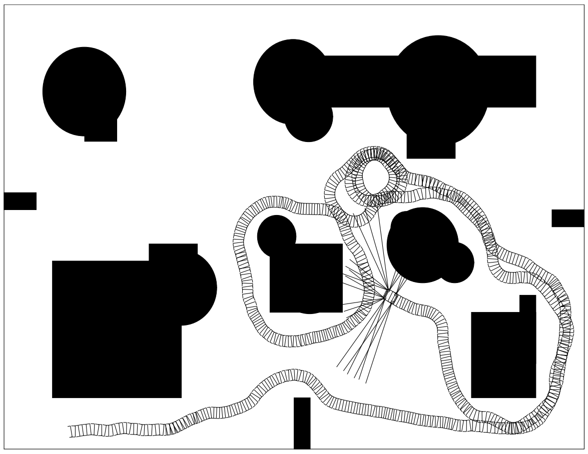


Abbildung 6.4: Erlerntes Verhalten

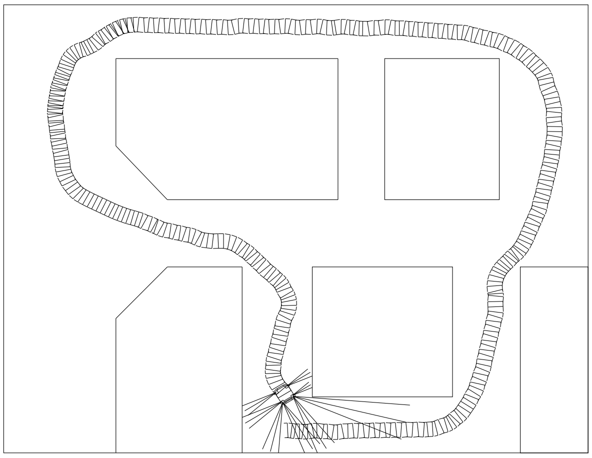


Abbildung 6.5: Erlerntes Verhalten in einer neuen Umgebung

Als grundlegende Kontrollarchitektur wird die Hindernisvermeidungskomponente des letzten FDCS-Ansatzes verwendet. Dieses Modul sieht eine Aktionskorrektur vor, die dazu dient, den AMR vor unerlaubten Aktionen zu schützen. Da einem Hindernis um so besser ausgewichen werden kann, je größer die Rotationsgeschwindigkeit ist, ist zu erwarten, daß sich die Politik so entwickelt, daß in bestimmten Situationen große Rotationsgeschwindigkeiten erzeugt werden. Um die Sicherheit des Fahrzeuges zu gewährleisten, unterdrückt die Aktionskorrektur Geschwindigkeiten, die außerhalb eines Bereiches zwischen 0 $\frac{mm}{s}$ und 1000 $\frac{mm}{s}$ und Rotationsgeschwindigkeiten, die außerhalb eines Bereiches zwischen $-20° \ s^{-1}$ und $20° \ s^{-1}$ liegen. Aktionen, die außerhalb dieser Bereiche liegen, werden durch die jeweils überschrittene Grenze ersetzt.
Die Abbildung 6.3 zeigt das Verhalten des AMR's zu Beginn der Lernphase. Der AMR fährt mit nahezu konstanter Geschwindigkeit und stößt immer wieder an Hindernisse. Während der Lernphase erwirbt der AMR allmählich das in Abbildung 6.4 dargestellte Hindernisvermeidungsverhalten. Der Roboter schafft es, durch die Trainingsumgebung kollisionsfrei zu navigieren. Selbst in einer neuen Umgebung, die sich von der Trainingsumgebung stark unterscheidet, gelingt es dem AMR, mit seinem erlernten Verhalten kollisionsfrei durch die Umgebung zu fahren (Abbildung 6.5).
Für dieses Experiment wurden als Lernparameter $\beta_V = 0.005$, $\beta_\mu = 0.31$, $\beta_\sigma = 0.061$, $\gamma = 0.99$, $\lambda = 0.19$, $v_0 = 400\frac{mm}{s}$ und $\sigma_v^0 = \sigma_\omega^0 = 1$ verwendet. Als direkte Belohnungen wurden Differenzen von Zustandsbewertungen verwendet und ein Fehlversuch wurde zusätzlich mit $r_{final} = -100$ bestraft. Für die Adaption der Resourcen wurde der TD-Fehler mit $\gamma_{res} = 0.9$ herangezogen. Zur Zentrenadaption wurden $\varepsilon_{bmu} = 0.0001$ und $\varepsilon_{Nh} = 0.000001$ bei Verwendung der fehlermodulierten Kohonen-Regel gesetzt. Neuronen wurden nach der gemischten Strategie mit $\lambda_{ins} = 50000$ und $\delta = 0.4$ eingefügt, wobei $N_{max} = 100$ die maximale Neuronenanzahl war. Für die Adaption der Verbindungsstruktur wurde als Alterungsrate $\alpha_c = 0.99$ und $\theta = 0.3$ gewählt. Als Basisfunktionen wurden Gaußsche Aktivierungsfunktionen mit Aktivierungsradius $\sigma = 0.3$ verwendet. Der Eingaberaum wurde auf $[0,1]^8$ normiert und die Evaluierungsfunktion normalisiert berechnet.

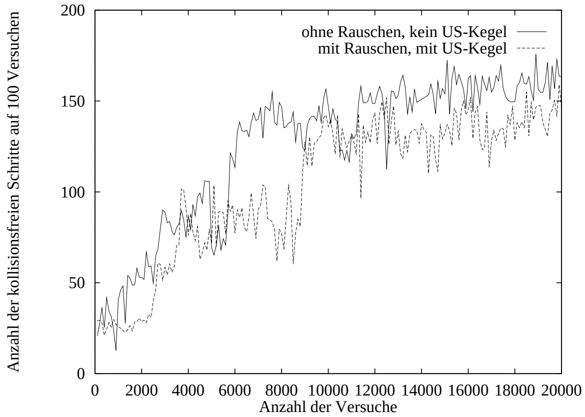


Abbildung 6.6: Schrittzahl

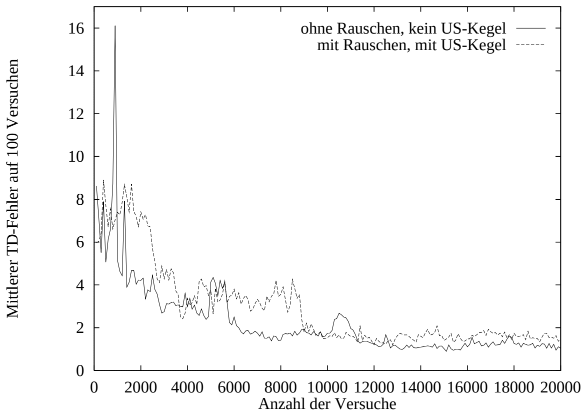


Abbildung 6.7: TD-Fehler

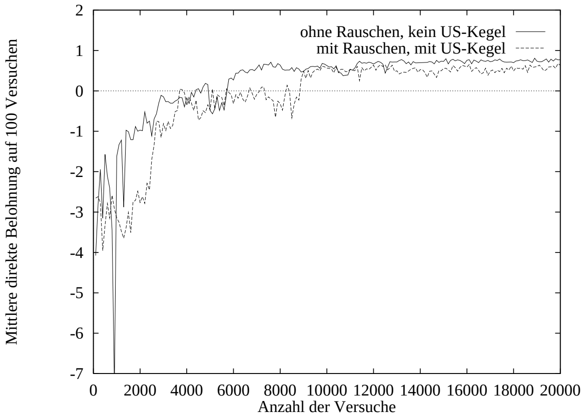


Abbildung 6.8: Belohnungen

Die Simulation wurde mit einer Zeitschrittweite von $\Delta t = 0.5s$ durchgeführt.

Eine systematische Untersuchung der wichtigsten Parameter fällt bei diesem Versuchsaufbau schwer. Zwei unterschiedliche Parametersätze sind hinsichtlich des erzielten Lernerfolges schlecht zu vergleichen. Kleinste Änderungen in den Parametern führen bereits dazu, daß der AMR anders die feste Umgebung erkundet und dadurch auch aus anderen Erfahrungen sein Verhalten erlernt. Der Unterschied in den gesammelten Erfahrungen kann eine Auswirkung auf den Lernerfolg haben, der die Wirkung der Parameteränderung übersteigt. Aus diesem Grunde wurde auf eine Optimierung der Parameter verzichtet und lediglich einige verschiedene Parametersätze wurden ausprobiert, die jedoch ein ähnliches Lernverhalten ergeben haben.

Die Abbildungen 6.6 - 6.8 zeigen den zeitlichen Verlauf des Lernvorganges. Dabei wird das Lernverfahren bei gleichen Parametern für ideale Sensoren (kein Ultraschallkegel und kein Rauschen) und für realistische Sensoren (Ultraschallkegel und Rauschen) verglichen. Das Rauschen wurde durch ein normalverteiltes Rauschen mit einer Standardabweichung von

5 *cm* simuliert.
Die Abbildung 6.6 zeigt den zeitlichen Verlauf des Lernvorganges. Dargestellt ist die über 100 Versuche gemittelte Anzahl der kollisionsfreien Schritte. Da nach 200 Schritten eine Neuorientierung durchgeführt wird, sind maximal 200 kollisionsfreie Schritte zu erreichen. Während des Lernvorganges werden 200 Schritte jedoch nicht erreicht. Dieses liegt daran, daß nach 200 kollisionsfreien Schritten das Fahrzeug zufällig neu orientiert wird. Steht das Fahrzeug zu diesem Zeitpunkt in einer sehr engen Umgebung, so kann die Neuorientierung das Fahrzeug direkt vor einem Hindernis plazieren. In diesem Fall ist das Fahrzeug auch mit einer gut erlernten Politik nicht in der Lage, ohne Kollision zu entkommen.
Die Abbildung 6.7 zeigt den zeitlichen Verlauf des über 100 Versuche gemittelten TD-Fehlers. Das Absinken des TD-Fehlers zeigt den Lernfortschritt der Kritikfunktion V.
In der Abbildung 6.8 ist schließlich der zeitliche Verlauf der über 100 Versuche gemittelten direkten Belohnungen dargestellt. Zu Beginn der Lernphase besitzen die mittleren Belohnungen negative Werte. In diesem Zeitraum überwiegt die Bestrafung, die durch Kollisionen vergeben wird. Mit zunehmenden Lernfortschritt steigt die mittlere Belohnung gegen den maximal zu erreichenden Belohnungswert 1. Kollisionen werden mithin immer seltener. Der Belohnungswert 1 wird jedoch nicht ganz erreicht, da dazu in jedem Schritt alle Sensoren kein Hindernis detektieren dürften. Das ist in einer Umgebung voller Hindernisse aber nicht zu erreichen, so daß hier nach ca. 8000 Versuchen keine nennenswerte Verbesserung der Politik mehr erreicht wird.
Durch den Vergleich der ideal und realistisch simulierten Sensoren wird deutlich, daß das Erlernen von Verhalten unter realen Bedingungen deutlich schwieriger ist. Die dargestellten Lernkurven der realistischen Simulation fallen schlechter aus, als die Lernkurven für die ideale Simulation. Dennoch ist das Lernverfahren robust genug, auch unter realistisch simulierten Bedingungen ein vernünftiges Verhalten zu erlernen.
Dieses deutet darauf hin, daß das FDCS-REINFORCE-Lernverfahren prinzipiell auch für den realen Roboter eingesetzt werden kann, obwohl die lange Lerndauer einen realen Einsatz fraglich erscheinen läßt.

6.3.3 Reellwertiges modifiziertes Q-Lernen mit FDCS- Netzwerken

Der Reinforcement-Lernansatz der reellwertigen stochastischen REINFORCE-Algorithmen eignet sich recht gut für Lernaufgaben wie sie im vorherigen Abschnitt angegeben wurden. Die theoretischen Grundlagen für diese Verfahren wurden in [Wil92] dargestellt, wo gezeigt werden konnte, daß die Adaption der Gewichte der Politik im Mittel einem Gradientenaufstiegsverfahren entspricht. Der Aufdatierungsvektor der Gewichte zeigt im Mittel in die Richtung des Gradienten des Erwartungswertes der Ertragsfunktion. Dabei kommen die REINFORCE-Ansätze gänzlich ohne eine explizite Berechnung dieser Gradienten aus.
Im folgenden wird dieser Ansatz mit einem neuen Lernverfahren verglichen, das ein direktes und kein stochastisches Gradientenverfahren darstellt.
Motiviert wird dieses neue Verfahren durch das Q-Lernen. Die Grundidee besteht darin, daß eine Politik $\pi : \mathcal{S} \rightarrow \mathcal{A}$ und eine reellwertige Q-Funktion $Q : \mathcal{S} \times \mathcal{A} \rightarrow I\!R$ jeweils durch ein FDCS-Netzwerk berechnet werden. Die Q-Funktion soll Vorhersagen über den

zu erwartenden zukünftigen Ertrag in einem Zustand $s \in \mathcal{S}$ bei Ausführung einer Aktion $a \in \mathcal{A}$ treffen. Die Politik π wird durch die Q-Funktion so adaptiert, daß der zu erwartende durch Q vorhergesagte Ertrag maximiert wird.
Dazu betrachte man einen Zustandsraum $\mathcal{S} \subseteq I\!R^n$ und einen Aktionsraum $\mathcal{A} \subseteq I\!R^m$. Die Politik wird durch ein π-FDCS-Netzwerk $\mathcal{P} = (\mathcal{S}, \mathcal{A}, U_\mathcal{P}, L_\mathcal{P}, \Pi_\mathcal{P}, \rho_\mathcal{P}, \alpha_\mathcal{P}, r_\mathcal{P}, \delta_\mathcal{P})$ dargestellt und die reellwertige Q-Funktion durch ein π-FDCS-Netzwerk $\mathcal{Q} = (\mathcal{S} \times \mathcal{A}, I\!R, U_\mathcal{Q}, L_\mathcal{Q}, \Pi_\mathcal{Q}, \rho_\mathcal{Q}, \alpha_\mathcal{Q}, r_\mathcal{Q}, \delta_\mathcal{Q})$ (vgl. Abbildung 6.9).

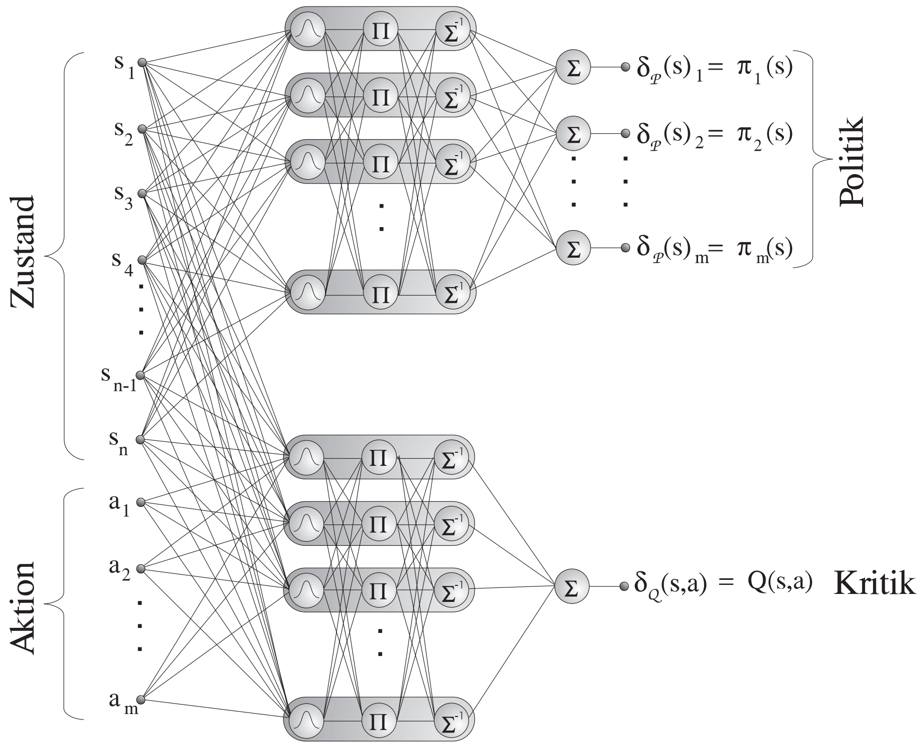


Abbildung 6.9: π-FDCS-Netzwerke für das reellwertige modifizierte Q-Lernen

Somit berechnet sich zu einem Zustand $s \in \mathcal{S}$ die durch die Politik berechnete Aktion gemäß

$$\pi(s) = \delta_\mathcal{P}(s). \tag{6.45}$$

Die Q-Funktion wird entsprechend durch das π-FDCS-Netzwerk $\mathcal{Q}$ berechnet:

$$Q(s,a) = \delta_\mathcal{Q}(s,a). \tag{6.46}$$

Adaption der Q-Funktion

Problematisch an der Aufdatierungsregel (6.9) des klassischen Q-Lernens in Verbindung mit einer reellwertigen Q-Funktion ist die notwendige Bestimmung einer Aktion a. Die Aktion muß so gewählt werden, daß die Q-Funktion ihr Maximum bei (s_{j+1}, a) annimmt. Falls die Aktionsmenge endlich ist, kann die Aktion in endlich vielen Schritten bestimmt

werden. Ist dagegen der Aktionsraum kontinuierlich, so ist eine Maximumbestimmung nur mit numerischen Verfahren zu lösen. Diese finden allerdings in der Regel keine globalen Maxima und lassen außerdem die Rechenzeit für einen Adaptionsschritt unakzeptabel ansteigen.
Dazu wird in [RN94] festgestellt, daß $\max_{a \in \mathcal{A}} Q(s_{j+1}, a)$ nicht die beste Approximation der Ertragsfunktion $V(s_{j+1})$ ist. In den ersten Schritten der Lernphase ist diese Approximation extrem fehlerbehaftet, da die Q-Funktion noch nicht ausreichend erlernt wurde. Aber auch in einer späteren Lernphase stellt diese Approximation der Ertragsfunktion eher eine *Überapproximation* dar [TS93].
Daher wird in [RN94] statt der Verwendung der Maximumbestimmung einfach die bisherige Q-Funktion verwendet. Dieser Ansatz kommt einem reellwertigen Verfahren sehr entgegen, da eine aufwendige Bestimmung der Maxima entfällt.
Weiterhin wird der klassische Q-Ansatz in [RN94] so erweitert, daß für die Adaption der Q-Funktion das TD-Lernen verwendet wird. Damit stellt sich die Adaptionsregel für das Netzwerk $\mathcal{Q}$ im Schritt t wie folgt dar:

$$\text{Für alle } u \in U_{\mathcal{Q}} :$$
$$\Delta w_u = \beta \cdot (r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \cdot \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial}{\partial (w_u)_k} Q(s_k, a_k). \quad (6.47)$$

Dabei bezeichnen s_t, a_t den Zustand und die ausgeführte Aktion zum Zeitpunkt t und $(w_u)_k$ gibt die Ausgabegewichte des Neurons u zum Zeitpunkt k an. Diese Lernregel wird in [RN94] im Zusammenhang mit neuronalen Netzwerken zur Speicherung der Q-Funktion als *MCQ-Lernen* (Modified Connectionist Q-learning) bezeichnet.
Unter Verwendung einer Spur wie beim FDCS-REINFORCE-Ansatz erhält man daraus die folgende Aufdatierungsregel:

Adaption der Spur:

$$\text{Für alle } u \in U_{\mathcal{Q}}: \quad \tau(u)_0 = 0 \quad \text{(Initialisierung)} \quad (6.48)$$
$$\text{Für alle } u \in U_{\mathcal{Q}}: \quad \tau(u)_t = \lambda \cdot \tau(u)_{t-1} + \frac{\partial}{\partial w_u} Q(s_t, a_t) \quad \text{(Iteration)} \quad (6.49)$$

Adaption der Ausgabegewichte der Q-Funktion Q :

$$\text{Für alle } u \in U_{\mathcal{Q}}:$$
$$\Delta w_u = \beta_Q \cdot (r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \cdot \tau(u)_t \quad (6.50)$$

Obwohl die Zentren nach einer ähnlichen Aufdatierungsregel adaptiert werden könnten, werden diese hier mit der fehlermodulierten Kohonen-Regel adaptiert. Dadurch erspart man sich eine aufwendige Berechnung der Gradienten nach den Eingabegewichten.

Adaption der Politik

Ein Vorteil des klassischen Q-Lernens liegt darin, daß bei endlichem Aktionsraum durch die Q-Funktion bereits eine gierige Politik gegeben ist (vgl. (6.11)). Da hier kein endlicher

Aktionsraum gegeben ist, muß die Politik explizit erlernt werden. Durch die erlernte Q-Funktion wird ein Modell angegeben, welches die Wechselwirkung zwischen dem Agenten und seiner zukünftig zu erwartenden Belohnung angibt. Anhand dieses Modells wird die Politik so adaptiert, daß der Aufdatierungsvektor der Politik in Richtung des Gradienten der Q-Funktion nach den Gewichten zeigt. Eine derartige Adaption der Politik bewirkt bei einer korrekt erlernten Q-Funktion, daß in einem Zustand die Aktionen so verändert werden, daß eine Steigerung des durch die Q-Funktion angegebenen Ertrages zu erwarten ist. Zum Zeitpunkt t werden die Gewichte der Politik folgendermaßen aktualisiert:

Adaption der Politik π **:**

Für alle $u \in U_{\mathcal{P}}$:

$$\Delta c_u = \beta_\pi^1 \cdot \nabla_{c_u} Q(s_t, \pi(s_t)), \tag{6.51}$$

$$\Delta w_u = \beta_\pi^2 \cdot \nabla_{w_u} Q(s_t, \pi(s_t)). \tag{6.52}$$

Die Berechnung der Gradienten ist aufgrund der Verwendung der FDCS-Netzwerke nicht einfach. Daher wird die Gradientenbestimmung im Anhang B separat erklärt.

Entfaltung der Netzwerke

Wie beim FDCS-REINFORCE-Ansatz müssen zu Beginn der Lernphase Neuronen im Raum $I \times O$ eingefügt werden. Dazu werden sowohl wieder nach der fehlergetriebenen als auch nach der distanzgetriebenen Strategie Neuronen eingefügt. Als Fehlermaß für die Resourcen der Politik bieten sich die Euklidischen Abstände gemäß (6.36) an.
Für die Q-Funktion bietet sich eher der TD-Fehler als Fehlermaß an, so daß die Resourcen der Q-Funktion ähnlich wie in (6.37) gebildet werden.

$$\Delta r(bmu_t) = \gamma_{res} \cdot r(bmu_t) + (1-\gamma_{res}) \cdot |r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)|, \quad \gamma_{res} \in]0,1[. \tag{6.53}$$

Soll nun ein neues Neuron nach einer Einfügestrategie im Netzwerk $\mathcal{P}$ eingefügt werden, so verwendet man bei der fehlergetriebenen Strategie den explizit berechneten Eingabegewichtsvektor c_{new} und $\pi(c_{new})$ als Ausgabegewichtsvektor.
Bei der distanzgetriebenen Einfügestrategie wird als Eingabegewicht der aktuelle Zustand s_t und als Ausgabegewicht des neuen Neurons $\pi(s_t)$ verwendet. Man erhält damit

$$u_{new} = (s_t, \pi(s_t)). \tag{6.54}$$

Entsprechend wird für ein Neuron, das in das Netzwerk $\mathcal{Q}$ eingefügt werden soll, der alte Q-Wert für den Ausgabegewichtsvektor verwendet

$$u_{new} = (s_t, a_t, Q(s_t, a_t)), \tag{6.55}$$

oder wenn die letzte Belohnung zusätzlich berücksichtigt werden soll, so wird

$$u_{new} = (s_t, a_t, r_t + \gamma \cdot Q(s_{t+1}, a_{t+1})) \tag{6.56}$$

verwendet. In den folgenden Simulationen wurden die besseren Ergebnisse mit (6.55) erzielt. Insbesondere bei einer Kollision ($r_t = r_{final}$) ist der direkte Einfluß von r_t auf das Netzwerk von Nachteil.

Exploration des Zustands-Aktions-Raumes

Beim klassischen Q-Lernen ist eine Voraussetzung für die Konvergenz des Lernverfahrens, daß jedes Zustands-Aktions-Paar unendlich oft besucht wird. Ähnlich ist es bei der hier vorgestellten reellwertigen Variante. Damit die Q-Funktion für die Adaption der Politik verwendet werden kann, müssen in einem Zustand $s \in \mathcal{S}$ für das Training der Q-Funktion möglichst viele verschiedene Aktionen ausprobiert werden. Beim klassischen Q-Lernen wird dieses Problem gelöst, in dem nicht immer die *greedy*-Politik ausgeführt, sondern beispielsweise mit der Boltzmann-Verteilung eine Aktion zufällig ausgewählt wird. Mit fortschreitender Zeit steigt die Wahrscheinlichkeit der *greedy*-Aktionen, so daß die ausgeführte Politik schließlich gegen die *greedy*-Politik konvergiert.
Eine ähnliche Technik zur Bestimmung der Aktionen bei ausreichender Exploration wird auch für die reellwertige Variante des Q-Lernens durchgeführt.
Dazu sei $p \in]0,1[$ eine Konstante. Ferner seien $\sigma_1, \cdots, \sigma_m \in I\!R$ vorgegebene Standardabweichungen, die das Rauschen um die Politik π für jede einzelne Komponente des Aktionsraumes angeben. Zum Zeitpunkt T befindet sich das System im Zustand $s \in \mathcal{S}$. Die Politik π berechnet dazu die Aktion $a = \pi(s) \in \mathcal{A}$. Statt direkt a auszuführen, wird zur Exploration eine verrauschte Aktion

$$\tilde{a}_i = noise(a_i, p^T \cdot \sigma_i), \quad i \in \{1, \cdots, m\} \tag{6.57}$$

verwendet. Der Ausdruck $noise(\mu, \sigma)$ liefert eine normalverteilte Zufallsvariable mit wählbarem Erwartungswert μ und Standardabweichung σ.

Der Lernalgorithmus

Der in Abbildung 6.10 dargestellte Algorithmus faßt den reellwertigen modifizierten Q-Ansatz zusammen und gibt das Lernverfahren für einen Versuch an. Zu Beginn aller durchgeführten Versuche sei $T = 0$.

{Initialisierung}
t **:=** 0;
Beobachte Startzustand s_0;
for $u \in U_\mathcal{Q}$ **do**
    $\tau(u) = 0$;
{Iteration}
do
    {Berechne Aktion}
    a^t **:=** $\pi(s_t)$;
    $\tilde{a}^t$ **:=** $(noise(a_1^t, p^T \cdot \sigma_1), \cdots, noise(a_m^t, p^T \cdot \sigma_m))$;
    Führe Aktion $\tilde{a}^t$ aus und gelange in Zustand s_{t+1} und erhalte direkte Belohnung r_t;
    {Berechne Wert der Q-Funktion}
    Q_t **:=** $Q(s_t, \tilde{a}^t)$;
    {Adaption der Q-Funktion}
    if neues Neuron bei $(s_t, \tilde{a}^t)$ in $\mathcal{Q}$ einfügen **then**
        Einfügen von $(s_t, \tilde{a}^t, Q_t)$ in $\mathcal{Q}$ und Anpassen der Verbindungsstruktur;

```
    else
        if (t > 1) then
            err := r_{t-1} + γ · Q_t − Q_{t-1};
            for u ∈ U_Q do
                w_u := w_u + β_Q · err · τ(u);
            od;
            Adaptiere laterale Verbindungsstruktur und Resourcen von Q;
        fi;
        Adaptiere Neuronenzentren von Q nach fehlermodulierter Kohonen-Regel;
    fi;
    {Adaption der Spur}
    for u ∈ U_Q do
        τ(u) := λτ(u) + ∂/∂w_u Q(s_t, ã^t);
    od;
    {Adaption der Politik}
    if neues Neuron bei s_t in π einfügen then
        Einfügen von (s_t, a^t) in π und Anpassen der Verbindungsstruktur;
    else
        for u ∈ U_P do
            for i ∈ {1, ···, m} do
                (c_u)_i = (c_u)_i + β_π^1 · ∂/∂(c_u)_i Q(s_t, π(s_t));
                (w_u)_i = (w_u)_i + β_π^2 · ∂/∂(w_u)_i Q(s_t, π(s_t));
            od;
        od;
        Adaptiere laterale Verbindungsstruktur und Resourcen von π;
    fi;
    t := t + 1;
    T := T + 1;
loop;
```

Abbildung 6.10: Lernalgorithmus für reellwertiges modifiziertes Q-Lernen mit FDCS-Netzwerken

6.3.4 Anwendung auf die adaptive Hindernisvermeidung

Nachdem das reellwertige modifizierte Q-Lernen im vorherigen Abschnitt dargestellt wurde, soll seine Wirksamkeit nun am Beispiel der adaptiven Hindernisvermeidung untersucht werden.
Als Lernaufgabe dient dabei der gleiche Versuchsaufbau, wie er bereits im Abschnitt über die Anwendung des FDCS-REINFORCE-Ansatzes verwendet wurde.

Die Simulation

In ersten Versuchen wurden erneut acht Eingabemerkmale verwendet, mit denen allerdings keine zufriedenstellenden Lernerfolge erzielt wurden. Da RBF-basierte neuronale Netze prinzipiell Schwierigkeiten mit höherdimensionalen Eingaberäumen (Dimension ≥ 10) haben können [RN94], wurde die Eingabedimension des Problems dadurch drastisch gesenkt, daß weniger Ultraschallsensoren für die Eingabe herangezogen werden. Dieses erhöht einerseits die Schwierigkeit der Lernaufgabe, da ein adäquates Verhalten aus weniger Ein-

gangsgrößen abgeleitet werden soll. Andererseits kommt eine Dimensionsreduktion gerade dem modellbasierten Ansatz des reellwertigen modifizierten Q-Lernens entgegen, da sogar Gradienteninformationen aus dem Q-Netzwerk zur Adaption der Politik herangezogen werden, was die Forderungen an die Qualität des Q-Netzes drastisch erhöht.
Daher wurden als Eingabemerkmale nur die Sensoren #0 und #3 verwendet, so daß das Q-Netz zusammen mit den beiden Aktionskomponenten einen vierdimensionalen Eingaberaum erhält.
Die Durchführung der folgenden Experimente im Simulator ist die gleiche wie beim FDCS-REINFORCE-Ansatz. In einer festen Umgebung wird der AMR ausgesetzt. Dort fährt er sukzessive durch die Umgebung, bis er an einem Hindernis anstößt, sich auf 200 mm einem Hindernis nähert oder 200 erfolgreiche Schritte durchführt. Danach wird eine Neuorientierung eingeleitet, bis beide Frontsensoren einen Freiraum von mindestens 2 m anzeigen. Nach 200 erfolgreichen Schritten wird wie in der vorherigen Simulation das Fahrzeug vor der Neuorientierung zufällig gedreht, damit der AMR in der Lernphase in möglichst viele Regionen der Umgebung gelangt.
Zu Beginn der Simulation enthält das Q-Netzwerk kein Neuron und die Politik enthält genau eine Fuzzy-Regel als Vorwissen:

$$\textbf{if}\ \ (s_0 \approx 5000) \wedge (s_3 \approx 5000)\ \textbf{then}\ (\omega(s) \approx 0) \wedge (v(s) = v_0).$$

Belohnungen

Die direkten Belohnungen wurden für die folgenden Experimente im Simulator genau so vergeben, wie in Abschnitt 6.3.2. Als direkte Belohnung eines Zustandsüberganges wurde die Differenz der Zustandsbewertungen herangezogen. Die Abstandsmaße beider Sensoren wurden dabei zu gleichen Teilen mit $c_0 = c_3 = 0.5$ verwendet.
Für einen Fehlversuch wurde ebenfalls eine finale Bestrafung r_{final} vergeben. Auf sie kann prinzipiell auch verzichtet werden. Durch reellwertige Belohnung von Zustandsübergängen werden genügend Informationen zum Erlernen der Q-Funktion gegeben. Es hat sich jedoch gezeigt, daß mit der finalen Bestrafung und dem TD-Lernen eine Erhöhung der Konvergenzgeschwindigkeit erreicht werden kann.

Simulationen

Wie beim FDCS-REINFORCE-Lernverfahren wird auch für das reellwertige modifizierte Q-Lernen der zeitliche Verlauf eines beispielhaften Lernvorganges betrachtet.
Für die Simulationen wurden als Lernparameter $\gamma = 0.99$, $\lambda = 0.19$, $\beta_\pi^1 = 0.0001$, $\beta_\pi^2 = 0.015$, $\beta_Q = 0.001$, $v_0 = 400\frac{mm}{s}$ und $p = 0.9999$ verwendet. Als direkte Belohnungen wurden Differenzen von Zustandsbewertungen verwendet und eine Kollision wurde zusätzlich mit $r_{final} = -100$ bestraft.
Für die Politik wurden ferner Gaußsche Aktivierungsfunktionen mit Aktivierungsradius $\sigma = 0.05$ verwendet. Der Eingaberaum wurde auf $[0,1]^2$ normiert und die Politik wurde mittels einer normalisierten Evaluierungsfunktion berechnet. Zur Adaption der Verbindungsstruktur wurden $\alpha_c = 0.99$ und $\theta = 0.3$ gewählt. Zum Entfalten des Politik-Netzwerkes wurde die distanzgetriebene Einfügestrategie mit $\delta = 0.15$ verwendet. Die maximale Neuronenanzahl der Politik war $N_{max} = 100$.

Ebenso wurden für die Q-Funktion Gaußsche Aktivierungsfunktionen mit Aktivierungsradius $\sigma = 0.3$ verwendet. Der Eingaberaum wurde auf $[0,1]^4$ normiert und die Q-Funktion wurde ebenfalls normalisiert ausgewertet. Zur Adaption der Verbindungsstruktur der Neuronen der Q-Funktion wurden die gleichen Parameter wie bei der Politik herangezogen. Und zur Entfaltung des Q-Netzes wurde die gemischte Neuroneneinfügestrategie mit $\lambda_{ins} = 50000$ und $\delta = 0.2$ verwendet. Die Resourcen wurden aus den TD-Fehlern mit $\gamma_{res} = 0.9$ gewonnen. Zur Zentrenadaption der Q-Funktion wurden schließlich $\varepsilon_{bmu} = 0.0001$ und $\varepsilon_{Nh} = 0.000001$ verwendet.
Die Simulation wurde mit einer Zeitschrittweite von $\Delta t = 0.5s$ durchgeführt.

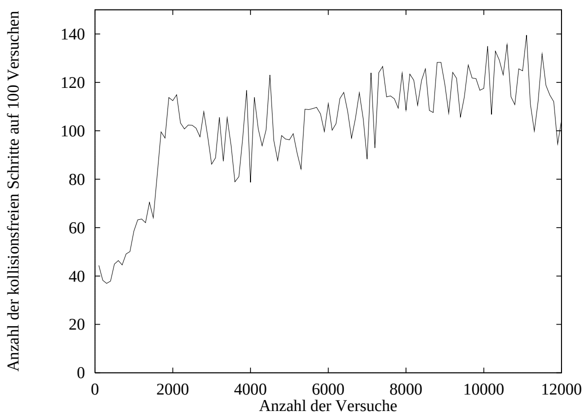


Abbildung 6.11: Schrittzahl

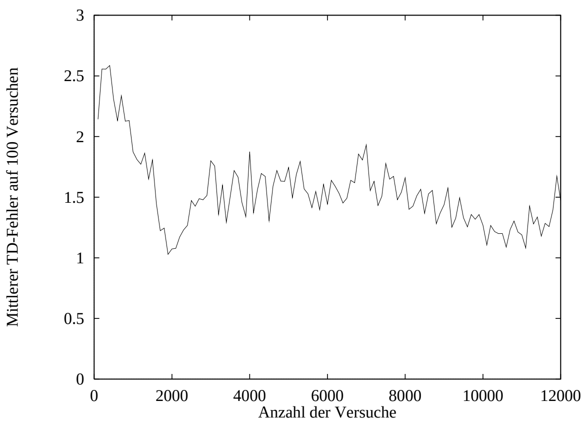


Abbildung 6.12: TD-Fehler

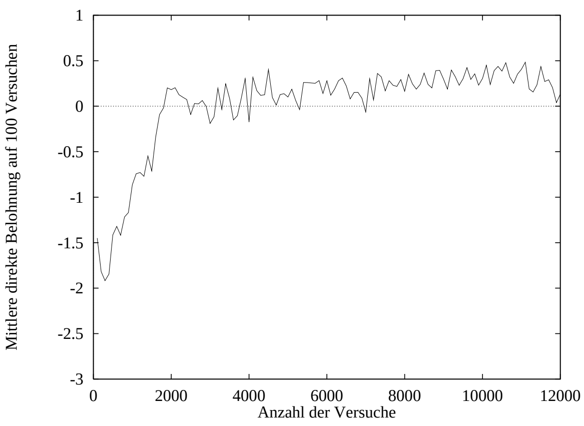


Abbildung 6.13: Belohnungen

Die Abbildungen 6.11 - 6.13 zeigen den zeitlichen Verlauf der Simulation für die angegebenen Parameter bei ideal simulierten Sensoren.
Obwohl bei diesen Experimenten der Eingaberaum der Politik nur zweidimensional ist (im Vergleich zu acht Dimensionen beim FDCS-REINFORCE-Ansatz) und ideal simulierte

Sensoren verwendet wurden, erlernt der AMR nur recht langsam ein Hindernisvermeidungsverhalten.
Offenbar ist der stochastische reellwertige Reinforcement-Ansatz einem Verfahren überlegen, welches explizit ein Prädiktionsmodell über den zukünftig zu erwartenden Erfolg erlernt und anhand dieses Prädiktionsmodells die Politik durch expliziten Gradientenaufstieg adaptiert.
Ein Grund für das schlechtere Abschneiden des reellwertigen modifizierten Q-Ansatzes im Vergleich zu einem REINFORCE-Ansatz könnten die hohen Anforderungen an die Qualität der Q-Funktion sein.
Um die Politik korrekt adaptieren zu können, muß nicht nur der betragsmäßige Fehler der Q-Funktion minimiert werden, sondern auch die erste Ableitung der Q-Funktion möglichst gut approximiert werden. Sind die Gradienteninformationen in der Q-Funktion falsch, so kann auch die Politik nicht korrekt adaptiert werden.

Kapitel 7

Diskussion und Ausblick

Zum Abschluß dieser Arbeit werden die erzielten Ergebnisse zusammengefaßt und bewertet. In einem darauffolgenden Ausblick werden Erweiterungsmöglichkeiten angegeben, die Gegenstand zukünftiger Arbeiten sein könnten.

7.1 Diskussion der vorgestellten Methoden

Der erste Ansatz dieser Arbeit für die ultraschallbasierte Hindernisvermeidung baut auf dem in [Mat92] angegebenen Verfahren auf. Dieses Verfahren ist an den *LABMATE* angepaßt und in wesentlichen Punkten verbessert worden.
In [Mat92] leistet die Hindernisvermeidung lediglich ein Wandfolgeverhalten. Die hier für den *LABMATE* entwickelte Hindernisvermeidung wurde zusätzlich um ein Zielfolgeverhalten ergänzt.
Da der *LABMATE* sich in einigen Punkten von dem in [Mat92] verwendeten AMR unterscheidet, mußte die Hindernisvermeidung stark an den *LABMATE* angepaßt werden. Die besondere Schwierigkeit besteht hier in der Größe des *LABMATE*, der fehlenden Radialsymmetrie und der geringen Anzahl von zur Verfügung stehenden Ultraschallsensoren.
Während die Hindernisvermeidung aus [Mat92] vor jedem Wechsel der Fahrtrichtung stoppt und eine Punktdrehung durchführt, wurde die Hindernisvermeidung für den *LABMATE* dahingehend erweitert, daß alle Änderungen der Fahrtrichtung während der Fahrt durchgeführt werden.
Hierbei stellte sich die Aufteilung in Verhaltensebenen bei [Mat92] als nicht zufriedenstellend heraus. Neben einem Notfallverhalten, das den Roboter durch eine Orientierung auch aus extrem engen Regionen befreit, wurde zur Vermeidung von Oszillationen ein weiteres Verhalten zum Passieren von schmalen Gängen hinzugefügt.
Mit dieser Architektur konnte für den realen Roboter eine funktionsfähige Hindernisvermeidung angegeben werden. Allerdings erwies sich die von [Mat92] übernommene starke Diskretisierung der sensorischen Eingaben und der motorischen Ausgaben als ein wesentlicher Nachteil, der Oszillationen im Fahrverhalten ermöglicht.
Erst durch eine feinere Diskretisierung und eine variable Geschwindigkeitskontrolle konnten diese Oszillationen erheblich reduziert werden. Während bei [Mat92] das Fahrzeug mit einer konstanten Geschwindigkeit von $0.2\frac{m}{s}$ fährt, wurden für den *LABMATE* Höchstge-

schwindigkeiten bis zu $0.6\frac{m}{s}$ erreicht.

Zur vollständigen Lösung der Navigationsaufgabe gehört neben der Hindernisvermeidung auch die Kartographierung und die Wegplanung. Beide Aspekte wurden in dieser Arbeit behandelt.
Die vorgestellte Methode zur Kartographierung orientiert sich an [CM93] und detektiert mit den Ultraschallsensoren Unstetigkeiten in der Umgebung. Diese markanten Weginformationen werden als Knoten in einen Wegegraphen eingefügt, durch den eine grobe Karte der Umgebung angegeben wird.
Die Verwendung eines Graphen zur Darstellung der Topologie der Umgebung hat dabei gegenüber Verfahren, die Hindernisgitter [BB+95] verwenden, wesentliche Vorteile. Graphen sind weniger speicherintensiv und sehr gut geeignet, die Wegplanungskomponente auf effiziente Weise einen Weg zur Erreichung eines bestimmten Ziels errechnen zu lassen. Durch die Graphdarstellung konnte somit der effiziente Dijkstra-Algorithmus zur Tourenoptimierung verwendet werden.
Durch die Verwendung einer reaktiven Hindernisvermeidung ist es weiterhin nicht notwendig, Informationen über die Lage von Hindernissen mit in die Karte aufzunehmen. Verfahren, deren Hindernisvermeidung die Karteninformationen ausnutzen, sind meistens deutlich aufwendiger als das hier vorgestellte reaktive System.

Ein grundsätzliches Problem, das sich bei der Kartographierung ergibt, ist die Selbstlokalisation. Dazu wurde in dieser Arbeit ein einfacher Ansatz vorgestellt, der auf der Grundannahme basiert, daß in der Umgebung nur Gänge vorkommen, die zueinander Vielfache eines bestimmten Winkels einschließen. Diese Annahme trifft für viele künstliche Umgebungen zu, in denen beispielsweise alle Gänge parallel oder rechtwinklig zueinander verlaufen. Ähnliche Voraussetzungen werden auch in [CM93] oder [BB+95] gefordert.
Forderungen an den Aufbau der Einsatzumgebungen zu stellen, hat den großen Nachteil, daß eine universelle Einsatzfähigkeit verloren geht. So konnte dieser Ansatz für den realen *LABMATE* in der zur Verfügung stehenden Umgebung nicht getestet werden. In dieser Umgebung treten Korridore auf, die Winkel von 70° und 90° einschließen. Der in dieser Arbeit vorgestellte histogrammbasierte Ansatz müßte dazu Winkelbereiche enthalten, die eine Größe von 10° besitzen. Diese Aufteilung ist aber zu fein, und ermöglicht keine eindeutige Identifizierung der vorherrschenden Richtung in den Histogrammen.
Daher konnte die Effektivität dieses Verfahrens nur im Simulator nachgewiesen werden.

Damit ist mit den bisherigen Komponenten bestehend aus reaktiver, verhaltensbasierter Hindernisvermeidung, Selbstlokalisation, Kartographierung und Wegplanung ein vollständiges Navigationssystem angegeben.

Die Resultate der reaktiven, verhaltensbasierten Hindernisvermeidung haben gezeigt, daß eine zu starke Diskretisierung von sensorischen Eingaben und motorischen Ausgaben die Qualität des Fahrverhaltens mindert und zu Oszillationen führt.
Für diese Problematik wird ein Regler benötigt, der in Abhängigkeit von kontinuierlichen Eingaben ebenfalls kontinuierliche Steuerungsgrößen berechnet. Dieses wird durch den Einsatz von Fuzzy-Reglern gewährleistet.

Außerdem sollte der Regler in der Lage sein, sich selbständig an Veränderungen der Randbedingungen anzupassen. Daher wurde ein kontinuierlicher Regler gesucht, der einerseits die Kodierung von Vorwissen wie ein Fuzzy-Regler zuläßt und sich andererseits wie ein neuronales Netzwerk durch Adaptivität auszeichnet.

Als Basis hierfür wurden Dynamische Zellstrukturen (DCS) verwendet. Die entscheidende Erweiterung der DCS-Netzwerke zu den FDCS-Netzwerken besteht dabei in dem Zulassen einer nicht vollständig verbundenen Ein- bzw. Ausgabeschicht. Dadurch ist die Kodierung von Fuzzy-Regeln möglich, die nur über Teilkomponenten des Ein- bzw. Ausgaberaumes Aussagen treffen.

Durch Erweiterung des Begriffs der perfekten Topologieerhaltung konnte in dieser Arbeit gezeigt werden, daß ebenfalls die FDCS-Netzwerke topologierepräsentierende neuronale Netzwerke darstellen.

Durch Hinzufügen einer Produkt- und einer Normalisierungsschicht zu den FDCS-Netzwerken wurde eine weitere Klasse von Netzwerken, die π-FDCS-Netzwerke, eingeführt. Mit diesen neuronalen Reglern ist es möglich, bestimmte Fuzzy-Regeln anzugeben, deren Ausgabe exakt gehalten werden, d. h. an die Stelle einer Approximation über die Ein- und Ausgabegewichte des Netzwerkes tritt die Interpolation. Damit sind Regeln zu formulieren, die einerseits genau eingehalten werden und andererseits auch durch weitere Lernschritte nicht verändert werden können. Somit können wichtige statische Regeln in ein ansonsten adaptives Netzwerk eingebunden werden.

Diese hier entwickelten adaptiven Fuzzy-Regler wurden in zwei Ansätzen auf die Hindernisvermeidung eines AMR's angewendet.
Im ersten Ansatz wurde versucht, die Architektur der verhaltensbasierten Hindernisvermeidung aufrecht zu erhalten, indem einzelne Verhaltensebenen durch einen adaptiven Fuzzy-Regler ersetzt wurden. Ein dabei auftretendes Problem ist die Auswahl der aktiven Verhaltensebene. Um eine glatte Regelung auch über mehrere Verhaltensebenen garantieren zu können, wurde dazu eine *weiche Auswahl* der aktiven Verhaltensebene eingeführt.
Ein Nachteil dieses Fuzzy-Ansatzes liegt darin, daß der angegebene Fuzzy-Regler auf symmetrische Eingabedaten auch mit symmetrischen Ausgaben reagiert. Dieses Verhalten führt insbesondere in Ecken der Umgebung zu Schwierigkeiten.
Um dieses Problem zu lösen und dennoch die Vorteile der glatten Fuzzy-Regelung aufrecht zu erhalten, wurde eine weitergehende Merkmalsextraktion aus Ultraschallmeßdaten entwickelt. Durch die Speicherung der Sensormessungen in der nahen Umgebung des Fahrzeuges erhält man eine lokale Umgebungsbeschreibung, aus der Merkmale, wie die Richtung eines Freiraumes, gewonnen werden. Insbesondere die Extraktion der Freiraumrichtung löst das Problem der symmetrischen Zustände und ermöglicht eine weitere Zusammenfassung der Verhaltensebenen.
Der zweite Fuzzy-Ansatz mit Merkmalsextraktion benötigt daher neben einer Notfallverhaltensebene nur noch einen einzigen FDCS-Regler zur Hindernisvermeidung.

Im letzten Teil der Arbeit steht die Adaptivität der zweiten Fuzzy-basierten Hindernisver-

meidung im Vordergrund.
Als Lernverfahren wurden dazu verschiedene Reinforcement-Lernansätze betrachtet. Da die zuvor entwickelten Regler kontinuierliche Ein- und Ausgaberäume besitzen, wurden spezielle reellwertige Reinforcement-Lernverfahren untersucht.
Das erste untersuchte Verfahren ist das stochastische reellwertige Reinforcement-Lernen [Wil92]. Es erlernt zusätzlich zur Politik eine adaptive Kritik, die zu einem Zustand eine Prädiktion des zukunftig erwarteten Erfolges liefert. Diese Kritik wird verwendet, um die Adaption der Politik so zu gestalten, daß nicht nur direkte Belohnungen maximiert werden, sondern stattdessen eine für die Zukunft zu erwartende akkumulierte Belohnung maximiert wird.
Die Besonderheit des dazu vorgestellten FDCS-REINFORCE-Lernverfahrens liegt darin, daß sowohl für die Kritik, als auch für die Politik nur ein einziges neuronales Netzwerk benötigt wird. Andere ähnliche Ansätze verwenden dagegen in der Regel zwei getrennte Netzwerke.
Die Verwendung des FDCS-Netzwerkes bietet hierbei den Vorteil, daß die Eingabeschicht des Netzwerkes für eine adaptive Vektorquantisierung sorgt. Dadurch werden die Neuronenzentren so im Eingaberaum verteilt, daß die darauf aufbauende Ausgabeschicht des FDCS-Netzwerkes sowohl für die Politik, als auch für die Kritik zu verwenden ist.

Dieser FDCS-REINFORCE-Ansatz wurde an die Hindernisvermeidung angepaßt, so daß insgesamt eine Hindernisvermeidungsarchitektur entwickelt wurde, die einerseits durch Fuzzy-Regeln kodiertes Vorwissen enthalten und andererseits dieses Vorwissen während der Lebenszeit des Roboters an sich ändernde Randbedingungen anpassen kann. In der Simulation wurde dazu gezeigt, daß der AMR sogar ohne jegliches Vorwissen in der Lage ist, ein hindernisvermeidendes Verhalten selbständig zu erlernen.

Ferner wurde noch ein weiterer Ansatz zum reellwertigen Reinforcement-Lernen angegeben, der das klassische Q-Lernen [Wat92] auf reellwertige Kontrollprobleme erweitert. Die notwendige Trennung zwischen Politik und Kritik stellte sich als ein wesentlicher Nachteil dieses Ansatzes heraus. Die separat erlernte Q-Funktion stellt ein Prädiktionsmodell des zukünftigen Ertrages in Abhängigkeit eines Zustandes und einer im folgenden auszuführenden Aktion dar. Aus diesem Prädiktionsmodell werden dann Gradienteninformationen zur Adaption der Politik benötigt.
Dieser Ansatz ist zwar in der Lage, ein adäquates Verhalten zu erlernen, jedoch überwiegen die Nachteile gegenüber dem FDCS-REINFORCE-Ansatz. Der Berechnungsaufwand ist durch eine kompliziertere Gradientenberechnung aufwendiger. Außerdem sind die Anforderungen an die Qualität der Kritik sehr hoch. Um die Gradienteninformationen aus der Q-Funktion zur Adaption der Politik heranziehen zu können, ist es notwendig, daß die Kritik nicht nur einen betragsmäßigen Fehler minimiert, sondern daß zusätzlich die Informationen über die Richtungen des steilsten Anstiegs in der Kritik stimmen müssen, um die Politik korrekt zu adaptieren.
Ferner vergrößert sich die Dimension des Eingaberaumes der Q-Funktion, da die aktuellen Aktionen ebenfalls zur Eingabe gehören. Hohe Dimensionen reduzieren aber die Qualität des Prädiktionsmodells und mindern die Lernfähigkeit des Systems.

7.2 Ausblick auf mögliche Erweiterungen

Da für diese Arbeit zur Steuerung des *LABMATE* nur ein Rechner mit Intel 486 SX CPU mit 25 MHz zur Verfügung stand, konnten die FDCS-basierten Architekturen zur Hindernisvermeidung nicht auf dem realen Roboter, sondern nur im Simulator getestet werden. Der einfach zu implementierende verhaltensbasierte Ansatz zur Hindernisvermeidung erreicht auf dem Laptop eine Rechenzeit von ca. $0.2s$ für einen Kontrollzyklus. Bei der Verwendung eines oder mehrerer FDCS-Regler mit bis zu 100 neuronalen Einheiten steigt die Rechenzeit auf dem Laptop so an, daß eine Echtzeitfähigkeit der Hindernisvermeidung nicht mehr erreicht wird.
Bei einer Erweiterung der Hardware um beispielsweise eine Funkverbindung zwischen dem *LABMATE* und einer SUN Workstation hätten die hier nur im Simulator durchgeführten Experimente auch am realen Roboter getestet werden können, insbesondere ob die Lernfähigkeit der hier vorgestellten Verfahren auch auf einen realen Roboter zutrifft. In der Simulation des FDCS-REINFORCE-Lernverfahrens wurden die Ultraschallsensoren als aufgeweitete Ultraschallkegel mit einem überlagerten normalverteilten Rauschen mit einer Standardabweichung von $5cm$ simuliert.
Daß das Lernverfahren auch bei dieser realistischen Simulation ein Hindernisvermeidungsverhalten erlernt hat, spricht für die mögliche Übertragbarkeit auf den realen Roboter. Zu bedenken sind allerdings die relativ langen Lernzeiten, die die durch die Batterieladung begrenzte Operationszeit (ca. 4 Std.) deutlich übersteigen, sowie die Notwendigkeit von abwechslungsreichen Trainingsumgebungen.

Der zweite Fuzzy-basierte Ansatz nutzt eine Vorverarbeitung der Ultraschalldaten und verwendet insbesondere das Merkmal der Freiraumrichtung. Diese Freiraumrichtung könnte in zukünftigen Architekturen zur Hindernisvermeidung durch visuelle Methoden ermittelt werden. So könnte beispielsweise eine Stereokamera verwendet werden, um in einem Bürogebäude anhand der Kanten zwischen Wand und Boden die Richtung des Korridors zu ermitteln und diese als Freiraumrichtung zu nutzen.
Auf diese Weise kann das hier entwickelte System zu einem visuell basierten, lernfähigen und Fuzzy-geregelten Verfahren zur Hindernisvermeidung ausgebaut werden.

Eine andere mögliche Erweiterung betrifft die Kartographierung. Die in dem hier vorgestellten Verfahren verwendeten Wegmarken markieren lediglich markante Punkte in der Umgebung, die durch die Ultraschalldaten als Unstetigkeiten in der Umgebung erkannt werden.
Prinzipiell ist es möglich, in die Wegmarken weitere symbolische Informationen aufzunehmen und beispielsweise detektierte Wände, Korridore oder Türen in die Karte einzutragen. Solche Wegmarken werden in [Mat92] verwendet.
Das Problem der Selbstlokalisation ist mit Hilfe einer topologischen Karte einfach zu lösen, wenn die Möglichkeit besteht, bestimmte Stellen in der Umgebung eindeutig zu identifizieren. Nach einer Identifikation einer sogenannten Landmarke kann die Odometrie auf die mit der jeweiligen Landmarke assoziierten Position gesetzt werden. Die eindeutigen Identifikation hängt von der Art der Landmarken ab.
Als Landmarken können in der Umgebung angebrachte Infrarot- oder Funksender verwen-

det werden, mit denen das Fahrzeug beim Passieren kommuniziert.
Ein sehr vielversprechendes Gebiet ist die Landmarkenidentifikation mit visuellen Methoden. Mit Hilfe von visueller Objekterkennung kann ein mit einer Kamera ausgerüsteter AMR eindeutige Markierungen in seiner Umwelt (z. B. Türschilder in Bürogebäuden) erkennen und als Landmarken identifizieren.

Eine weitere Ergänzung für die Selbstlokalisation könnte ein neuronaler Lösungsansatz zur Rekalibrierung der Odometrie darstellen. Wird der Fehler in der Odometrie durch einen systematischen Fehler verursacht, so könnte ein neuronales Netzwerk dieses Fehlersignal in einem überwachten Trainingsvorgang erlernen. Zur Korrektur der Odometrie könnte der durch das Netzwerk geschätzte Fehler von den Odometriedaten abgezogen werden. Schwierigkeiten eines solchen Ansatzes ergeben nach wie vor die nicht systematischen Fehler, die entstehen, wenn beispielsweise das Fahrzeug mit einem Hindernis kollidiert und sich dadurch seine Lage ändert.

Das Hauptproblem der Selbstlokalisation ist die Bestimmung der Orientierung des Fahrzeuges. Wenn der Boden eine signifikante Textur besitzt, aus der die jeweilige Orientierung zu ermitteln ist, so könnte zur Orientierungsbestimmung ein auf dem AMR angebrachtes Stereokamerasystem verwendet werden. Eine solche Textur wird beispielsweise durch einen gekachelten Boden geliefert. Durch die Orientierungen der Kanten der Kacheln könnte somit auf die Orientierung des Fahrzeuges geschlossen werden. Restriktionen hinsichtlich der vorkommenden Fahrtrichtungen entfallen hierbei vollkommen.

In Anbetracht dieser Erweiterungsmöglichkeiten können die in dieser Arbeit vorgestellten Verfahren die Grundlage für zukünftige Projekte bilden.

Anhang A

Der Simulator

In diesem Anhang soll die Funktionsweise und die wichtigsten Fragen der Implementierung des für diese Arbeit erstellten Simulators für den *LABMATE* dargestellt werden.

Der Simulator für den *LABMATE* simuliert die Dynamik des Roboters bei frei wählbarer Diskretisierung der Zeit in einer beliebigen 2D-Umgebung. Ferner stellt er die wichtigsten Steuerkommandos des Roboters zur Verfügung, so daß bis auf wenige hardwarenahe Kommandos alle Befehle der Steuersoftware des *LABMATE* bereitgestellt werden. Da die Ultraschallsensoren im Normalfall auf der Roboterplattform in gleicher Höhe über dem Boden angebracht sind, genügt es, die Simulation zweidimensional durchzuführen. Die Umgebung kann durch zahlreiche Grafikprimitiva modelliert werden.

Die Modellierung der Umwelt ist insbesondere für die Simulation der Ultraschallsensorik wichtig. Ultraschallsensoren können in der Simulation an beliebiger Position und in beliebiger Orientierung auf dem AMR angeordnet werden. Die Ultraschallkegel werden durch eine wählbare Anzahl von *Meßstrahlen* simuliert, die den Abstand zu einem Hindernis in der Umgebung ermitteln.

Der Simulator besteht aus zwei Komponenten. Die obere Komponente stellt die Schnittstelle zwischen Anwendungssoftware und der eigentlichen Simulation dar und orientiert sich an dem Befehlssatz der von TRC zur Verfügung gestellten Kommunikationssoftware. Die untere Komponente besteht aus der eigentlichen Simulation, in der die Wechselwirkung des AMR's mit der 2D-Umwelt simuliert wird. Sie ersetzt den realen AMR. Durch diese Zweiteilung ist es möglich, die Anwendungssoftware des *LABMATE* sowohl für den realen Roboter als auch für den Simulator zu verwenden. Dazu genügt ein einfacher Austausch der Schnittstellenkomponenten der TRC-Kommunikationssoftware und der Simulatorschnittstelle unter Beibehaltung der Anwendungssoftware (vgl. Abbildung A.1).

A.1 Die Schnittstelle des Simulators

Die Schnittstelle des Simulators ist vollständig zur Schnittstelle der TRC-Software für den *LABMATE* und für die Ultraschallsensorik kompatibel. Die vom Simulator unterstützten Kommandos sind: (vgl. [Tra91a], [Tra91b])

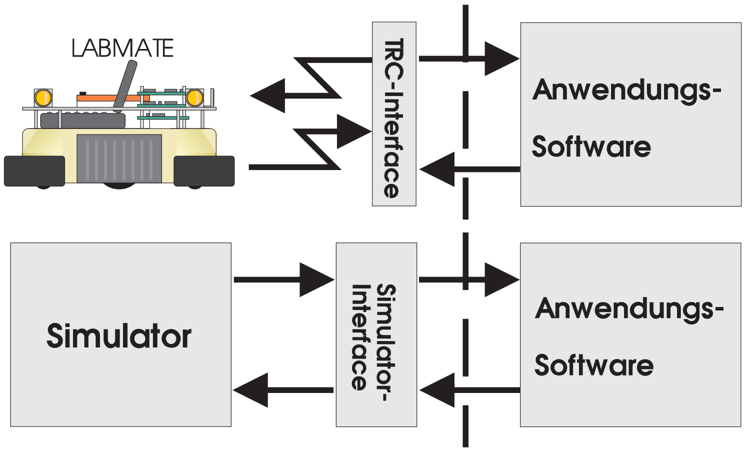


Abbildung A.1: Schnittstellen des Simulators und der TRC-Software

> *Reset LABMATE, Joystick Mode, Go Mode, Absolute Turn, Relative Turn, Point to Point Go, Point to Point Turn Absolute, Point to Point Turn Relative, Jog, Set Velocity, Set Acceleration, Clear Wheel Positions, Set Watchdog Timeout Value, Set Position, Set Bumper Mask, Get Report, Read Velocity, Read Status, Read Bumpers, Emergency Stop, Pause, Resume, Reset Subsystem, Enable/Disable Primary Sensors, Set Timeout Distance, Read Sensors*

Nicht unterstützte Kommandos sind:

> *Proportional Go Mode, Enable Encoder Heading, Enable Gyro Heading, Set Turn Constant, Download Joystick Lookup Table, Set Thresholds, Read Wheel Positions, Read A/D Ports, Enable/Disable Secondary Sensors, Set Partners, Define Infrared Verification, Enable Infrared Verification, Disable Infrared Verification*

Es handelt sich bei den nicht unterstützten Kommandos in erster Linie um hardwarenahe Kommandos, die im Simulator unnötig sind. Die Simulation von Infrarotsensoren ist ferner nicht implementiert. Um die Schnittstelle voll kompatibel zu gestalten, wurden die nicht unterstützten Kommandos durch *leere* Prozeduren dargestellt, die im Simulator keine Wirkung haben.
Die wesentliche Aufgabe der Schnittstelle besteht darin, durch Aufruf von der Anwendungssoftware die Parameter des Simulators so einzustellen, daß dieser ein Verhalten des Roboters simuliert, wie es ein vergleichbarer Aufruf der TRC-Schnittstelle für den *LABMATE* zur Folge hat.

A.2 Die Simulation

Die eigentliche Simulation besteht aus zwei Hauptkomponenten. Die erste Komponente betrifft die Simulation der *räumlichen Gegebenheiten*, also die Darstellung des Roboters und der Umwelt und die Überprüfung von Kollisionen zwischen Roboter und modellierten Gegenständen in der simulierten Umwelt. Ferner kann hier auch die Reflektion von

Ultraschallwellen an Gegenständen der Umwelt genannt werden. Die Hauptprobleme, die sich dabei ergeben, sind die *Modellierung von Gegenständen* in der 2D-Umwelt und die *Überprüfung von Kollisionen* von zwei Gegenständen.
Die zweite Komponente betrifft die Simulation der *Dynamik* des Roboters. Damit ist die möglichst naturgetreue Nachbildung des Bewegungsverlaufs des Roboters in der Umwelt gemeint. Durch Diskretisierung der Zeitachse ist es möglich, die in [Tra91a] angegebenen Zeit-Geschwindigkeits-Diagramme als Vorlage für die simulierte Bewegung heranzuziehen.

A.2.1 Umweltmodellierung

Der Simulator verwaltet folgende Arten verschiedener Objekte: *Roboter*, *vordere und hintere Stoßstange* des Roboters und die *Umwelt*. Jedes dieser Objekte kann durch eine Anzahl von geometrischen Elementen zusammengesetzt werden. Unterstützte Elemente sind dabei: *Strecken, Rechtecke, Ellipsensegmente, Ellipsen* und *Kreise*.
Dabei kann zwischen *virtuellen (virtual)* und *gegenständlichen (solid) Elementen* unterschieden werden. Die virtuellen Elemente dienen nur zur optischen Verfeinerung von Objekten hinsichtlich ihrer grafischen Darstellung. Gegenständliche Objekte werden dagegen als reale Gegenstände in der Umwelt modelliert und müssen daher auch auf Kollisionen überprüft werden. Jedes Objekt wird bei der Initialisierung des Simulators aus einer Startdatei eingelesen, wobei jedes Element gemäß der Syntax in Tabelle A.1 beschrieben wird.

Syntax in Simulatorinitialisierungsdatei	Beschreibung
`circle(`*xpos*, *ypos*, *rad*, *step*, *fill*, *attrib*`);`	Kreis
`ellipse(`*xpos*, *ypos*, *xrad*, *yrad*, *step*, *fill*, *attrib*`);`	Ellipse
`arc(`*xpos*, *ypos*, *xrad*, *yrad*, *sang*, *eang*, *step*, *fill*, *attrib*`);`	Ellipsensegment
`rectangle(`*xpos*1, *ypos*1, *xpos*2, *ypos*2, *fill*, *attrib*`);`	Rechteck
`line(`*xpos*1, *ypos*1, *xpos*2, *ypos*2, *fill*, *attrib*`);`	Strecke

Tabelle A.1: Syntax der Objektbeschreibungen

Der Metabezeichner *attrib* bezeichnet eines der Attribute `SOLID` oder `VIRTUAL`. *fill* kann entweder `0` oder `1` annehmen und gibt an, ob das Objekt gefüllt dargestellt werden soll. *step* gibt an, über welchen Winkelbereich ein Kreis- oder Ellipsensegment durch Polygone approximiert werden soll. *xpos*, *ypos*, *xpos*1, *ypos*1, *xpos*2 und *ypos*2 bezeichnen x- und y-Positionen. *rad*, *xrad* und *yrad* geben Radien für Kreise und Ellipsen an.

Jedes Objekt wird durch beliebig viele geometrische Elemente beschrieben, die in der Initialisierungsdatei innerhalb eines `DEFINE`-Blocks angegeben werden:

`DEFINE` *object name*
  Objektbeschreibung durch geometrische Elemente...
`END`

Als Objektname können dabei `ROBOT`, `FRONTBUMPER`, `REARBUMPER` und `ENVIRONMENT` angegeben werden.
Jedes Objekt wird intern durch eine verkettete Liste von Datensätzen repräsentiert, wobei jeder Datensatz die charakteristischen Daten eines geometrischen Elementes speichert.

A.2.2 Detektion von Kollisionen

Hauptproblem der Modellierung der Umwelt ist die Feststellung, ob der Roboter mit einem Objekt der Umwelt zusammengestoßen ist. Jedes Objekt besteht aus einer Anzahl von geometrischen Elementen, die ihrerseits intern durch eine Anzahl von Liniensegmenten zusammengesetzt sind. Eine Kollision zweier Objekte kann man also leicht ermitteln, indem man feststellt, ob zwei Liniensegmente unterschiedlicher Objekte einen gemeinsamen Schnittpunkt besitzen. Da der Roboter und insbesondere die Umwelt aus vielen Liniensegmenten bestehen kann, muß diese Schnittpunktüberprüfung effizient durchgeführt werden.
Dazu kann zu jedem geometrischen Element und jedem Objekt ein umschreibendes Rechteck definiert werden, das gerade so groß ist, daß die jeweiligen Elemente bzw. Objekte genau innerhalb dieses Rechtecks liegen. Bevor dann die Schnittpunkte eines Liniensegmentes mit anderen Objekten bzw. geometrischen Elementen ermittelt werden, wird untersucht, ob das Liniensegment überhaupt das umschriebene Rechteck berührt. Ist das nicht der Fall, so werden keine weiteren Schnittpunktberechnungen für das betreffende Objekt benötigt. Diese Vorgehensweise spart für sehr viele Geradensegmente eine aufwendige Schnittpunktberechnung und steigert die Effizienz erheblich.

Zur *Schnittpunktberechnung* zweier Geradensegmente kann zunächst ein Schnittpunkt der durch die Geradensegmente definierten Geraden ermittelt werden. Dazu seien zwei Geradensegmente durch ihre Randpunkte definiert.
Seien x_{11}, x_{12}, x_{21}, x_{22}, y_{11}, y_{12}, y_{21}, $y_{22} \in I\!R$. Dann sind $p_1 := (x_{11}, y_{11})$, $p_2 := (x_{12}, y_{12})$, $q_1 := (x_{21}, y_{21})$ und $q_2 := (x_{22}, y_2)$ Punkte im $I\!R^2$. Falls $p_1 \neq p_2$ und $q_1 \neq q_2$, so sind

$$G_1 := (p_1, p_2) := \{x \in I\!R^2 | x = p_1 + \lambda \cdot (p_2 - p_1), \lambda \in I\!R\}$$

und

$$G_2 := (q_1, q_2) := \{x \in I\!R^2 | x = q_1 + \mu \cdot (q_2 - q_1), \mu \in I\!R\}$$

zwei Geraden durch p_1 und p_2 bzw. durch q_1 und q_2. Zur Berechnung eines Schnittpunktes $s \in (p_1, p_2) \cap (q_1, q_2)$ kann folgende Gleichung betrachtet werden:

$$p_1 + \lambda \cdot (p_2 - p_1) = q_1 + \mu \cdot (q_2 - q_1), \quad \lambda, \mu \in I\!R. \tag{A.1}$$

Daraus erhält man folgendes lineares Gleichungssystem:

$$\begin{pmatrix} x_{11} - x_{21} \\ y_{11} - y_{21} \end{pmatrix} = \begin{pmatrix} x_{11} - x_{12} & x_{22} - x_{21} \\ y_{11} - y_{12} & y_{22} - x_{21} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix}. \tag{A.2}$$

Mit $A := \begin{pmatrix} x_{11} - x_{12} & x_{22} - x_{21} \\ y_{11} - y_{12} & y_{22} - x_{21} \end{pmatrix}$ gilt:

$$\det A = (x_{11} - x_{12})(y_{22} - y_{21}) - (x_{22} - x_{21})(y_{11} - y_{12}). \tag{A.3}$$

Durch Anwendung der Cramerschen Regel erhält man somit im Falle $\det A \neq 0$

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} y_{22} - y_{21} & x_{21} - x_{22} \\ y_{12} - y_{11} & x_{11} - x_{12} \end{pmatrix} \tag{A.4}$$

Damit erhält man:

$$\lambda = \frac{(y_{22} - y_{21})(x_{11} - x_{21}) + (x_{21} - x_{22})(y_{11} - y_{21})}{(x_{11} - x_{12})(y_{22} - y_{21}) - (x_{22} - x_{21})(y_{11} - y_{12})}. \tag{A.5}$$

Einsetzen von Gleichung (A.5) in Gleichung (A.1) ergibt als Schnittpunkt:

$$s = \begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} x_{11} + \lambda \cdot (x_{12} - x_{11}) \\ y_{11} + \lambda \cdot (y_{12} - y_{11}) \end{pmatrix}. \tag{A.6}$$

Ist $\det A = 0$, so ist das Gleichungssystem (A.2) nicht eindeutig lösbar und man erhält keinen eindeutigen Schnittpunkt, d. h. die Geraden $[p_1, p_2]$ und $[q_1, q_2]$ sind parallel. Da $q_1 \neq q_2$ und $p_1 \neq p_2$ kann dieser Fall anhand der Gleichung

$$(x_{11} - x_{12})(y_{22} - y_{21}) = (x_{22} - x_{21})(y_{11} - y_{12}) \tag{A.7}$$

behandelt werden. Sind alle vier Faktoren ungleich Null, dann sind die Geraden G_1 und G_2 gleich, da mit $\xi := \frac{x_{11} - x_{22}}{x_{22} - x_{21}}$ gilt:

$$p_1 = q_1 + \xi \cdot (q_2 - q_1) \in (q_1, q_2). \tag{A.8}$$

Genau einer oder genau drei Faktoren können nicht gleich Null sein. Ebenso können nicht alle Faktoren gleich Null sein, da $p_1 \neq p_2$ und $q_1 \neq q_2$ vorausgesetzt ist. Also verbleibt nur noch der Fall, daß auf jeder Seite der Gleichung (A.7) genau ein Faktor gleich Null ist. Wegen $p_1 \neq p_2$ und $q_1 \neq q_2$ existieren dann nur noch die folgenden beiden Fälle:

$$\begin{aligned} &a) \quad x_{11} = x_{12} \wedge x_{22} = x_{21} \\ &b) \quad y_{11} = y_{12} \wedge y_{22} = y_{21} \end{aligned}$$

Im Fall a) sind die Geraden G_1 und G_2 genau dann gleich, falls $x_{11} = x_{22}$ gilt und im Fall b) sind G_1 und G_2 genau dann gleich, falls $y_{11} = y_{22}$ gilt. Damit ist vollständig geklärt, ob die Geraden G_1 und G_2 keinen Schnittpunkt besitzen, genau einen Schnittpunkt besitzen oder sogar schon gleich sind.

Nachdem $G_1 \cap G_2$ bestimmt wurde, kann nun ermittelt werden, ob die beiden Geradensegmente

$$[p_1, p_2] := \{x \in \mathbb{R}^2 | x = \lambda \cdot p_1 + (1 - \lambda) \cdot p_2, \lambda \in [0, 1]\}$$

und

$$[q_1, q_2] := \{x \in \mathbb{R}^2 | x = \mu \cdot p_1 + (1 - \mu) \cdot p_2, \mu \in [0, 1]\}$$

einen nicht leeren Schnitt besitzen. Ist $\det A \neq 0$, so ist $(p_1, p_2) \cap (q_1, q_2) \neq \emptyset$ genau dann, wenn s innerhalb der beiden p_1 und p_2 bzw. q_1 und q_2 umschreibenden Rechtecke liegt. Ebenso kann im Falle $G_1 = G_2$ ermittelt werden, ob die beiden umschreibenden Rechtecke sich überlappen. Ist dieses der Fall, so wird eine Kollision festgestellt.

A.2.3 Simulation der Dynamik

Die Simulation der Dynamik des Roboters setzt die Einführung einer Größe voraus, die die Zeit modelliert. Dazu wird ein Zeitzähler t verwendet, der Schrittweise in jedem Durchlauf eines Simulationszyklus um ein Δt erhöht wird. Auf diese Weise wird die Zeit diskretisiert. Wird ein Bewegungskommando an die Simulatorschnittstelle gesendet, so werden dort bestimmte Parameter festgelegt, die den Bewegungsverlauf des Roboters vorgeben. Als Beispiel dazu wird die Simulation eines *Point-to-Point-Go*-Kommandos im folgenden näher betrachtet. In diesem Fall legen die Parameter die Zeiträume von drei Bewegungsphasen fest.

1. **Beschleunigungsphase:** Durch Festlegung der Dauer Δt_a wird festgelegt, wie lange das Fahrzeug beschleunigt werden soll. In jedem Simulationszyklus wird in der Beschleunigungsphase die Geschwindigkeit v um $\Delta v = a \cdot \Delta t$ erhöht.

2. **Haltephase:** In dieser Phase wird die Geschwindigkeit über eine Zeitdauer Δt_h konstant gehalten, d. h. es wird nicht mehr beschleunigt.

3. **Verzögerungsphase:** Ist die Zeitdauer der 2. Phase in der Simulationszeit abgelaufen, so wird in der dritten Phase analog zur ersten Phase die Geschwindigkeit um Δv über eine Zeitdauer Δt_d verringert.

In jeder Phase wird eine neue Geschwindigkeit festgelegt. Mit Hilfe der aktuellen Geschwindigkeit v und der aktuellen Position und Orientierung des Roboters (x_t, y_t, α_t) wird durch

$$\begin{aligned}
x_{t+\Delta t} &= x_t + v \cdot \Delta t \cdot \cos\left(\frac{(\alpha_t + 90°) \cdot \pi}{180°}\right), \\
y_{t+\Delta t} &= y_t + v \cdot \Delta t \cdot \sin\left(\frac{(\alpha_t + 90°) \cdot \pi}{180°}\right), \\
\alpha_{t+\Delta t} &= \alpha_t,
\end{aligned} \tag{A.9}$$

eine neue Position und Orientierung im allozentrischen Koordinatensystem ermittelt. Durch die Diskretisierung der Zeitachse ist es möglich, daß die Zeitdauer einer Bewegungsphase kein Vielfaches von Δt ist. Eine sukzessive Erhöhung der Geschwindigkeit in einem Simulationszyklus kann dann dazu führen, daß am Ende eine größere Endgeschwindigkeit erreicht wurde, als durch Wahl der Zeitdauer einer Bewegungsphase beabsichtigt war. Damit dieser Fall nicht eintritt, muß die jeweilige beabsichtigte Endgeschwindigkeit immer mit der aus der jeweiligen Bewegungsphase resultierenden Geschwindigkeit verglichen werden und bei Überschreitung der beabsichtigten Endgeschwindigkeit durch diese ersetzt werden. Die Simulation einer solchen Bewegungsphase wird in Abbildung A.2 dargestellt. Geringe Abweichungen im zeitlichen Verlauf der Bewegungskurve sind aufgrund der Zeitdiskretisierung nicht zu verhindern. Ebenso wie der Verlauf eines *Point-to-Point-Go*-Kommandos hier durch Einteilung in mehrere Bewegungsphasen simuliert wird, kann dieses auch für alle anderen Bewegungskommandos realisiert werden. Auch die Überlagerung einer rotatorischen Bewegung kann durch sukzessive Veränderung der Orientierung α_t vorgenommen werden.

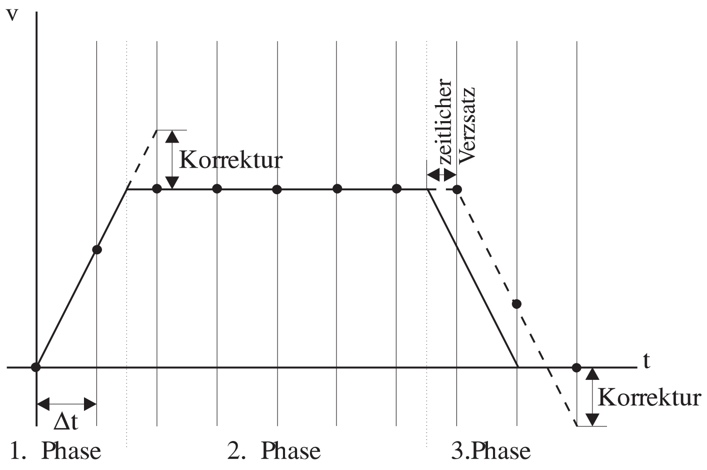


Abbildung A.2: Simulation der Bewegungsphasen

A.2.4 Simulation der Sensorik

Als letzter Bestandteil des Simulators verbleibt noch die Simulation eines Ultraschallsensors. Jeder Ultraschallsensor wird durch eine Anzahl von *Meßstrahlen* simuliert, die punktförmig, vom Ursprung des Sensors ausgehend einen Ultraschallkegel mit vorgegebenem Öffnungswinkel darstellen. Die Länge der Meßstrahlen entspricht genau der maximal meßbaren Entfernung eines Sensors, die durch den *Timeout*-Wert des Sensors angegeben wird. Für jeden Meßstrahl kann dann ein Schnittpunkt mit einem Geradensegment der Umgebung ermittelt werden. Die Entfernung eines Schnittpunktes mit minimalem Abstand zum Sensorursprung gibt den gemessenen Entfernungswert des Sensors an.
Ebenso wie Objekte durch einen `DEFINE`-Block definiert werden, werden alle simulierten Ultraschallsensoren definiert. Als Objektnamen wird der Bezeichner `SENSORS` verwendet.

`sensor(`$nr,xpos,ypos,\alpha,timeout,col,rad,\beta,num$ `);`

Die Parameter $xpos$, $ypos$ und α geben die Position und Orientierung eines Sensors im egozentrischen *LABMATE*-Koordinatensystem an. Der Wert von nr gibt die logische Nummer des Sensors an. Der Timeout-Wert des Sensors wird über $timeout$ festgelegt. β gibt den halben Öffnungswinkel der Sensorkegel an und num definiert die Anzahl der Meßstrahlen für einen Sensorkegel. Die Parameter col und rad geben schließlich die Farbe und den Radius des Kreises an, mit dem ein Sensor grafisch dargestellt wird.

Anhang B

Mathematische Ergänzungen

Dieser Anhang enthält mathematische Ergänzungen zu einigen Themen aus den Kapiteln 4 und 6.
Der Schwerpunkt dieses Anhangs besteht aus der Verallgemeinerung des Satzes von Martinetz für FDCS-Netzwerke. Die hier bewiesene verallgemeinerte Variante dieses Satzes zeigt, daß mittels des Hebb'schen Wettbewerbslernen auch in FDCS-Netzwerken eine Art topologierepräsentierende Karte erlernt wird.
Den Abschluß dieses Anhangs bildet der Nachweis über die Interpolationseigenschaft von π-FDCS-Netzwerken und die Berechnung von Gradienten in π-FDCS-Netzwerken.

B.1 Perfekte Topologieerhaltung in FDCS - Netzwerken

Während sich die Neuronenzentren der klassischen Neuronen von DCS-Netzwerken als Punkte im Eingaberaum auffassen lassen, trifft dies für die verallgemeinerten Neuronen in FDCS-Netzwerken nicht mehr zu. Diese lassen sich durch eine Menge von Punkten veranschaulichen, die bzgl. des Abstandsmaßes aus (4.47) einen Abstand 0 zum Neuronenzentrum besitzen. Somit erhält das Zentrum eines verallgemeinerten Neurons $u \in U$ die Repräsentation

$$R(u) = \{x \in I\!R^n | \forall i \in \{1, \cdots, n\} : \quad (c_u)_i \neq \perp \Rightarrow x_i = (c_u)_i\}, \tag{B.1}$$

d. h. ein verallgemeinertes Neuron läßt sich als affiner Teilraum des $I\!R^n$ auffassen.

Sei $k \in I\!N$. Mit

$$T^k = \{V | V \leq I\!R^n \wedge \dim V = k\} \tag{B.2}$$

wird die Menge aller *k-dimensionalen Teilräume* von $I\!R^n$ bezeichnet.

$$A^k = \left\{V + v | V \in T^k \wedge v \in I\!R^n\right\} \tag{B.3}$$

gibt die Menge aller *k-dimensionalen affinen Teilräume* von $I\!R^n$ an und durch

$$A^* = \bigcup_{k=0}^{n} A^k \tag{B.4}$$

werden alle affinen Teilräume von $I\!R^n$ definiert. Für Repräsentationen von Zentren von Neuronen $u \in U$ gilt dann $R(u) \in A^*$. Als verallgemeinertes Abstandsmaß zwischen Neuronenzentren wird die folgende Abbildung definiert

$$d : A^* \times A^* \to I\!R_{\geq 0}, \quad (V, W) \mapsto \inf\{\|v - w\| | v \in V \wedge w \in W\}, \tag{B.5}$$

die hier als *Pseudometrik* bezeichnet wird.

Bemerkung B.1 (Eigenschaften der Pseudometrik)
Seien $V, W \in A^*$. Dann gilt:

$$a) \quad d(V, W) = d(W, V), \tag{B.6}$$

$$b) \quad d \cap \left(T^0 \times T^0 \times I\!R_{\geq 0}\right) \quad \textit{ist Metrik auf } T^0. \tag{B.7}$$

Beweis:
Zu a): Die Aussage ist klar, da für alle $v \in V$ und $w \in W$ $\|v - w\| = \|w - v\|$ gilt.
Zu b): Für alle $\{v\}, \{w\} \in T^0$ gilt $d(\{v\}, \{w\}) = \|v - w\|$ und somit die Behauptung. □

Zu bemerken ist ferner, daß d im allgemeinen nicht die Dreiecksungleichung erfüllt und somit keine Metrik darstellt. Dennoch ist d nach Aussage (B.1) b) eine Fortsetzung der Euklidischen Metrik auf A^*. Für $v \in I\!R^n$ und $W \in A^*$ wird im folgenden statt $d(\{v\}, W)$ auch kurz $d(v, W)$ geschrieben. Mit dieser Pseudometrik läßt sich nun der Begriff der Voronoi-Zellen auch auf A^* verallgemeinern.

Definition B.1 (Verallgemeinerte Voronoi-Zelle)
Sei $N \in I\!N$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. Dann heißt

$$V_i = \{c \in I\!R^n | \forall j \in \{1, \cdots, N\} : \quad d(c, c_i) \leq d(c, c_j)\} \tag{B.8}$$

die zu $c_i \in S$ gehörige verallgemeinerte Voronoi-Zelle.

Eine Voronoi-Zelle eines Zentrums c_j enthält alle diejenigen Punkte des Raumes, zu denen c_j das bzgl. d nächste Zentrum bildet. Man kann zwei Neuronenzentren als benachbart ansehen, wenn ihre zugehörigen Voronoi-Zellen im Eingaberaum benachbart sind. Diese Nachbarschaftsaussage wird durch einen Graphen festgelegt, der in der folgenden Definition angegeben ist.

Definition B.2 (Verallgemeinerte Delaunay-Triangulation)
Sei $N \in I\!N$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. Der durch

$$\mathcal{D}_S = (\{1, \cdots, N\}, \{\{i, j\} | i \neq j \wedge V_i \cap V_j \neq \emptyset\}) \tag{B.9}$$

festgelegte ungerichtete Graph heißt die verallgemeinerte Delaunay-Triangulation *zu S.*

Definition B.3 (Verallgemeinerte Voronoi-Zelle zweiter Ordnung)
Sei $N \in I\!N$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. Das durch

$$V_{ij} = \{v \in I\!R^n | \forall k \in \{1, \cdots, N\} \setminus \{i, j\} : d(v, c_i) \leq d(v, c_k) \wedge d(v, c_j) \leq d(v, c_k)\} \tag{B.10}$$

definierte Gebiet heißt verallgemeinerte Voronoi-Zelle zweiter Ordnung *zu c_j und c_i.*

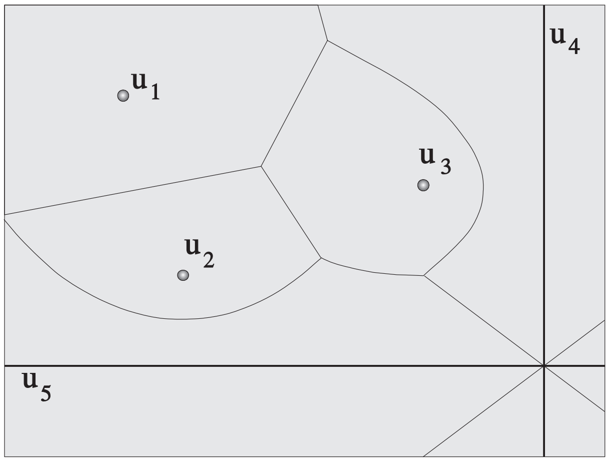


Abbildung B.1: Verallgemeinertes Voronoi-Diagramm für fünf Neuronen in einem zweidimensionalen Eingaberaum: Die drei Neuronen u_1, u_2 und u_3 sind vollständig mit der Eingabeschicht verbunden und als Punkte dargestellt. Die anderen beiden Neuronen u_4 und u_5 sind jeweils nur mit einer Eingabekomponente verbunden und werden als horizontale bzw. vertikale Geraden repräsentiert. Die dargestellten Umrandungen kennzeichnen die zu den Neuronen gehörenden verallgemeinerten Voronoi-Zellen.

Da der Eingaberaum I, aus dem die Eingaben des Netzwerkes stammen, selten gleich dem gesamten Raum $\mathbb{R}^n$ ist, führt Martinetz in [MS93] den Begriff der *maskierten Voronoi-Zellen* ein, um den Nachbarschaftsbegriff zwischen Neuronenzentren auch in allgemeinen Eingaberäumen I zur Verfügung zu haben. In dem hier betrachteten verallgemeinerten Rahmen definiert man:

Definition B.4 (Maskierungen)
Sei $I \subseteq \mathbb{R}^n$, $N \in \mathbb{N}$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. $V_j^{(I)} = V_i \cap I$ heißt verallgemeinerte maskierte Voronoi-Zelle *zu c_i. $V_{ij}^{(I)} = V_{ij} \cap I$ heißt* verallgemeinerte maskierte Voronoi-Zelle zweiter Ordnung *zu c_i und c_j. Der Graph*

$$\mathcal{D}_S^{(I)} = \left(\{1, \cdots, N\}, \left\{\{i,j\} | i \neq j \wedge V_i^{(I)} \cap V_j^{(I)} \neq \emptyset\right\}\right) \tag{B.11}$$

heißt verallgemeinerte induzierte Delaunay-Triangulation *zu S und ist ein Teilgraph von $\mathcal{D}_S$*

Zwei Neuronen sind im Eingaberaum benachbart, wenn ihre verallgemeinerten maskierten Voronoi-Zellen einen nichtleeren Schnitt besitzen. Ein Graph wird durch folgende Definition als perfekt topologieerhaltende Karte bezeichnet, wenn er genau diese Nachbarschaftsbeziehung zwischen Neuronen zum Ausdruck bringt.

Definition B.5 (Perfekt topologieerhaltende Karten)
Sei $I \subseteq \mathbb{R}^n$, $N \in \mathbb{N}$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. Sei $G = (V_G, E_G)$ ein ungerichteter Graph mit Knotenmenge $V_G = \{1, \cdots, N\}$. Eine Abbildung

$$\Phi_S : I \to V_G, x \mapsto i^*(x), \tag{B.12}$$

wobei $i^*(x)$ *ein Knoten von* G *mit* $x \in V_{i^*(x)}^{(I)}$ *bezeichnet, heißt* nachbarschaftserhaltend, *falls für alle* $i, j \in \{1, \cdots, N\}$ *gilt:*

$$V_i^{(I)} \cap V_j^{(I)} \neq \emptyset \Rightarrow \forall x \in c_i \cap I, y \in c_j \cap I : \{\Phi_S(x), \Phi_S(y)\} \in E_G. \tag{B.13}$$

Die dazu entgegengesetzte Abbildung

$$\tilde{\Phi}_S : V_G \rightarrow A^*, i \mapsto c_i \tag{B.14}$$

heißt nachbarschaftserhaltend, *falls gilt:*

$$\forall i, j \in \{1, \cdots, N\} : \quad \{i, j\} \in E_G \Rightarrow V_i^{(I)} \cap V_j^{(I)} \neq \emptyset. \tag{B.15}$$

Dann heißt G perfekt topologieerhaltende Karte von S auf I *genau dann, wenn* Φ_S *und* $\tilde{\Phi}_S$ *nachbarschaftserhaltend sind.*

Ein Resultat, welches in [MS93] bewiesen wurde, läßt sich auch in der hier angegebenen allgemeineren Fassung aussprechen. Der Beweis läßt sich dabei direkt übertragen.

Satz B.1 (Delaunay-Triangulation & perfekt topologieerhaltende Karten)
Sei $I \subseteq I\!R^n$, $N \in I\!N$ *und* $S = \{c_1, \cdots, c_N\} \subseteq A^*$. *Ferner sei* G *ein ungerichteter Graph mit Knotenmenge* $V_G = \{1, \cdots, N\}$. *Dann ist* G *eine perfekt topologieerhaltende Karte von* S *auf* I *genau dann, wenn* $G = \mathcal{D}_S^{(I)}$ *gilt.* □

Das Hauptziel ist nun der Nachweis, daß mit dem Hebb'schen Wettbewerbslernen aus (4.10) und der dadurch implizit definierten Menge L von Neuronenverbindungen eine perfekt topologieerhaltende Karte erlernt wird. Damit sind auch FDCS-Netzwerke in dem hier dargestellten verallgemeinerten Rahmen topologierepräsentierende Netzwerke (TRN).
Betrachtet man im FDCS-Netzwerk zwei Neuronen als verbunden, wenn ihre Verbindungsstärke größer als 0 ist, so werden durch das Hebb'sche Wettbewerbslernen zwei Neuronen mit Zentren c_i und c_j verbunden, wenn im Laufe der Trainingsphase eine Eingabe in die Voronoi-Zelle zweiter Ordnung V_{ij} fällt. Setzt man den Eingaberaum als $I = \{x \in I\!R^n | P(x) > 0\}$, wobei P die Dichtefunktion der Verteilung der Eingaben ist, so werden zwei Neuronen durch das Hebb'sche Wettbewerbslernen verbunden, falls $V_{ij} \cap I = V_{ij}^{(I)} \neq \emptyset$ gilt.
Zum Nachweis, daß das Hebb'sche Wettbewerbslernen eine perfekt topologieerhaltende Karte erzeugt, benötigt Martinetz in [MS93] den Begriff der dichten Verteilung der Neuronenzentren im Eingaberaum. Diese Forderung gewährleistet, daß zu jeder Eingabe das aus der Eingabe und den beiden am besten passenden Neuronen gebildete Dreieck vollständig im Eingaberaum enthalten ist. Da in dem hier betrachteten verallgemeinerten Rahmen Neuronenzentren keine Punkte, sondern affine Teilräume darstellen, liegt hier die einzige Schwierigkeit. Auch der Begriff der dichten Verteilung muß daher verallgemeinert werden.

Definition B.6 (Dreieck)
Seien $x, y, z \in I\!R^n$.

$$\Delta(x, y, z) = [x, y] \cup [y, z] \cup [z, x] \tag{B.16}$$

heißt das durch x, y *und* z *aufgespannte* Dreieck, *wobei* $[x, y] = \{x + t(y - x) | t \in [0, 1]\}$.

Eine Menge S von Neuronenzentren ist *dicht* auf einem Eingaberaum I, wenn zu jeder Eingabe $x \in I$ und den zu x am besten passenden Neuronenzentren c_{i_0} und c_{i_1} Punkte $\xi \in c_{i_0}$, $\eta \in c_{i_1}$ in diesen Neuronenzentren existieren, so daß das durch x, ξ und η aufgespannte Dreieck in I enthalten ist und der Abstand von x zu den beiden Neuronenzentren durch den Abstand von x zu ξ und zu η angegeben wird. Anschaulich stellen ξ und η die Projektionen von x auf die Neuronenzentren dar. Formal definiert man:

Definition B.7 (Dichte Zentrenverteilung)
Sei $I \subseteq \mathbb{R}^n$, $N \in \mathbb{N}$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$. Die Verteilung der Zentren aus S heißt dicht *auf I, falls für alle $x \in I$ gilt:*

$$\exists \xi \in c_{i_0}, \eta \in c_{i_1} : \quad \Delta(x, \xi, \eta) \subseteq I \wedge d(x, \xi) = d(x, c_{i_0}) \wedge d(x, \eta) = d(x, c_{i_1}), \tag{B.17}$$

wobei c_{i_0} und c_{i_1} die beiden bzgl. d nächsten Zentren zu x bezeichnen.

Für den Beweis des folgenden Satzes wird schließlich die Hilfsaussage der folgenden Bemerkung benötigt.

Bemerkung B.2
Seien $K_1, K_2 \subseteq \mathbb{R}^n$ kompakte Mengen und $f : K_1 \times K_2 \to \mathbb{R}$ eine stetige Abbildung. Dann ist die Abbildung

$$g : K_1 \to \mathbb{R}, \quad x \mapsto \min\{f(x, y) | y \in K_2\} \tag{B.18}$$

ebenfalls stetig.

Beweis:
<u>Annahme:</u> g ist unstetig.
Dann existiert eine Unstetigkeitsstelle $x_0 \in K_1$ und nach dem ε-δ-Kriterium ein $\varepsilon > 0$ mit

$$\forall \delta > 0 \quad \exists x_1 \in K_1 : \quad \|x_0 - x_1\| < \delta \wedge |g(x_0) - g(x_1)| > \varepsilon. \tag{B.19}$$

Da f stetig ist und $K_1 \times K_2$ kompakt ist, ist f bereits gleichmäßig stetig. Ferner existiert $y_0 \in K_2$, so daß $f(x_0, \cdot)$ bei y_0 ein Minimum annimmt. Dann gilt:

$$g(x_0) = f(x_0, y_0). \tag{B.20}$$

Da f gleichmäßig stetig ist, existiert $\delta_1 > 0$ mit

$$\forall (x, y), (\bar{x}, \bar{y}) \in K_1 \times K_2 : \quad \|(x, y) - (\bar{x}, \bar{y})\| < \delta_1 \Rightarrow |f(x, y) - f(\bar{x}, \bar{y})| < \varepsilon. \tag{B.21}$$

Setze $\delta_2 := \min\{\delta_0, \delta_1\}$
Nach (B.19) existiert $x_1 \in K_1$ mit

$$\|x_0 - x_1\| < \delta_2 \text{ und } |g(x_0) - g(x_1)| > \varepsilon. \tag{B.22}$$

Analog zur Wahl von y_0 wählt man $y_1 \in K_2$ mit

$$g(x_1) = f(x_1, y_1). \tag{B.23}$$

<u>1. Fall:</u> $g(x_0) < g(x_1)$.
Dann gilt

$$g(x_1) - g(x_0) > \varepsilon. \tag{B.24}$$

Da in der Euklidischen Norm $\|(x_0, y_0) - (x_1, y_0)\| = \|x_0 - x_1\| < \delta_2$ gilt, folgt aus (B.21)

$$|g(x_0) - f(x_1, y_0)| = |f(x_0, y_0) - f(x_1, y_0)| < \varepsilon. \tag{B.25}$$

Aus (B.24) und nach Wahl von y_0 und y_1 folgt

$$\varepsilon < g(x_1) - g(x_0) = f(x_1, y_1) - f(x_0, y_0) \leq f(x_1, y_0) - f(x_0, y_0) \leq |g(x_0) - f(x_1, y_0)|. \tag{B.26}$$

Dieses steht im Widerspruch zu (B.25).
<u>2. Fall:</u> $g(x_1) < g(x_0)$.
Dieser Fall verläuft durch Vertauschen von (x_0, y_0) und (x_1, y_1) analog zum 1. Fall. □

Damit sind die Grundlagen gelegt, um den zentralen Satz von Martinetz in der verallgemeinerten Fassung aussprechen und beweisen zu können.

Satz B.2 (Perfekte Topologieerhaltung durch Hebb'sches Wettbewerbslernen)
Sei $I \subseteq \mathbb{R}^n$ kompakt, $N \in \mathbb{N}$ und $S = \{c_1, \cdots, c_N\} \subseteq A^*$ eine Menge von Neuronenzentren. Ist S dicht auf I, dann ist der Graph G, der durch das Hebb'sche Wettbewerbslernen erzeugt wird (vgl. (4.10)), gleich der verallgemeinerten induzierten Delaunay-Triangulation $\mathcal{D}_S^{(I)}$ von S und ist somit nach Satz B.1 eine perfekt topologieerhaltende Karte von I.

Beweis:
Durch das Hebb'sche Wettbewerbslernen werden zwei Neuronen mit Zentren c_i und c_j verbunden, wenn $V_{ij}^{(I)} \neq \emptyset$ gilt. In $\mathcal{D}_S^{(I)}$ sind zwei Knoten i und j benachbart, wenn $V_i^{(I)} \cap V_j^{(I)} \neq \emptyset$ gilt. Daher genügt es also, die folgende Aussage zu zeigen:

$$\forall i, j \in \{1, \cdots, N\}, i \neq j : \quad V_i^{(I)} \cap V_j^{(I)} \neq \emptyset \Leftrightarrow V_{ij}^{(I)} \neq \emptyset. \tag{B.27}$$

Seien $i, j \in \{1, \cdots, N\}$ mit $i \neq j$.
<u>$\Rightarrow$:</u> Sei $x \in V_i^{(I)} \cap V_j^{(I)}$. Dann gilt $d(x, c_i) = d(x, c_j) \leq d(x, c_k)$ für alle $c_k \in S$. Damit gilt aber bereits $x \in V_{ij}^{(I)}$.
<u>$\Leftarrow$:</u> Es gelte $V_{ij}^{(I)} \neq \emptyset$. Dann existiert ein $v^* \in V_{ij}^{(I)}$. O. B. d. A. sei $d(c_i, v^*) \leq d(c_j, v^*)$. Da S dicht auf I ist, existieren $v_i \in c_i$ und $v_j \in c_j$ mit

$$\Delta(v^*, v_i, v_j) \subseteq I \wedge d(v^*, v_j) = d(v^*, c_j) \wedge d(v^*, v_i) = d(v^*, c_i). \tag{B.28}$$

<u>Z.z.:</u> $(*)$ Für alle $u \in [v^*, v_j]$ ist c_j nächstes oder zweitnächstes Neuronenzentrum zu u.
Also ist die Aussage

$$\forall u \in [v^*, v_j] : \quad \forall c_k \in S \setminus \{c_i, c_j\} : \quad d(u, c_k) \geq d(u, c_j) \tag{B.29}$$

zu zeigen. Sei dazu $u \in [v^*, v_j]$. Dann existiert $t \in [0, 1]$ mit $u = v^* + t(v_j - v^*)$. Sei $c_k \in S \setminus \{c_i, c_j\}$. Dann gilt $d(v^*, c_k) \geq d(v^*, c_j)$. Wegen $d(v^*, c_j) = d(v^*, v_j) = \|v^* - v_j\|$ gilt

$$d(v^*, c_k) \geq \|v^* - v_j\|. \tag{B.30}$$

Damit gilt

$$
\begin{aligned}
d(u, c_k) &= \inf \{ \|u - \eta\| \mid \eta \in c_k \} \\
&= \inf \{ \|v^* + t(v_j - v^*) - \eta\| \mid \eta \in c_k \} \\
&= \inf \{ \|(v^* - \eta) - t(v^* - v_j)\| \mid \eta \in c_k \} \\
&\geq \inf \{ \|v^* - \eta\| - t\|v^* - v_j\| \mid \eta \in c_k \} \\
&= \inf \{ \|v^* - \eta\| \mid \eta \in c_k \} - t\|v^* - v_j\| \qquad \text{(B.31)} \\
&= d(v^*, c_k) - t\|v^* - v_j\| \\
&\geq \|v^* - v_j\| \cdot (1 - t) \quad \text{(vgl. (B.30))} \\
&= \|u - v_j\| \quad (\text{da } (1-t)(v^* - v_j) = v^* + t(v_j - v^*) - v_j = u - v_j) \\
&\geq d(u, c_j).
\end{aligned}
$$

Folglich gilt $(*)$.

$$
\text{Man definiere:} \quad \varphi : [0,1] \to [v^*, v_j], \quad t \mapsto v_j + t \cdot (v^* - v_j), \qquad \text{(B.32)}
$$
$$
\psi : [v^*, v_j] \to I\!R, \quad x \mapsto d(x, c_j) - d(x, c_i). \qquad \text{(B.33)}
$$

Da S dicht auf I ist, gilt für alle $x \in [v^*, v_j]$:

$$
\begin{aligned}
\psi(x) &= \inf\{\|x - \eta\| \mid \eta \in c_j\} - \inf\{\|x - \eta\| \mid \eta \in c_i\} \\
&= \min\{\|x - \eta\| \mid \eta \in c_j \cap I\} - \min\{\|x - \eta\| \mid \eta \in c_i \cap I\}. \qquad \text{(B.34)}
\end{aligned}
$$

Da $c_i \cap I$ bzw. $c_j \cap I$ kompakt sind, ist nach Bemerkung B.2 die Abbildung ψ stetig und damit ist auch $\omega := \psi \circ \varphi : [0,1] \to I\!R$ eine stetige Abbildung.
Für $u = v^*$ ist c_i das bzgl. d nächste Neuronenzentrum zu u und c_j das zweitnächste. Für $u = v_j$ ist c_j das bzgl. d nächste Neuronenzentrum zu u und c_i das zweitnächste.
Also gilt $\omega(0) \leq 0$ und $\omega(1) \geq 0$. Nach dem Zwischenwertsatz existiert somit ein $t^* \in [0,1]$ mit $\omega(t^*) = 0$. Für $u^* := \varphi(t^*) \in [v^*, v_j]$ gilt somit

$$
d(u^*, c_j) = d(u^*, c_i). \qquad \text{(B.35)}
$$

Also gilt $u^* \in [v^*, v_j] \subseteq \Delta(v^*, v_j, v_i) \subseteq I$ und $u^* \in V_i \cap V_j \cap I = V_i^{(I)} \cap V_j^{(I)} \neq \emptyset$. □

B.2 Interpolation in π-FDCS-Netzwerken

Lemma B.1 (Interpolationseigenschaft der π-FDCS-Netzwerke)
Sei $\mathcal{F} = (I, O, U, L, \Pi, \rho, \alpha, r, \delta_{\mathcal{F}})$ ein π-FDCS-Netzwerk und $\delta_{\mathcal{F}}$ die in (4.49) angegebene Evaluierungsfunktion. Bei Verwendung der in (4.45) und (4.46) angegebenen Aktivierungsfunktionen gilt dann für alle $u \in \Pi$ und alle $x \in I$ mit

$$
\alpha_u(x) = 1 \wedge \forall v \in Nh(u) : 0 < \alpha_v(x) < 1 \qquad \text{(B.36)}
$$

die folgende Interpolationseigenschaft:

$$
\delta_{\mathcal{F}}(x) = w_u. \qquad \text{(B.37)}
$$

Beweis:
Sei $u \in \Pi$ und sei $x \in I$ mit $\alpha_u(x) = 1$, so daß $\alpha_v(x) < 1$ für alle $v \in Nh(u)$ gilt. Mit einer der Aktivierungsfunktionen (4.45) oder (4.46) gilt dann $u = bmu$. Dann ist $(1 - \alpha_u(x)) = 0$ und somit $\bar{\alpha}_v(x) = 0$ für alle $v \in Nh(u)$. Folglich gilt dann mit $y := \delta_{\mathcal{F}}(x)$ für alle $i \in \{1, \cdots, m\}$:

$$y_i = \frac{\bar{\alpha}_u(x) \cdot (w_u)_i}{\bar{\alpha}_u(x)} = (w_u)_i, \tag{B.38}$$

und somit $\delta_{\mathcal{F}}(x) = w_u$.

□

Da für alle $x \in c_u$ bei Verwendung der Aktivierungsfunktionen (4.45) und (4.46) $\alpha_u(x) = 1$ gilt, werden die Stützvektoren von exakten Neuronen an den Neuronenzentren gehalten. Also ist die Interpolationseigenschaft gewährleistet. Problematisch ist nur der Fall, wenn $x \in c_u \cap c_v \cap I$ mit $w_u \neq w_v$ eintritt. Dann sind widersprüchliche exakte Regeln im Netzwerk enthalten, die zu einer undefinierten Ausgabe führen!

B.3 Gradientenbestimmung in π-FDCS-Netzwerken

Sowohl im FDCS-REINFORCE-Ansatz als auch im Ansatz des reellwertigen modifizierten Q-Lernen werden Gradienten nach der Eingabe und auch Gradienten nach den Gewichten von π-FDCS-Netzwerken verwendet. Wie sich diese Gradienten berechnen lassen, soll hier gezeigt werden.
Dazu wird zunächst das folgende Lemma benötigt:

Lemma B.2 (Mehrfache Produktregel)
Seien $f_1, \cdots, f_m : I\!R^n \to I\!R$ partiell differenzierbar. Dann ist $\prod_{i=1}^m f_i$ partiell differenzierbar und es gilt

$$\frac{\partial}{\partial x_i}\left(\prod_{k=1}^m f_k\right)(x) = \sum_{l=1}^m \left(\frac{\partial}{\partial x_i} f_l(x) \cdot \prod_{\substack{k=1\\k\neq l}}^m f_k(x)\right). \tag{B.39}$$

Beweis: Sei $i \in \{1, \cdots, m\}$
Induktion über $m \in I\!N$:
<u>I. A.:</u> Der Fall $m = 1$ ist trivial.
<u>I. S.:</u> Sei $m \in I\!N$ und gelte dafür die I. V. (B.39). Dann gilt:

$$\begin{aligned}
\frac{\partial}{\partial x_i}\left(\prod_{k=1}^{m+1} f_k\right)(x) &= \frac{\partial}{\partial x_i}\left[\left(\prod_{k=1}^{m} f_k\right) \cdot f_{m+1}\right](x) \\
&\overset{Produktregel}{=} \frac{\partial}{\partial x_i}\left(\prod_{k=1}^{m} f_k\right)(x) \cdot f_{m+1}(x) + \left(\prod_{\substack{k=1\\k\neq m+1}}^{m+1} f_k\right)(x) \cdot \frac{\partial}{\partial x_i} f_{m+1}(x) \\
&\overset{\underline{I.V.}}{=} \sum_{l=1}^m \frac{\partial}{\partial x_i} f_l(x) \cdot \prod_{\substack{k=1\\k\neq l}}^{m+1} f_k(x) + \frac{\partial}{\partial x_i} f_{m+1}(x) \cdot \prod_{\substack{k=1\\k\neq m+1}}^{m+1} f_k(x)
\end{aligned}$$

$$= \quad \sum_{l=1}^{m+1} \frac{\partial}{\partial x_i} f_l(x) \cdot \prod_{\substack{k=1\\k\neq l}}^{m+1} f_k(x).$$

□

Im folgenden wird die komplexeste Variante der FDCS-Netzwerke bezüglich der Gradientenberechnung betrachtet, nämlich die π-FDCS-Netzwerke mit Normalisierung. Für alle weiteren FDCS-Funktionen (ohne exakte Neuronen, mit voll verbundenen Ein- und Ausgabeschichten, ohne Normalisierung) sind die Gradienten einfacher zu berechnen oder sind schon als Spezialfall in den π-FDCS-Netzwerken enthalten.
Dazu wird nun ein π-FDCS-Netzwerk $\mathcal{F} = (I, O, U, L, \Pi, \rho, \alpha, r, \delta_{\mathcal{F}})$ als vorgegeben betrachtet. Ferner seien $u, v \in U$ neuronale Einheiten, $n \in \mathbb{N}$ die Anzahl der Eingabekomponenten, $m \in \mathbb{N}$ die Anzahl der Ausgabekomponenten, und $i \in \{1, \cdots, n\}$ und $j, k \in \{1, \cdots, m\}$.
Der Beweis des folgenden Satzes ist trivial.

Satz B.3 (Gradienten nach den Ausgabegewichten)
Es gilt:

$$\frac{\partial}{\partial (w_u)_j} \delta_{\mathcal{F}}(x)_k = \begin{cases} \bar{\alpha}_u(x) & : \quad u \in \{bmu\} \cup Nh(bmu) \wedge (w_u)_j \neq \perp \wedge j = k \\ 0 & : \quad sonst \end{cases} \tag{B.40}$$

□

Aufwendiger dagegen ist die Berechnung der Gradienten nach den Zentren und nach den Stimuli. Es gilt das folgende

Lemma B.3 (Gradienten der Produktaktivitäten nach den Zentren)

$$\frac{\partial}{\partial (c_u)_i} \bar{\alpha}_v(x) = \begin{cases} (1-\alpha_u(x))\bar{\alpha}_u(x)\frac{\partial}{\partial (c_u)_i}\alpha_u(x) & : \quad u = v \\ -\alpha_v(x)\frac{\partial}{\partial (c_u)_i}\alpha_u(x)\prod_{p\in\Pi\setminus\{v,u\}}(1-\alpha_p(x)) & : \quad u \neq v \end{cases} \tag{B.41}$$

Beweis: Es gilt:

$$\begin{aligned}
\frac{\partial}{\partial (c_u)_i}\bar{\alpha}_v &= \frac{\partial}{\partial (c_u)_i}\alpha_v(x)\prod_{p\in\Pi\setminus\{v\}}(1-\alpha_p(x)) + \alpha_v(x)\frac{\partial}{\partial (c_u)_i}\prod_{p\in\Pi\setminus\{v\}}(1-\alpha_p(x)) \\
&\overset{Lemma B.2}{=} \frac{\partial}{\partial (c_u)_i}\alpha_v(x)\prod_{p\in\Pi\setminus\{v\}}(1-\alpha_p(x)) \\
&\quad -\alpha_v(x)\left(\sum_{p\in\Pi\setminus\{v\}}\frac{\partial}{\partial (c_u)_i}\alpha_p(x)\prod_{q\in\Pi\setminus\{v,p\}}(1-\alpha_q(x))\right)
\end{aligned}$$

Ist $u \neq v$, so ist der erste Summand im letzten Ausdruck gleich 0. Ferner sind nur die Summanden der großen Summe ungleich 0, für die $p = u$ gilt. Durch das Zusammenfassen dieser Ausdrücke erhält man die Behauptung.

□

Ganz entsprechend erhält man folgendes Lemma, welches die Gradienten der Produktaktivitäten nach den Stimuli angibt. Der Beweis ist ähnlich einfach wie beim vorhergehenden Lemma, nur daß sich die Ausdrücke hier nicht vereinfachen lassen.

Lemma B.4 (Gradienten der Produktaktivitäten nach den Stimuli)
Es gilt

$$\frac{\partial}{\partial x_i}\bar{\alpha}_u(x) = \frac{\partial}{\partial x_i}\alpha_u(x) \prod_{p\in\Pi\setminus\{u\}} (1-\alpha_p(x)) - \alpha_u(x) \sum_{p\in\Pi\setminus\{u\}} \frac{\partial}{\partial x_i}\alpha_p(x) \prod_{q\in\Pi\setminus\{u,p\}} (1-\alpha_q(x)). \tag{B.42}$$

□

Obwohl insbesondere die letzten beiden Arten von Gradienten sehr aufwendig zu berechnen sind, kann man für eine praktische Implementierung in den meisten Fällen den Aufwand drastisch senken. Setzt man

$$P(x) = \prod_{p\in\Pi}(1-\alpha_p(x)) \tag{B.43}$$

und gilt für ein Neuron $q \in \Pi$ $\alpha_q(x) \neq 1$, so erhält man in allen obigen Formeln für das große Produkt:

$$\prod_{p\in\Pi\setminus\{q\}} (1-\alpha_p(x)) = \frac{P(x)}{1-\alpha_q(x)}. \tag{B.44}$$

Dadurch wird bis auf den Ausnahmefall $\alpha_q(x) = 1$ der Aufwand um eine ganze Größenordnung reduziert.
Mit der Kenntnis der bisherigen partiellen Ableitungen, können nun die Gradienten der durch das Netzwerk berechenten Funktion nach den Zentren und nach den Stimuli angegeben werden.

Satz B.4 (Gradienten der Evaluierungsfunktion nach den Zentren)
Für die normalisierte Evaluierungsfunktion $\delta_{\mathcal{F}}$ gilt

$$\frac{\partial}{\partial (c_u)_i}\delta_{\mathcal{F}}(x)_j = \frac{1}{\sum\limits_{\substack{v\in\{bmu\}\cup Nh(bmu)\\ (w_v)_j\neq\perp}} \bar{\alpha}_v(x)} \cdot \sum_{\substack{v\in\{bmu\}\cup Nh(bmu)\\ (w_v)_j\neq\perp}} ((w_v)_j - \delta_{\mathcal{F}}(x)_j)\frac{\partial}{\partial (c_u)_j}\bar{\alpha}_v(x) \tag{B.45}$$

Beweis: Setze

$$T_j(x) := \sum_{v\in\{bmu\}\cup Nh(bmu)\wedge(w_v)_j\neq\perp} \bar{\alpha}_v(x).$$

Dann gilt

$$\begin{aligned}
\frac{\partial}{\partial (c_u)_i}\delta_{\mathcal{F}}(x)_j &= \frac{T_j(x) \sum\limits_{\substack{v\in\{bmu\}\cup Nh(bmu)\\ (w_v)_j\neq\perp}} \frac{\partial\bar{\alpha}_v(x)}{\partial (c_u)_i}\cdot (w_v)_j - T_j(x)\delta_{\mathcal{F}}(x)_j \sum\limits_{\substack{v\in\{bmu\}\cup Nh(bmu)\\ (w_v)_j\neq\perp}} \frac{\partial\bar{\alpha}_v(x)}{\partial (c_u)_i}}{T_j(x)^2} \\
&= \frac{1}{T_j(x)} \cdot \sum_{v\in\{bmu\}\cup Nh(bmu)\wedge(w_v)_j\neq\perp} ((w_v)_j - \delta_{\mathcal{F}}(x)_j)\,\frac{\partial\bar{\alpha}_v(x)}{\partial (c_u)_j}.
\end{aligned}$$

□

Ganz analog läßt sich auch die Berechnung der partiellen Ableitungen der Evaluierungsfunktion nach den Stimuli aussprechen:

Satz B.5 (Gradienten der Evaluierungsfunktion nach den Stimuli)
Für die normalisierte Evaluierungsfunktion $\delta_{\mathcal{F}}$ gilt

$$\frac{\partial}{\partial x_i}\delta_{\mathcal{F}}(x)_j = \frac{1}{\sum\limits_{\substack{v\in\{bmu\}\cup Nh(bmu)\\(w_v)_j\neq\perp}}\bar{\alpha}_v(x)} \cdot \sum_{\substack{v\in\{bmu\}\cup Nh(bmu)\\(w_v)_j\neq\perp}}((w_v)_j - \delta_{\mathcal{F}}(x)_j)\frac{\partial}{\partial x_i}\bar{\alpha}_v(x) \tag{B.46}$$

□

Damit sind nun alle partiellen Ableitungen im Zusammenhang mit FDCS-Netzen angegeben. Es bleibt nur noch anzugeben, wie die partiellen Ableitungen nach den Gewichten einer Evaluierungsfunktion bestimmt werden können, dessen Evaluierungsfunktion mit einer weiteren verknüpft ist. Dieses ist beim reellwertigen modifizierten Q-Lernen der Fall, wo Q und π zwei Evaluierungsfunktionen sind und zur Adaption der Gewichte von π die Gradienten $\nabla_{c_u}Q\circ(id,\pi)$ und $\nabla_{w_u}Q\circ(id,\pi)$ bestimmt werden müssen. Es gilt der

Satz B.6 (Gradienten verknüpfter Netzwerke)
Seien $n_s, n_a \in \mathbb{N}$ und $Q:\mathbb{R}^{n_s}\times\mathbb{R}^{n_a}\rightarrow\mathbb{R}$ und $\pi:\mathbb{R}^{n_s}\rightarrow\mathbb{R}^{n_a}$ differenzierbare Funktionen. Sei $f = Q\circ(id_{\mathbb{R}^{n_s}},\pi)$ und p sei ein Parameter, von dem π abhängt, und sei $x\in\mathbb{R}^{n_s}$. Dann gilt:

$$\frac{\partial}{\partial p}f(x) = \sum_{i=n_s+1}^{n_s+n_a}\frac{\partial}{\partial x_i}Q(x,\pi(x))\frac{\partial}{\partial p}\pi_i(x). \tag{B.47}$$

Beweis: Der Beweis folgt direkt aus der Kettenregel der totalen Differentiale (vgl. [For84], S. 50).

□

Setzt man für den Parameter p ein Gewicht der Evaluierungsfunktion π ein so erhält man die Gradienten, wie sie beim reellwertigen modifizierten Q-Lernen benötigt werden.

Anhang C

Merkmalsextraktion

In diesem Teil des Anhangs wird dargestellt, wie die Hindernisliste, die im zweiten FDCS-basierten Hindernisvermeidungsmodul verwendet wurde, erstellt wird und wie aus ihr Merkmale zur Hindernisvermeidung extrahiert werden.

C.1 Erstellung der Hindernisliste

Betrachtet man einen Ultraschallkegel im zweidimensionalen egozentrischen Roboterkoordinatensystem, so wird dieser durch seine Lage $(x_s, y_s) \in \mathbb{R}^2$, seine Orientierung α_s, seinen charakteristischen Öffnungswinkel α_{char}, den durch den Sensor detektierten Abstand r zu einem Hindernis und die maximal meßbare Entfernung r_{max} beschrieben. In diesem Koordinatensystem wird der Ultraschallkegel als gleichschenkliges Dreieck $\Delta_1 = (p_s, p_1, p_2)$ mit $p_s = (x_s, y_s)$, $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ modelliert (vgl. Abbildung C.1), das in einem Raum ohne Hindernisse in das Dreieck $\Delta_2 = (p_s, p_1^*, p_2^*)$ mit $p_1^* = (x_1^*, y_1^*)$ und $p_2^* = (x_2^*, y_2^*)$ übergeht. Die gemessene Entfernung r ist in einem hindernisfreien Raum gleich der durch den Timeout-Wert des Sensors vorgegebene maximal meßbare Entfernung r_{max}. Setzt man

$$\begin{aligned}
R_x^\alpha : \mathbb{R}^2 \to \mathbb{R}, &\quad (x, y) \mapsto x \cdot \cos(\alpha) - y \cdot \sin(\alpha), &\text{(C.1)}\\
R_y^\alpha : \mathbb{R}^2 \to \mathbb{R}, &\quad (x, y) \mapsto x \cdot \sin(\alpha) + y \cdot \cos(\alpha), &\text{(C.2)}\\
R^\alpha : \mathbb{R}^2 \to \mathbb{R}^2, &\quad (x, y) \mapsto (R_x^\alpha(x), R_y^\alpha(y)), &\text{(C.3)}
\end{aligned}$$

so wird durch R^α die Rotation um den Koordinatenursprung mit dem Winkel α definiert. Die Punkte des Dreiecks zu einem Ultraschallsensor erhält man aus den charakteristischen Daten eines Sensors durch:

$$\begin{aligned}
y_h &= r \cdot \tan(\alpha_{char}) &\text{(C.4)}\\
y_h^* &= r_{max} \cdot \tan(\alpha_{char}) &\text{(C.5)}\\
p_1 = (x_1, y_1) &= R^{90^\circ - \alpha_s}(r, y_h) + (x_s, y_s) &\text{(C.6)}\\
p_2 = (x_2, y_2) &= R^{90^\circ - \alpha_s}(r, -y_h) + (x_s, y_s) &\text{(C.7)}\\
p_1^* = (x_1^*, y_1^*) &= R^{90^\circ - \alpha_s}(r_{max}, y_h) + (x_s, y_s) &\text{(C.8)}\\
p_2^* = (x_2^*, y_2^*) &= R^{90^\circ - \alpha_s}(r_{max}, -y_h) + (x_s, y_s) &\text{(C.9)}
\end{aligned}$$

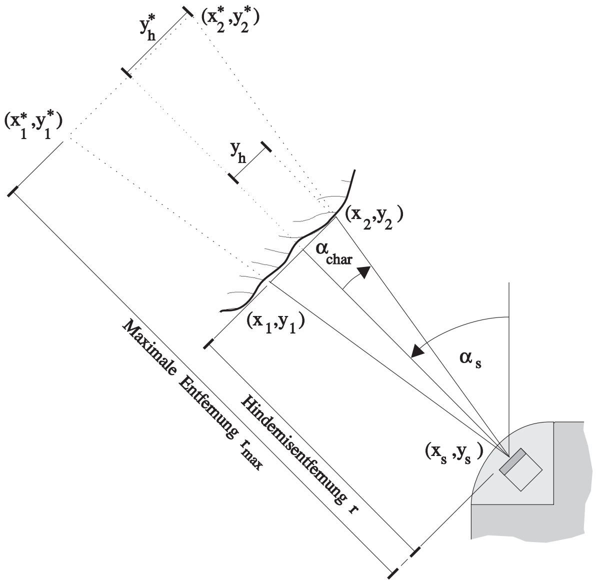


Abbildung C.1: Charakteristische Daten eines Ultraschallkegels

Das *Geradensegment* $[p_1, p_2]$ steht nun stellvertretend für ein Hindernis im zweidimensionalen egozentrischen Roboterkoordinatensystem. Die lokale Umgebung des Fahrzeuges wird im folgenden durch derartige Geradensegmente modelliert. Dazu sei $\mathcal{U}_t$ eine Menge von Geradensegmenten, die die lokale Umgebung des Fahrzeuges zum Zeitpunkt t repräsentiert. In der Implementation läßt sich diese Menge als verkettete Liste darstellen, deren Datensätze die Koordinaten der Randpunkte eines Geradensegmentes enthalten. Daher wird $\mathcal{U}_t$ hier auch als *Hindernisliste* bezeichnet.
Während des Überganges vom Zeitpunkt t zum Zeitpunkt $t+1$ führt das Fahrzeug eine Bewegung durch, die sich aus einer Translation um $(\Delta x_t, \Delta y_t)$ und eine Rotation zusammensetzt. α_t bezeichne dazu die Orientierung des Fahrzeuges zum Zeitpunkt t.
Um aus $\mathcal{U}_t$ auch für den Zeitpunkt $t+1$ eine Modellierung der Umwelt zu gewinnen, werden die Geradensegmente $[p, q] \in \mathcal{U}_t$ entgegen der Fahrzeugbewegung transformiert.
Da die Randpunkte der Geradensegmente immer in egozentrischen Koordinaten gespeichert sind, erhält man die neuen Positionen der Randpunkte durch eine Rotation um $-\alpha_t$, eine Translation um $(-\Delta x_t, -\Delta y_t)$ und eine anschließende Rotation um α_{t+1}. Somit wird jedes Geradensegment durch Anwendung der Transformation

$$T_t : \mathbb{R}^2 \to \mathbb{R}^2, \quad (x, y) \mapsto R^{\alpha_t} \left(R^{-\alpha_t}(x, y) - (\Delta x_t, \Delta y_t) \right) \tag{C.10}$$

auf seine Randpunkte in eine Lage gebracht, die zum Zeitpunkt $t+1$ Hindernisse im egozentrischen Koordinatensystem des Roboters beschreibt.
Da die Umgebungsrepräsentation nur zur Merkmalsextraktion und nicht zur Kartographierung dient, genügt es, ausschließlich die Geradensegmente in der nahen Umgebung des Fahrzeuges zu verwalten. Daher werden beim Übergang von t zu $t+1$ alle Geradensegmente aus der Hindernisliste gelöscht, deren Randpunkte außerhalb eines rechteckigen Bereiches

$B = [-o, o] \times [-o, o]$, $o \in \mathbb{R}_{>0}$ liegen. Dadurch bleibt die Größe der Hindernisliste immer beschränkt.
Um die Größe der Hindernisliste weiter zu reduzieren, bleiben Geradensegmente nur dann in der Liste enthalten, wenn ihre Länge größer einer vorgegebenen Schwelle $s \in \mathbb{R}_{>0}$ ist. Diese Reduktion der Liste ist notwendig, da durch die im folgenden zu beschreibenden Operationen Geradensegmente verkleinert werden können. Der Informationsgehalt zu kleiner Geradensegmente ist so gering, daß auf sie verzichtet wird.
Für den Übergang von t zu $t+1$ erhält man daher aus den bisher beschriebenen Transformationen ein erstes Zwischenergebnis $\tilde{\mathcal{U}}_t$ der neuen Hindernisliste

$$\tilde{\mathcal{U}}_t = \{[T_t(p), T_t(q)] \,|[p,q] \in \mathcal{U}_t \wedge \|T_t(p) - T_t(q)\| > s \wedge [T_t(p), T_t(q)] \cap B \neq \emptyset\}\,. \tag{C.11}$$

Dieses Zwischenergebnis beschreibt bisher nur die Lage der alten Geradensegmente zum Zeitpunkt $t+1$. Zur vollständigen Aktualisierung der Hindernisliste zum Zeitpunkt $t+1$ fehlen nun noch die Sensormessungen zum Zeitpunkt $t+1$.
Da jeder Sensor jeweils ein neues Geradensegment $[p_1, p_2]$ liefert, müssen diese zur alten Hindernisliste hinzugefügt werden. Das alleine reicht allerdings nicht aus, da damit nur Informationen über die Hindernisse und nicht über die Freiräume ausgedrückt werden.
Ist p_s die Position eines Ultraschallsensors im egozentrischen Koordinatensystem, so wird durch das Innere des Dreiecks $\Delta_1 = (p_s, p_1, p_2)$ ein hindernisfreier Raum angegeben. Da auch Geradensegmente hinter dem neu einzufügenden Hindernis $[p_1, p_2]$ zur Beschreibung der Umgebung des Fahrzeuges nicht mehr von Interesse sind, werden alte Geradensegmente in $\tilde{\mathcal{U}}_t$, die innerhalb des Dreiecks $\Delta_2 = (p_s, p_1^*, p_2^*)$ liegen, aus der Hindernisliste entfernt. Entsprechend werden alte Geradensegmente, die nur teilweise innerhalb des Dreiecks Δ_2 liegen, verkürzt oder in zwei kleinere nicht zusammenhängende Geradensegmente aufgespalten.
Dazu betrachte man ein altes Geradensegment $[p, q] \in \tilde{\mathcal{U}}_t$, welches das Dreieck Δ_2 schneidet.

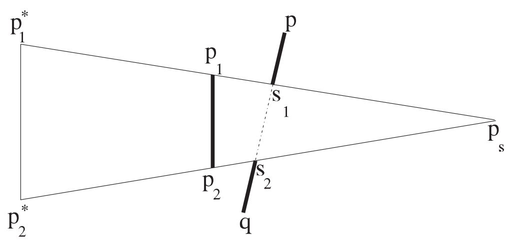


Abbildung C.2: Aktualisierung der Geradensegmente

Die Abbildung C.2 zeigt einen möglichen Fall, wie ein altes Geradensegment das Dreieck des Ultraschallkegels schneiden kann.
Das Geradensegment $[p, q]$ schneidet Δ_2 in den Punkten s_1 und s_2, die mit dem in Anhang A beschriebenen Verfahren zur Schnittpunktberechnung von Geradensegmenten ermittelt werden.
In diesem Fall ist das Segment $[p, q]$ in der Hindernisliste durch die beiden Segmente $[p, s_1]$ und $[q, s_2]$ zu ersetzen.

Entsprechend sind die verbleibenden Fälle zu behandeln, bei denen sich zwischen dem Rand von Δ_2 und alten Geradensegmenten ein, zwei oder auch kein Schnittpunkt ergibt. Erhält man keinen Schnittpunkt, so ist zu testen, ob ein altes Geradensegment innerhalb oder außerhalb des Dreiecks Δ_2 liegt.
Somit erhält man die Hindernisliste $\mathcal{U}_{t+1}$ aus dem Zwischenergebnis $\tilde{\mathcal{U}_t}$ durch Hinzufügen der aktuell durch die Ultraschallsensoren gelieferten Geradensegmente und durch anpassen der alten Geradensegmente aus $\tilde{\mathcal{U}_t}$ an die neuen Ultraschallkegel.

C.2 Extraktion von Merkmalen

Ausgehend von den Geradensegmenten der Menge $\mathcal{U}_t$, die Hindernisse in der Umgebung repräsentieren, werden nun weitere Merkmale abgeleitet.
Statt die gewünschten Merkmale direkt aus der Liste der Hindernissegmente zu berechnen, ist es günstiger, eine geeignetere Darstellung der Umgebung aus der Hindernisliste zu gewinnen.
Dazu wird die lokale Umgebung des Fahrzeuges in $S \in I\!N$ Sektoren eingeteilt.

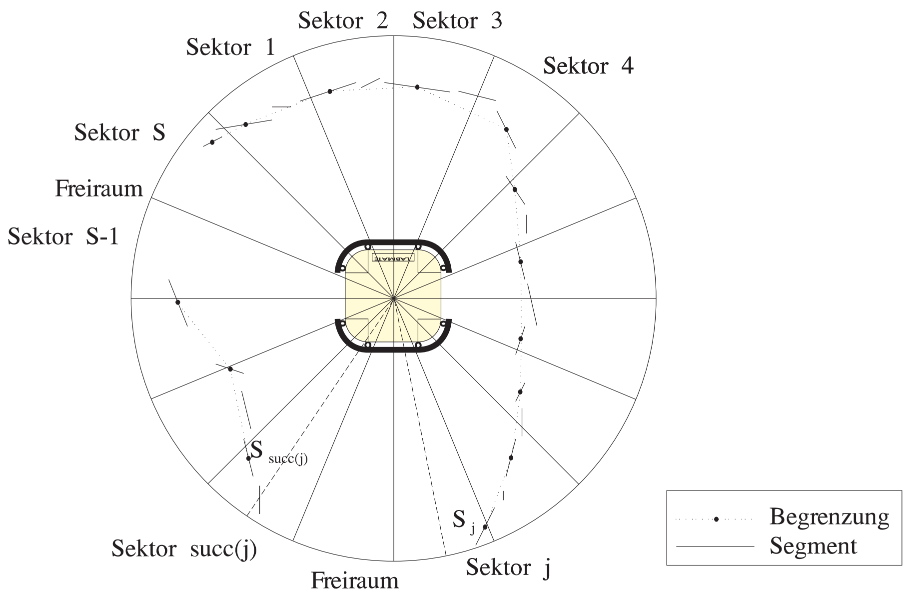


Abbildung C.3: Sektorliste und Umgebungsbegrenzung

Für jeden Sektor wird derjenige Punkt ermittelt, der Mittelpunkt eines Geradensegmentes innerhalb des Sektors ist und minimalen Abstand zum Ursprung besitzt.
Für Geradensegmente, die länger als die Diagonale des Fahrzeuges[1] ($> \sqrt{2}d_{LAB}$) sind, werden nicht nur die Mittelpunkte der Segmente, sondern zusätzlich auch die Randpunkte berücksichtigt.
Diese Sektorliste wird durch zwei eindimensionale Felder d_k und $S_k = (x_k, y_k)$, $k \in \{1, \cdots, S\}$ angegeben, die aus der Hindernisliste mit einem Aufwand von $O(|\mathcal{U}_t|)$ berechnet

[1] Der Wert von d_{LAB} gibt die Fahrzeugbreite an.

werden. Hierbei gibt $S_k = (x_k, y_k)$ den Punkt selbst und d_k den Abstand des Punktes zum Koordinatenursprung an. Zu Beginn der Berechnung der Sektorenliste werden die d_k mit einem Wert d_∞ initialisiert, der größer als die maximal meßbare Entfernung ist. Damit können später Sektoren erkannt werden, in denen sich kein Hindernis befindet. Dazu ist lediglich $d_k = d_\infty$ zu überprüfen.
Die in der Sektorenliste abgespeicherten Punkte (x_k, y_k) geben einen groben Verlauf der Begrenzung der Umgebung des Roboters an. Um diese Begrenzung geeignet zu modellieren werden die Punkte so miteinander verbunden, daß der verbindende Polygonzug die Umgebungsbegrenzung darstellt. Der Polygonzug kann sehr effizient durch die Datenstruktur einer Nachbarschaftsliste repräsentiert werden. Mit dieser Datenstruktur sind weitere Merkmale mit deutlich weniger Aufwand zu berechnen, als dieses direkt mit der Hindernisliste möglich wäre.
Diese Nachbarschaftsliste wird als ein Feld $next_j$, $j \in \{1, \cdots, S\}$ repräsentiert. Dabei enthält $next_j$ den Index k des im Uhrzeigersinn folgenden Sektors, wenn die Punkte S_j und S_k auf dem Polygonzug der Umgebungsbegrenzung verbunden sind. Ist in der Umgebung ein größerer Freiraum, so muß die Umgebungsbegrenzung unterbrochen sein. Dazu enthält die Nachbarschaftsliste den Wert -1, wenn ein Sektor den Rand einer Umgebungsbegrenzung darstellt und im Uhrzeigersinn mit keinem weiteren Sektor verbunden ist.
Unter einem Freiraum ist eine Lücke in der Umgebungsbegrenzung zu verstehen, die durch benachbarte Sektoren k mit $d_k = d_\infty$ gebildet wird. Die an diesen Freiraum angrenzenden Randpunkte der Umgebungsbegrenzung müssen einen Abstand zueinander besitzen, der größer einem vorgegebenem Schwellwert D ist. Der Schwellwert D gibt den maximalen Abstand zweier Punkte auf der Umgebungsbegrenzung an, die noch verbunden sein sollen. Es wird damit angegeben, wie groß eine Lücke in der Umgebungsbegrenzung sein muß, damit sie als Freiraum aufgefaßt wird.
Ein Eintrag $next_j$ der Nachbarschaftsliste wird nun berechnet, indem ausgehend von dem Sektor j die im Uhrzeigersinn $\frac{S}{2}$ folgenden Sektoren betrachtet werden. Unter diesen Sektoren k, deren zugehöriger Punkt S_k einen Abstand von S_j besitzt, der kleiner als D ist und für die $d_k < d_\infty$ gilt, wird der Sektor als Nachbar von j angesehen, dessen Indexabstand zu j minimal ist.
Für eine formale Definition eines Eintrages $next_j$ der Nachbarschaftsliste wird eine Hilfsmenge M_j definiert, die die Indexabstände der als Nachbarn von j in Frage kommenden Sektoren enthält.

$$M_j = \left\{ k \in \mathbb{N}_{\leq \frac{S}{2}} \,\middle|\, d_{(k+j) \bmod (S+1)} < d_\infty, \left\| S_j - S_{(k+j) \bmod (S+1)} \right\| < D \right\} \tag{C.12}$$

Diese Menge wird nur über die nächsten $\frac{S}{2}$ Sektoren gebildet, um zu verhindern, daß in der Nachbarschaftsliste Verbindungen entstehen, die entgegen dem Uhrzeigersinn gerichtet sind.
Schließlich wird durch

$$next_j = \begin{cases} (j + \min M_j) \bmod (\mathrm{S}{+}1) & : \quad M_j \neq \emptyset \\ -1 & : \quad sonst \end{cases}, \quad j \in \{1, \cdots, S\}. \tag{C.13}$$

der Nachbarsektor von j definiert. Ist $next_j = -1$, so bildet Sektor j den Rand eines Freiraumes oder das Innere eines Freiraumes. Der Modulo-Operator wird hier verwendet, da die zyklische Sektoraufteilung des Kreises auf ein lineares Feld abgebildet wird.

Diese Verbindungsstruktur läßt sich mit einem Zeitaufwand von $O(S)$ Schritten erstellen und bietet eine effiziente Datenstruktur zur weiteren Merkmalsgewinnung.
In Abbildung C.3 ist das Fahrzeug mit der Sektoreinteilung seiner lokalen Umgebung dargestellt. Ferner sind die Hindernissegmente dargestellt, auf denen die Punkte mit der geringsten Entfernung zum Ursprung für jeden Sektor eingezeichnet sind. Aufeinanderfolgende Punkte, deren Abstand nicht zu groß ist, sind durch einen Polygonzug verbunden, der die Begrenzung der Umgebung darstellt. Dieser Polygonzug bietet eine effiziente Darstellung der Umgebung, um weitere Merkmale ableiten zu können.

C.2.1 Freiraumrichtungen

Das erste Merkmal, das aus der Umgebungsbegrenzung abgeleitet werden soll, ist die Freiraumrichtung. Sie gibt vom Ursprung ausgehend eine Richtung an, die auf einen Freiraum in der Umgebungsbegrenzung zeigt. Unter einem Freiraum wird eine Unterbrechung in der Umgebungsbegrenzung verstanden, wobei der Abstand der beiden Randpunkte der Unterbrechung so groß ist, daß das Fahrzeug gefahrlos durch diesen freien Raum in der Umgebung fahren kann. Da es im allgemeinen in der Umgebung mehrere solche Freiräume gibt, man aber nur an einer Fahrtrichtung interessiert ist, wird aus allen Freiraumrichtungen die Richtung ausgewählt, die am besten mit einer vorgegebenen Richtung $\alpha_{goal} \in [-180°, 180°]$ übereinstimmt. Diese vorgegebene Richtung ist bei der Zielverfolgung direkt die Richtung des Zielvektors oder bei einer Art Wandverfolgung wird sie als $\alpha_{goal} = 0°$ gesetzt. Dann wird der Freiraum ausgewählt, der die geringste Abweichung von der momentanen Fahrtrichtung besitzt.
Eine Freiraumrichtung wird ermittelt, indem man einen Sektor $j \in \{1, \cdots, S\}$ betrachtet, der selbst Nachbarsektor eines anderen Sektors ist und für den $next_j = -1$ gilt. Dann stellt der Punkt $S_j = (x_j, y_j)$ einen Randpunkt der Umgebungsbegrenzung dar. Diese Sektoren werden in der Menge

$$L = \{j \in \{1, \cdots, S\} \,|next_j = -1 \wedge \exists k \in \{1, \cdots, S\} : \quad next_k = j\} \tag{C.14}$$

zusammengefaßt. Durch die Abbildung

$$\begin{aligned} succ \;:\; & L \to \{1, \cdots, S\}, \\ & j \mapsto \left(j + \min\left\{k \in I\!N_{\leq S} \,\middle|\, next_{(j+k) \bmod (S+1)} \neq -1\right\}\right) \bmod (S+1) \end{aligned} \tag{C.15}$$

wird zu jedem Sektor j der im Uhrzeigersinn nachfolgende Sektor angegeben, der wieder einen Randpunkt der Umgebungsbegrenzung enthält. Bestimmt man ein $j \in L$, so befindet sich zwischen j und $succ(j)$ ein Freiraum.
Bei genügend großer Sektorenzahl können die Winkel der Freiraumränder durch $\bar{\alpha}_j$ und $\bar{\alpha}_{succ(j)}$ mit

$$\bar{\alpha}_j \;=\; 45° - \left(j + \frac{1}{2}\right) \cdot \frac{360°}{S} \tag{C.16}$$

$$\bar{\alpha}_{succ(j)} \;=\; 45° - \left(succ(j) + \frac{1}{2}\right) \cdot \frac{360°}{S} \tag{C.17}$$

approximiert werden. Durch eventuelles Addieren oder Subtrahieren von 360° erreicht man außerdem, daß $\bar{\alpha}_j \leq \bar{\alpha}_{succ(j)}$ gilt.
Da das Fahrzeug nicht punktförmig ist, sondern eine Breite d_{LAB} besitzt, muß der effektiv nutzbare Winkelbereich verkleinert werden, so daß der nutzbare Freiraum durch den Winkelbereich zwischen α_j und $\alpha_{succ(j)}$ angegeben wird, wobei

$$\alpha_j = \bar{\alpha}_j + \frac{180° \cdot \sqrt{2} \cdot d_{LAB}}{\pi \cdot d_j} \tag{C.18}$$

$$\alpha_{succ(j)} = \bar{\alpha}_{succ(j)} - \frac{180° \cdot \sqrt{2} \cdot d_{LAB}}{\pi \cdot d_{succ(j)}} \tag{C.19}$$

gilt. Gilt $\alpha_j \leq \alpha_{succ(j)}$, so erhält man einen Winkelbereich eines Freiraumes. Je nach Wahl von α_{goal} wird aus diesem Winkelbereich $[\alpha_j, \alpha_{succ(j)}]$ eine Freiraumrichtung abgeleitet. Diejenige Freiraumrichtung mit der geringsten Abweichung zu α_{goal} wird als *beste Freiraumrichtung* α_{best} definiert und gibt ein Merkmal an, das in der zweiten FDCS-basierten Hindernisvermeidung zur Steuerung eines Fahrzeuges verwendet wird.

C.2.2 Extraktion weiterer Merkmale

Neben der Extraktion der Freiräume aus der Hindernisliste, sind noch weitere nützliche Merkmalsextraktionen denkbar.
Durch die Darstellung der Umgebungsbegrenzung des Fahrzeuges als ein Polygonzug, sind *virtuelle Abstandssensoren* möglich. Diese werden durch eine Lage (x_v, y_v) und eine Orientierung α_v im egozentrischen Koordinatensystem angegeben. Die Messung einer Entfernung durch einen solchen virtuellen Sensor beschränkt sich auf eine Schnittpunktberechnung des Polygonzuges der Umgebungsbegrenzung mit der durch (x_v, y_v) und α_v definierten Halbgeraden.
Durch eine systematische Anordnung dieser virtuellen Sensoren sind damit Punkte auf seitlich oder frontal liegenden Wänden zu detektieren. Durch eine einfache Geradenanpassung an diese Punkte kann auf diese Weise beispielsweise die Entfernung und die Orientierung von seitlich oder frontal liegenden Wänden bestimmt werden, die als weitere Merkmale zur Steuerung des Fahrzeuges in Betracht kommen.

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

$\mathbb{N}$	Natürliche Zahlen	48
$\mathbb{N}_0$	Natürliche Zahlen mit 0	41
$\mathbb{B}$	Menge der Booleschen Wahrheitswerte 0, 1	48
$\mathbb{R}$	Reelle Zahlen	129
$\mathbb{R}^n$	Vektorraum reeller n-dimensionaler Vektoren	129
$\wedge$, $\vee$	Logisches Und, logisches Oder (auch für Fuzzy-Mengen)	47
$\forall x \in A$	Für alle $x \in A$ gilt	42
$\exists x \in A$	Es existiert ein $x \in A$ mit	50
$\emptyset$	Leere Menge	49
$\cap$, $\cup$	Durchschnitt und Vereinigung von Mengen	129
$\|A\|$	Mächtigkeit einer endlichen Menge	66
$V^{(2)}$	Menge aller zweielementigen Teilmengen von V	48
A^B	Menge aller Abbildungen $f : B \to A$	61
$\setminus$	Mengendifferenz	50
$\overline{A}$	Topologische Abschluß einer Menge	59
$\mathrm{Def}(f)$	Definitionsbereich einer Abbildung	59
$\mathrm{supp}(f)$	Träger einer Funktion	59
$\bigtimes_{i=1}^{n} A_i$	Mehrfaches kartesisches Produkt $A_1 \times \cdots \times A_n$	60
(p, q)	Gerade durch die Punkte $p \neq q \in \mathbb{R}^n$	129
$[p, q]$	Verbindungsstrecke von $p, q \in \mathbb{R}^n$	129
$[a, b]$	Abgeschlossenes Intervall von a bis b, $a, b \in \mathbb{R}$	130
$V \leq W$	V ist Teilraum des Vektorraums W	133
$\det A$	Determinante einer reellen $n \times n$-Matrix $A \in \mathbb{R}^{n \times n}$	129
$\nabla_w f(x)$	Gradient der Funktion f nach w an der Stelle x	65
$\frac{\partial}{\partial w} f(x)$	Partielle Ableitung der Funktion f nach w an der Stelle x	98
$f \circ g$	Hintereinanderausführung von Abbildungen	60
id_A	Identität auf einer Menge A	61
$n \bmod m$	Ganzzahliger Divisionsrest von n/m	148
χ_A	Charakteristische Funktion zu einer Menge A	42
$\min A$	Minimum einer Menge A	42
$\max A$	Maximum einer Menge A	42
$\arg\min\{f(x)\|x \in A\}$	Argument $x \in A$, für das $f(x)$ minimal ist	53
$\arg\max\{f(x)\|x \in A\}$	Argument $x \in A$, für das $f(x)$ maximal ist	53
$\inf A$	Infimum einer Menge A	134

$\lceil x \rceil$	Kleinste ganze Zahl z mit $z > x$	55
$\lvert x \rvert$	Betrag von $x \in I\!R$	42
$\lVert x \rVert$	Euklidische Norm von $x \in I\!R^n$	48
d_{stop}	Minimal erlaubter Frontabstand	38
d_{min}	Minimaler frontaler Abstand	38
v_{max}	Maximal zulässige Geschwindigkeit	39
$\mathcal{J}$	Winkelbereiche zur Rekalibrierung der Odometrie	41
h_k	Histogrammfunktion zur Rekalibrierung der Odometrie	42
a_k	Alter eines Winkelbereichs	42
M_k	Maximum des Histogramms im $k$ten Schritt	42
$\mathcal{M}_k$	Menge der lokalen Maxima im Histogramm im $k$ten Schritt	42
V	Knotenmenge des Wegegraphen	48
E	Kantenmenge des Wegegraphen	48
κ	Kantenmarkierung des Wegegraphen	48
δ_0	Mindestabstand zwischen zwei Wegemarken	49
δ_1	Radius zur Identifizierung einer alten Wegemarke	49
η_{low}, η_{high}	Felder zur Wegmarkenerkennung	47
I	Eingaberaum eines neuronalen Netzwerkes	58
O	Ausgaberaum eines neuronalen Netzwerkes	58
U	Neuronenmenge eines DCS-Netzwerkes	58
L	Laterale Verbindungen zwischen neuronalen Einheiten	58
$\delta_{\mathcal{N}}$	Evaluierungsfunktion eines neuronalen Netzes $\mathcal{N}$	58
c_u	Zentrum eines Neurons $u \in U$	58
w_u	Ausgabegewicht eines Neurons $u \in U$	58
$(w_u)_j$	$j$te Komponente eines Gewichtsvektors w_u	58
r_u	Resource eines Neurons $u \in U$	61
α_u	Aktivierungsfunktion eines Neurons $u \in U$	58
η_H	Normierung des Eingaberaumes	60
$\rho(u, v)$	Verbindungsstärke einer Kante zwischen zwei Neuronen u, v	61
bmu, $second$	Zwei Neuronen, die am besten zur Eingabe passen	62
$Nh(u)$	Nachbarschaft eines Neurons u	63
ε_{bmu}, ε_{Nh}	Lernkonstanten zur Adaption der Neuronenzentren	63
η	Lernkonstante der Delta-Regel	65
$E_{\mathcal{D}}(x, y)$	Quadratischer Fehler eines Netzwerkes $\mathcal{D}$	65
λ_{ins}	Schrittzahl nachdem neues Neuron eingefügt wird	66
N_{max}	Maximale Neuronenanzahl	66
α_c	Alterungsrate für Neuronenverbindungen	66
θ	Schwellwert zum Löschen von Neuronenverbindungen	62
δ	Abstands-Schwellwert zum Einfügen eines neuen Neurons	66
γ_{res}	Faktor zur Bildung der Resource eines Neurons	67
μ	Fuzzy-Menge	69
$x \approx c$	Unscharfe Gleichheitsrelation	70
$\perp$	Undefinierter Wert	72
Π	Menge von exakten Neuronen	74

$eval$	Zustandsbewertung	79
p	Schmerzfunktion	79
v	Fahrtgeschwindigkeit des AMR	82
ω	Überlagerte Rotationsgeschwindigkeit des AMR	82
$\mathcal{S}$	Menge aller Systemzustände eines Agenten	94
$\mathcal{A}$	Menge aller ausführbaren Aktionen eines Agenten	94
π	Politik-Funktion eines Agenten	94
r_t	Belohnungssignal zum Zeitpunkt t	94
s_t	Zustand des Agenten zum Zeitpunkt t	94
a^t	Durch den Agenten ausgeführte Aktion zum Zeitpunkt t	94
$E[X\|A]$	Erwartungswert von X bei gegebenem Ereignis A	95
$P(X\|A)$	Bedingte Wahrscheinlichkeit von X bei gegebenem A	97
$V_\pi(s)$	Für die Zukunft erwarteter Ertrag	95
$Q_\pi(s,a)$	Erwarteter Ertrag bei Ausführung der Aktion a	96
γ	Diskontierungsrate	95
$\tau(u)_t$	Spur eines Neurons u zum Zeitpunkt t	103
λ	Lernrate des TD-Lernens	99
β_V, β_μ, β_σ	Lernkonstanten des REINFORCE-Lernens	103
V_i	Verallgemeinerte Voronoi-Zelle	134
V_{ij}	Verallgemeinerte Voronoi-Zelle zweiter Ordnung	134
$A^{(I)}$	Maskierung von A mit I: $A \cap I$	135
$\mathcal{D}_S$	Delaunay-Triangulation zu einer Menge S von Zentren	134
A^*	Menge aller affinen Teilräume des $\mathbb{R}^n$	133
$\Delta(x,y,z)$	Das von x, y und z aufgespannte Dreieck	136
$\mathcal{U}_t$	Menge der Hindernis repräsentierenden Geradensegmente	145
$\tilde{\mathcal{U}}_t$	An die Bewegung des AMR angepaßte Geradensegmente	146
Δ_1, Δ_2	Dreiecke zur Modellierung eines Ultraschallkegels	144
T_t	Transformation eines Randpunktes eines Geradensegmentes	145
$next_j$	Verweis auf nächsten Sektor in der Umgebungsbegrenzung	148
S_j	Punkt in der Umgebungsbegrenzung	148
$succ(j)$	Nachfolger des $j$ten Sektors in der Umgebungsbegrenzung	149

Literaturverzeichnis

[ABS95] I. Ahrns, J. Bruske und G. Sommer. On-line learning with dynamic cell structures. In *Proc. of the ICANN'95*, Seiten 141–146, Paris, Oktober 1995.

[And90] C. W. Anderson. Q-Learning with Hidden-Unit Restarting. In *Proc. of the NIPS 2*, Seiten 81–87, San Mateo, 1990. Morgan Kaufman.

[BB93] A. G. Barto und S. J. Bradtke. Learning to Act using Real-Time Dynamic Programming. Technical report, University of Massachusettes, 1993.

[BB+95] J. Buhmann, W. Burgard et al. The Mobile Robot Rhino. *AI Magazine*, 16(1), 1995. Erhältlich unter:
http://www.cs.cmu.edu/~thrun/papers/buhmann.rhino-ai-magazine.html.

[Bra93] V. Braitenberg. *Vehikel Experimente mit kybernetischen Wesen*. Rowohlt, Reinbek bei Hamburg, 1993.

[Bro86] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Transactions on Robotics and Automation*, 2(1):14–23, 1986.

[BS95] J. Bruske und G. Sommer. Dynamic Cell Structure learns Perfectly Topology Preserving Map. *Neural Computation*, 7(4):845–865, 1995.

[BSA83] A. G. Barto, R. S. Sutton und C. W. Anderson. Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems. *IEEE Transaction on Systems, Man and Cybernetics*, 5:834–846, 1983.

[CM93] J. H. Connell und S. Mahadevan. *Robot Learning*, Kapitel Rapid Task Learning for Real Robots, Seiten 105–140. Kluwer Academic, 1993.

[Cro87] J. L. Crowley. The State of the Art in Mobile Robotics. In *Proc. of the Fourth International Symposium on Robotics in Construction*, Seiten 516–531, Haifa, Israel, 1987.

[DC94] A. Dubrawski und J. L. Crowley. Learning Locomotion Reflexes: A self-supervised neural system for a mobile robot. *Robotics and Autonomous Systems*, 12:33–142, 1994.

[Dig93] B. L. Digney. *Emergent Intelligence in a Distributed Adaptive Control System*. Dissertation, University of Saskatchewan, 1993.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[For84] O. Forster. *Analysis 2, Differentialrechnung im $\mathbb{R}^n$, Gewöhnliche Differentialgleichungen.* Vieweg, Braunschweig/Wiesbaden, 1984.

[Fri95] B. Fritzke. Growing Cell Structures - a self organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460, 1995.

[GF94] V. Gullapalli und J.A. Franklin. Acquiring Robot Skills via Reinforcement Learning. *IEEE Control Systems*, Seiten 13–24, 1994.

[HH93] D. R. Hush und B. G. Horne. Progress in Supervised Neural Networks. *IEEE Signal Processing Magazine*, Seiten 8–39, 1993.

[Hop92] P. Hoppen. *Autonome mobile Roboter: Echtzeitnavigation in bekannter und unbekannter Umgebung.* BI-Wiss.-Verlag, Mannheim; Leipzig; Wien; Zürich, 1992.

[KGK93] R. Kruse, J. Gebhardt und F. Klawonn. *Fuzzy-Systeme.* Teubner, Stuttgart, 1993.

[Kö90] M. Köhle. *Neurale Netze.* Springer, Wien; New York, 1990.

[Koh87] T. Kohonen. Adaptive, associative, and self-organizing functions in neural computing. *Applied Optics*, 26:4910–4918, 1987.

[Kos92] B. Kosko. *Neural Networks And Fuzzy Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Lin93] L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks.* Dissertation, Carnegie Mellon University, 1993.

[Mat92] M. J. Mataric. Integration of Representation Into Goal-Driven Behavior-Based Robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[MS93] T. Martinetz und K. Schulten. Topology Representing Networks. *Neural Networks*, 7(3):507–522, 1993.

[MT92] J. R. Millan und C. Torras. A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments. *Machine Learning*, 8:363–395, 1992.

[Neh92] U. Nehmzow. *Experiments in Competence Acquisition for Autonomous Mobile Robots.* Dissertation, University of Edinburgh, 1992.

[Nil69] N. J. Nilsson. A mobile Automaton: An Application of Artificial Intelligence Techniques. In *Proc. of the First International Joint Conference in Artificial Intelligence (IJCAI 69)*, Seiten 509–520, Washington D. C., 1969.

[NKK94] D. Nauck, F. Klawonn und R. Kruse. *Neuronale Netze und Fuzzy-Systeme.* Vieweg, Braunschweig/Wiesbaden, 1994.

[OW90] T. Ottmann und P. Widmayer. *Algorithmen und Datenstrukturen.* BI-Wiss.-Verlag, Mannheim; Wien; Zürich, 1990.

[PM92] T. J. Prescott und J. E. W. Mayhew. Obstacle Avoidance through Reinforcement Learning. *Advances in Neural Information Processing Systems*, 4:523–530, 1992.

[Rei94] P. Reignier. Fuzzy logic techniques for mobile robot obstacle avoidance. *Robotics and Autonomous Systems*, 12:143–153, 1994.

[RMS90] H. Ritter, T. Martinetz und K. Schulten. *Neuronale Netze: eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke.* Addison-Wesley, Bonn; München; Reading, Mass. [u. a.], 1990.

[RN94] G. A. Rummery und M. Niranjan. On-line Q-Learning using Connectionist Systems. Technical Report 166, Cambridge University Engineering Department, 1994.

[Roj93] R. Rojas. *Theorie der neuronalen Netze: eine systematische Einführung.* Springer, Berlin; Heidelberg, 1993.

[SE94] A. Standfuss und R. Eckmiller. To swing up an inverted Pendulum using stochastic real-valued Reinforcement Learning. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 655–658, 1994.

[Som95] G. Sommer. Verhaltensbasierter Entwurf technischer visueller Systeme. *Künstliche Intelligenz*, 1:42–45, 1995.

[Sut88] R. S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, 1988.

[Sut90] R. S. Sutton. Reinforcement Learning Architectures for Animats. In *Proc. of the First International Conference on Simulation of Adaptive Behavior*, Seiten 288–296. MIT Press, 1990.

[Thr93] S. Thrun. Exploration and Model Building in Mobile Robot Domains. In *Proc. of the IEEE International Conference on Neural Networks*, Seiten 175–180, San Francisco, März 1993.

[Thr95] S. Thrun. An Approach to Learning Mobile Robot Navigation. *Robotics And Autonomous Systems*, Special Issue on Robot Learning, 1995. Erhältlich unter: http://www.cs.cmu.edu/~thrun/papers/thrun.ras.html.

[TM93] S. Thrun und T. M. Mitchell. Lifelong Robot Learning. IAI-TR-93-7, University of Bonn, Dept. of Computer Science III, 1993.

[Tra91a] Transitions Research Corporation (TRC), Danburry, Connecticut. *LABMATE User Manual*, Release 5.21L-f. Auflage, 1991.

[Tra91b] Transitions Research Corporation (TRC), Danburry, Connecticut. *Proximity Subsystem User Manual*, Release 4.6H. Auflage, 1991.

[TS93] S. Thrun und A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proc. of the Fourth Connectionist Model Summer School*, Lawrence Erblaum, Hillsdale, NJ, 1993.

[Wat89] C. Watkins. *Learning from Delayed Rewards*. Dissertation, King's College, Cambridge University, UK, 1989.

[Wat92] C. Watkins. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[Wil92] R. J. Williams. Simple Statistical Gradient-Following Algorithms for connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

Index