Eine Verhaltenssteuerung für autonome mobile Roboter auf der Basis von Potentialfeldern

Diplomarbeit an der Universität Bremen
Fachbereich Mathematik und Informatik
Arbeitsgruppe von Prof. Dr. Bernd Krieg-Brückner

Tim Laue

5. Januar 2004

Gutachter: Dr. Thomas Röfer
Prof. Dr. Bernd Krieg-Brückner

Betreuer: Dr. Thomas Röfer

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Aufbau und Inhalte der Diplomarbeit

Im Rahmen dieser Diplomarbeit ist eine Verhaltensarchitektur für autonome mobile Roboter entworfen und implementiert worden. Innerhalb dieser Architektur ist es möglich, die Bewegungen eines Roboters sowie die Manipulation externer Objekte zu modellieren. Die möglichen Handlungen eines Roboters werden dabei mit Hilfe von Potentialfeldern beschrieben, bewertet und ausgewählt.

Die Arbeit gliedert sich in insgesamt sechs Kapitel sowie zwei Anhänge, deren Inhalt hier kurz dargestellt wird.

In der *Einleitung* werden das Prinzip und die Anwendungsmöglichkeiten von Potentialfeldern beschrieben. Aus bisherigen Veröffentlichungen zu diesem Thema werden die Motivation und die Ziele dieser Arbeit abgeleitet. Die Einleitung schließt mit einer Einführung in die für die Implementierung verwendete Testplattform.

Das Kapitel *Architektur* gibt einen Überblick über das Gesamtsystem und beschreibt das Zusammenspiel der in den folgenden Kapiteln beschriebenen Elemente sowie die Mechanismen zur Auswahl einzelner Verhalten. Ebenso wird das Verfahren der Verhaltensmodellierung über XML-Dateien erläutert.

Zentraler Bestandteil aller Verfahren sind die *Objekte in Potentialfeldern*. Deren Rolle und genauer Aufbau werden daher in einem eigenen Kapitel beschrieben.

Im Kapitel *Bewegung im Potentialfeld* wird die Umsetzung des klassischen Verfahrens zur Bewegungsplanung in einem Potentialfeld beschrieben. Dieses wurde unter anderem um Relativbewegungen sowie um Algorithmen zur Behandlung lokaler Minima, welche ein zentrales Problem des Potentialfeld-Ansatzes darstellen, erweitert.

Neben der Eigenbewegung ist die *Aktionsauswahl* die zweite Möglichkeit der Verhaltensmodellierung. Es wird die Auswahl von Aktionen zur Veränderung des eigenen Zustands und der Manipulation von Objekten in der Umgebung ebenso wie ein Verfahren zur Suche möglichst optimaler Handlungssequenzen vorgestellt.

Die Diplomarbeit endet mit dem Kapitel *Anwendungen, Ergebnis und Bewertung*, in dem die bisherigen Einsätze des Verfahrens beschrieben, analysiert und letztendlich auch bewertet werden. Neben einem Fazit der gesamten Diplomarbeit wird zudem ein Ausblick auf mögliche weitere Arbeiten gegeben.

In den verbleibenden zwei Anhängen, der *Referenz der Programmierschnittstelle* und der *Referenz der XML-Konfigurationsdateien*, sind die zur Verwendung der implementierten Verhaltensarchitektur benötigten Schnittstellen und sämtliche Möglichkeiten der Konfiguration ausführlich dokumentiert.

1.2 Potentialfelder

Da die Verhaltenssteuerung auf Potentialfeldern basiert, soll vorab eine kleine Einführung in deren Struktur und Anwendungsmöglichkeiten gegeben werden.

1.2.1 Mathematische Grundlagen

Ein Potentialfeld, auch *konservatives Feld* genannt, ist eine spezielle Form eines (ebenen oder räumlichen) Vektorfeldes. Einem solchen Feld ist stets eine *Potentialfunktion* φ zugeordnet, die auch als das *Potential* des Feldes bezeichnet werden kann. Die zentrale Eigenschaft von Potentialfeldern ist die Möglichkeit, das Vektorfeld $\vec{F}$ an einer beliebigen Position aus dem Gradienten der Potentialfunktion zu berechnen.

In einem ebenen Feld gilt also immer:

$$\vec{F}\,(x;y) = grad\;\varphi\,(x;y) \tag{1.1}$$

Potentialfelder sind von großer Bedeutung in Naturwissenschaften und Technik. Sie dienen der Beschreibung und mathematischen Behandlung diverser physikalischer Phänomene; beispielsweise handelt es sich beim Magnetfeld eines stromdurchflossenen linearen Leiters oder auch beim elektrischen Feld eines geladenen Drahtes um Potentialfelder.

Ausführlichere Informationen zu Potentialfeldern finden sich unter anderem in [Papula, 1999] sowie in [Bronstein et al., 1999].

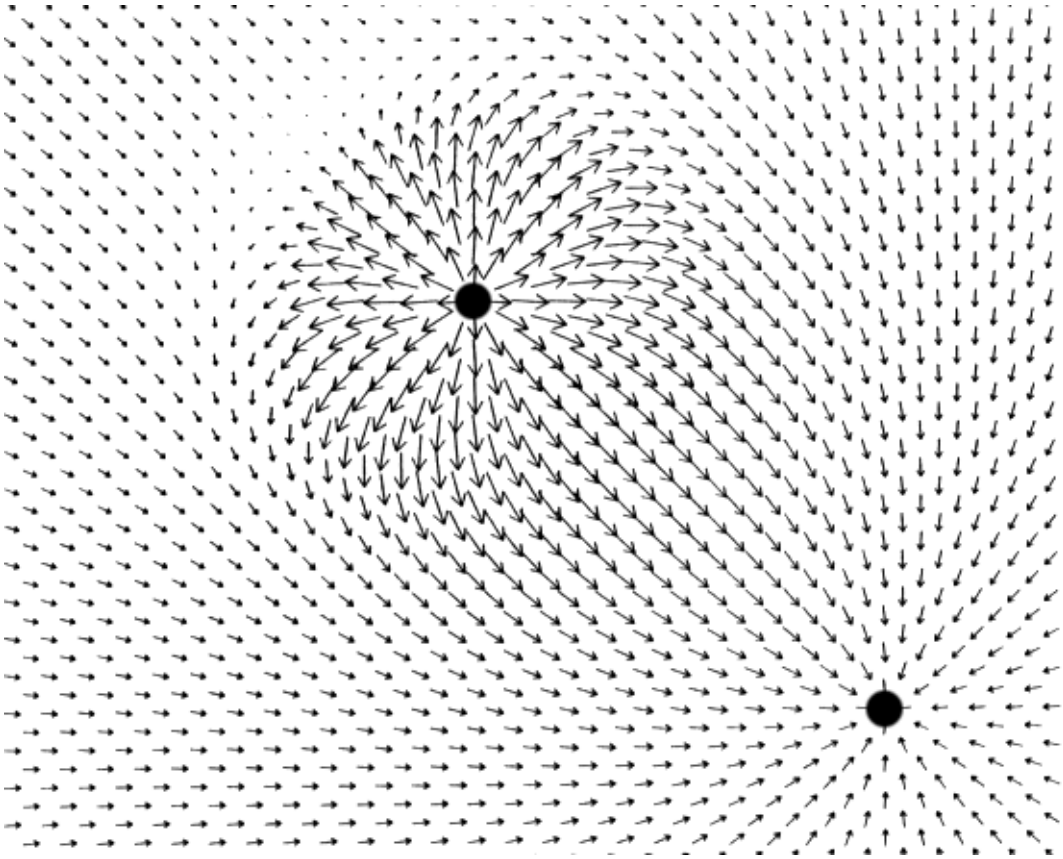


Abbildung 1.1: *Beispiel eines Potentialfelds. Das Gesamtfeld führt zur Zielposition (unten rechts) um ein Hindernis herum. Links oberhalb des Hindernisses befindet sich ein lokales Minimum. Das Bild wurde, leicht verändert, aus [Arkin, 1998] entnommen.*

1.2.2 Bewegungsplanung für Roboter

Im Bereich der Robotik wurden Potentialfelder erstmals von [Khatib, 1986] zur Realisierung einer Echtzeit-Hindernisvermeidung sowohl für mobile Roboter als auch für Roboterarme eingesetzt. Die Grundidee besteht darin, jedem Objekt in der Umgebung des Roboters ein Potentialfeld zuzuordnen, wobei der Zielposition ein attraktives, also zum Ziel hinführendes, Feld und allen anderen Objekten jeweils repulsive, also von den Objekten wegführende, Felder zugeordnet werden. Diese Felder werden zu einem Gesamtfeld überlagert (siehe Abbildung 1.1). Durch die Berechnung des Gradienten ist es dem Roboter nun möglich, an jeder beliebigen Position die Richtung des Vektorfeldes und somit eine Bewegungsrichtung zu bestimmen, in der er kollisionsfrei zum Ziel gelangen kann. Dieses Verfahren wird in [Latombe, 1991] ebenfalls sehr ausführlich beschrieben.

In ähnlicher Art und Weise, aber eher aus Sicht der verhaltensbasierten Robotik als aus Sicht der klassischen Bewegungsplanung, werden Potentialfelder von [Arkin, 1989, Arkin, 1998] im Rahmen einer *Motor-Schemas* genannten Steuerungsarchitektur für autonome mobile Roboter behandelt.

Es existieren eine Vielzahl von Erweiterungen des Potentialfeld-Ansatzes. Im Kontext dieser Diplomarbeit sind von diesen vor allem die Arbeiten von [Reif and Wang, 1995] und [Balch and Arkin, 1999] interessant, die sich mit Gruppen von autonomen Robotern beschäftigen, die mit Hilfe von Potentialfeldern gemeinsam Aufgaben bearbeiten und Formationen bilden.

Ein Potentialfeldern inhärentes Problem sind lokale Minima. Dies sind Bereiche in einem Vektorfeld, in denen die Längen der Vektoren gegen Null konvergieren, ein Roboter somit keine Bewegungsrichtung mehr bestimmen kann und sozusagen an einer Position „gefangen“ ist. Ein solches Minimum ist auch in Abbildung 1.1 zu sehen. Für dieses Problem existiert keine allgemeine Lösung; eine ausführliche Analyse findet sich in [Koren and Borenstein, 1991].

1.2.3 Aktionsauswahl mit Potentialfeldern

Eine weitere Anwendung von Potentialfeldern ist die Bewertung der Umgebung eines Roboters und daraus folgend die Auswahl bestimmter Aktionen. Hierzu wurden bisher allerdings vergleichsweise wenige Arbeiten veröffentlicht.

Statt eines Vektorfeldes wird im Allgemeinen einzelnen Objekten lediglich eine Potentialfunktion zugeordnet, so dass an jeder beliebigen Position der Umgebung das Gesamtpotential bestimmt werden kann. Sowohl in [Meyer and Adolph, 2003] als auch in [Ball and Wyeth, 2004] wird beispielsweise die gesamte Umgebung mit einem Raster überzogen, das Potential jeder Rasterzelle berechnet und somit die günstigste Region bestimmt, welche das Ziel einer nächsten Aktion sein wird.

Einen davon abweichenden Ansatz präsentieren [Johannson and Saffiotti, 2002]. Unter dem, an die Anwendungen von Potentialfeldern in der Physik erinnernden, Titel *Electric Field Approach* wird ein Verfahren vorgestellt, dass Aktionen bewertet, deren Auswirkungen auf die Umgebung zuvor von einem externen Planer berechnet worden sind.

1.2.4 Anwendung von Potentialfeldern

Aufgrund der Möglichkeit der Echtzeit-Bewegungsplanung in einem kontinuierlichen Raum sind Potentialfelder ein oft eingesetztes Verfahren für autonome Roboter. Dies gilt besonders für die *RoboCup*-Domäne, die auch die Testumgebung

für diese Diplomarbeit darstellt und ausführlich in Abschnitt 1.4 beschrieben wird. Dort finden sich auch einige Beispiele für die Aktionsbewertung durch Potentialfelder. Neben den zuvor schon erwähnten [Johannson and Saffiotti, 2002], [Meyer and Adolph, 2003] und [Ball and Wyeth, 2004] werden Potentialfelder unter anderem auch von [Weigel et al., 2002], [Kiat, 2004] und [Vail and Veloso, 2003] verwendet.

1.3 Motivation und Ziele

1.3.1 Motivation

Wie zuvor beschrieben werden Potentialfelder in der Robotik oft und zu verschiedenen Zwecken eingesetzt. Hierbei handelt es sich jedoch im Allgemeinen um spezielle Implementierungen, in denen das Potentialfeld jeweils für einen einzelnen Aspekt eines Gesamtverhaltens genutzt wird. Weitreichendere Ansätze, wie beispielsweise [Johannson and Saffiotti, 2002], deren gesamtes Verhalten auf einer Bewertung durch Potentialfelder basiert, sind auf weitere Komponenten wie einen Planer angewiesen.

Was in der Robotik im Großen zu beobachten ist, spiegelt sich auch im Kleinen im studentischen RoboCup-Projekt der Universität Bremen wider. In nahezu allen dort entwickelten Mannschaften, die in verschiedenen Ligen des RoboCup antreten, finden sich Potentialfelder, jeweils in eigenen Implementierungen zur Unterstützung spezieller Teile des Verhaltens.

Aus diesem Zustand ergeben sich die Ziele dieser Diplomarbeit.

1.3.2 Ziele

Das Gesamtziel der Diplomarbeit ist die Integration und Erweiterung bisheriger Ansätze in einer abstrakten Architektur, die es, ohne weitere externe Mechanismen, ermöglicht, das Verhalten beliebiger mobiler Roboter zu modellieren. Dies beinhaltet folgende Einzelaspekte:

Umsetzung und Erweiterung bisheriger Verfahren

Zur Bewegungssteuerung soll der klassische Ansatz aus [Khatib, 1986, Latombe, 1991, Arkin, 1998] verwendet werden. In diesen integriert wird ein Modell zur Beschreibung von Relativbewegungen in Anlehnung an [Reif and Wang, 1995] und [Balch and Arkin, 1999]. Zur Überwindung auftretender lokaler Minima wird

ein neues effizientes Suchverfahren zur Wegplanung vorgestellt. Darüber hinaus wird die Bewegungssteuerung ergänzt durch mehrere kleinere Erweiterungen.

Als effizientestes Verfahren zur Aktionsauswahl erscheint der *Electric Field Approach* von [Johannson and Saffiotti, 2002]. Dieser wird erweitert um ein internes Verfahren zur Beschreibung und Planung von Aktionen zur Manipulation der Umgebung, sowie um die Möglichkeit der vorausschauenden Suche nach möglichst erfolgversprechenden Aktionssequenzen.

Modellierung komplexeren Verhaltens

Die zuvor beschriebenen Einzelverfahren sollen in einer gemeinsamen Architektur zusammengefasst werden. Innerhalb dieser Architektur soll es dann möglich sein, eine Menge einzelner Verhalten zu definieren, die unabhängig voneinander arbeiten, jedoch eine gemeinsame Modellierung der Umgebung nutzen. Zur tatsächlichen Ausführung wird lediglich das zum jeweiligen Zeitpunkt am geeignetsten erscheinende Verhalten ausgewählt. Hierzu sind entsprechende Auswahl- und Bewertungsmechanismen in die Architektur zu integrieren.

Aus der Menge dieser konkurrierenden Verhalten ergibt sich dann ein komplexeres Gesamtverhalten, siehe hierzu auch [Brooks, 1991].

Übertragbarkeit und Plattformunabhängigkeit

Die Verhaltenssteuerung soll nicht auf ein spezielles Robotersystem oder eine einzelne Domäne beschränkt sein. Sämtliche Schnittstellen, Verfahren und Möglichkeiten der Modellierung werden derart abstrakt gehalten, dass die Architektur möglichst problemlos auf beliebigen Plattformen eingesetzt werden kann.

Ebenso soll die Implementierung in C++ plattformunabhängig sein, so dass die Verhaltenssteuerung theoretisch auf jedem System, für das ein entsprechender Übersetzer existiert, lauffähig ist.

Externe Modellierung

Da die Modellierung eines Verhaltens mit Potentialfeldern auf der Beschreibung der Umwelt basiert und keinerlei Abläufe oder Algorithmen spezifiziert werden müssen, liegt es nahe, diese Modellierung von der Implementierung der verwendeten Verfahren zu trennen.

In der hier vorgestellten Architektur sollen sämtliche Verhalten in externen Konfigurationsdateien beschrieben werden. Die dazu verwendete Beschreibungssprache wird in XML definiert.

Integration einer probabilistischen Modellierung

In der mobilen Robotik sind probabilistische Lokalisierungsverfahren, wie beispielsweise die Monte-Carlo-Lokalisierung [Fox et al., 1999], derzeit sehr populär. Dabei wird im Allgemeinen nicht von einer möglichen Position des Roboters oder anderer Objekte ausgegangen sondern eine Vielzahl von Hypothesen verfolgt, von denen dann zumeist lediglich die wahrscheinlichste ausgewählt und aus Sicht der Verhaltenssteuerung als einzig gültige betrachtet wird.

Ebenso exisitiert bei Aktionen eines Roboters in der realen Welt, zum Beispiel nach dem Schuss eines Balls, oftmals keine gesicherte Kenntnis des Folgezustands. Diese Tatsache ignorierend wird zumeist vom wahrscheinlichsten beziehungsweise gewünschten Verlauf der Aktion ausgegangen. Es können jedoch durch Versuchsreihen Wahrscheinlichkeitsverteilungen möglicher Endzustände bestimmt werden.

An diese beiden Tatsachen anknüpfend soll die Modellierung der Umwelt sowie der Aktionen in den Potentialfeldern sich nicht auf diskrete Positionen beschränken, sondern, falls vorhanden, mit Wahrscheinlichkeitsverteilungen arbeiten können. Bisherige Arbeiten zu diesem Thema sind nicht bekannt.

1.3.3 Einschränkungen

Neben den zuvor vorgestellten Zielen exisitieren weitere mögliche Funktionen oder Eigenschaften, die zwar in anderen Ansätzen enthalten sind, hier jedoch explizit nicht behandelt werden sollen.

Handeln in höherdimensionalen Räumen

Die Verhaltenssteuerung beschränkt sich auf das Handeln in einem zweidimensionalen Raum. Der in [Khatib, 1986] erstmals vorgestellte Potentialfeld-Ansatz behandelt auch die Steuerung von Robotern in einem dreidimensionalen Raum am Beispiel eines Roboterarms. Dies wird auch in einigen neueren Veröffentlichungen aufgegriffen, beispielsweise in [Lin and Chuang, 2003]. Da sich jedoch die während der Erstellung der Diplomarbeit verwendeten Testplattformen ausschließlich in einer Ebene bewegen und kein Roboterarm oder ähnliches zur Verfügung steht, wurde die Architektur zunächst bewusst auf zwei Dimensionen beschränkt.

Eine mögliche Erweiterung der Implementierung auf drei Dimensionen wird in Abschnitt 6.4.1 aufgezeigt.

Parallelität

Im Gegensatz zu den in [Arkin, 1989] beschriebenen Motor-Schemas werden einzelne Verhalten zwar unabhängig voneinander berechnet, jedoch nicht parallel. Dies wäre von Nutzen bei Systemen mit mehreren Sensoren, die stark zeitversetzte Messungen liefern sowie bei zu langen Berechnungszeiten einzelner Verhalten. Ein entscheidender Nachteil ist allerdings die mangelhafte Portierbarkeit paralleler Lösungen. Auf der verwendeten Testplattform ist eine entsprechende technische Umsetzung sehr aufwändig und würde die Performanz des Systems stark beeinträchtigen.

Des Weiteren ist im implementierten Ansatz die Berechnungszeit einzelner Verhalten, wie in Abschnitt 6.3 ausführlicher dargelegt wird, relativ gering.

Generierung von Trajektorien

Die umgesetzte Bewegungsplanung erzeugt Richtungsvektoren und berücksichtigt dabei lediglich holonome Roboter. Für nicht-holonome Roboter, wie beispielsweise Fahrzeuge mit Antriebs- und Lenkachse, wäre zusätzlich die Generierung von Trajektorien notwendig, um bestimmte Konfigurationen im Raum ineinander zu überführen. Ein Ansatz, der eine Bewegungsplanung mit Hilfe von Potentialfeldern für nicht-holonome Fahrzeuge vorstellt, ist in [Damas et al., 2003] zu finden.

Dynamische Umgebung

Während der Berechnung der Auswirkungen einzelner Aktionen oder auch während der Wegplanung wird davon ausgegangen, dass die Umgebung statisch ist. Auf etwaige Geschwindigkeiten oder Bewegungsrichtungen anderer Objekte wird somit keine Rücksicht genommen. Dies ist eine aus Gründen der Performanz gemachte Prämisse. Des Weiteren ist die Sensorik vieler Robotersysteme lediglich in der Lage, Objekte in der Umgebung zu identifizieren, jedoch nicht deren Bewegungsrichtung und Geschwindigkeit zuverlässig zu berechnen. Dies gilt besonders für die verwendete Testplattform.

Die geringe Berechnungszeit der Verhalten, die ein relativ schnelles Reagieren auf Veränderungen der Umwelt ermöglicht, sowie mehrere Möglichkeiten, durch die Modellierung einzelner Objekte deren Bewegungen zu antizipieren, schaffen jedoch einen gewissen Ausgleich für diese Einschränkung.

1.4 Die Testplattform

Da sich in den folgenden Kapiteln einige Beispiele auf die konkrete Implementierung und den möglichen Einsatz der Verhaltenssteuerung beziehen werden, soll hier ein kurze Einführung in die verwendete Plattform, ein Sony ERS-210 Roboter, und die Testumgebung, Fußball spielen im *RoboCup*, gegeben werden.

1.4.1 Der RoboCup

Der RoboCup ist eine internationale Initiative zur Förderung der Forschung in den Bereichen *Künstliche Intelligenz* und *autonome mobile Roboter*. Roboterfußball wird als ein standardisiertes Problem verwendet, an dem sich die Ergebnisse aus verschiedenen Forschungsrichtungen direkt vergleichen lassen. Der Dachverband des RoboCup ist die RoboCup Federation [RoboCup Federation, 2003]. Sie wurde von [Kitano et al., 1995] ins Leben gerufen und veranstaltet seit 1997 jährliche RoboCup-Weltmeisterschaften.

Derzeit spielen autonom agierende Roboter in mehreren verschiedenen Ligen Fußball. Die einzelnen Ligen unterscheiden sich durch die Größe und den Aufbau der Roboter, die Anzahl der Spieler, die Spielfeldgröße und das Anforderungsprofil an die Roboter.

Neben dem Fußball-Szenario existieren noch die beiden Bereiche RoboCup Rescue und RoboCup Junior.

1.4.2 Die Sony Four-legged Robot League

Eine der Roboter-Ligen im Rahmen des RoboCup ist die *Sony Four-legged Robot League*, im Folgenden als *Sony-Liga* bezeichnet. Das besondere an dieser Liga ist die einheitliche Roboter-Plattform. Sämtliche Teams benutzen die gleiche Hardware (siehe Abschnitt 1.4.3), der Fokus liegt somit auf der Entwicklung leistungsfähiger Software. Die Roboter können über Sony bezogen werden und dürfen von den Mannschaften nicht modifiziert werden.

Sowohl das Spielfeld als auch die allgemeinen Spielregeln werden jährlich verändert, um den wissenschaftlichen Anspruch zu erhöhen, dem Vorbild des menschlichen Fußballs näher zu kommen, dem technischen Fortschritt durch verbesserte Roboter Rechnung zu tragen oder auch um ein attraktiveres Spielgeschehen zu fördern. Detaillierte aktuelle Informationen über die Umgebung in der Sony-Liga finden sich auf deren offiziellen Internet-Seiten [Sony Corporation, 2003b]. Es folgt eine kurze Zusammenfassung der wichtigsten Daten auf dem Stand der Weltmeisterschaft 2003:

Abbildung 1.2: *Ein Spiel in der Sony-Liga beim RoboCup 2002 in Fukuoka*

Es spielen stets zwei Mannschaften mit jeweils vier Robotern gegeneinander auf einem 4,20 $\times$ 2,70 m großen Feld aus grünem Teppichboden, welches von zwei Toren und einer Bande umgeben ist. Rund um das Feld befinden sich sechs Zylinder mit jeweils eindeutigen Farbkodierungen. Sie dienen, ebenso wie die verschiedenen Farben der Tore, als Anhaltspunkte für die Selbstlokalisierung der Roboter. Zusätzlich ist das Feld mit weißen Linien in verschiedene Bereiche unterteilt. Gespielt wird mit einem orangenen Plastikball. Die Abbildung 1.2 zeigt ein Spiel bei einer Weltmeisterschaft.

Die Roboter dürfen während des Spiels miteinander kommunizieren um Informationen auszutauschen oder sich untereinander zu koordinieren. Es dürfen keinerlei Berechnungen auf einem externen Rechner durchgeführt werden oder die Roboter über ein zentrale Instanz koordiniert werden. Jeder Roboter muss autonom entscheiden und handeln.

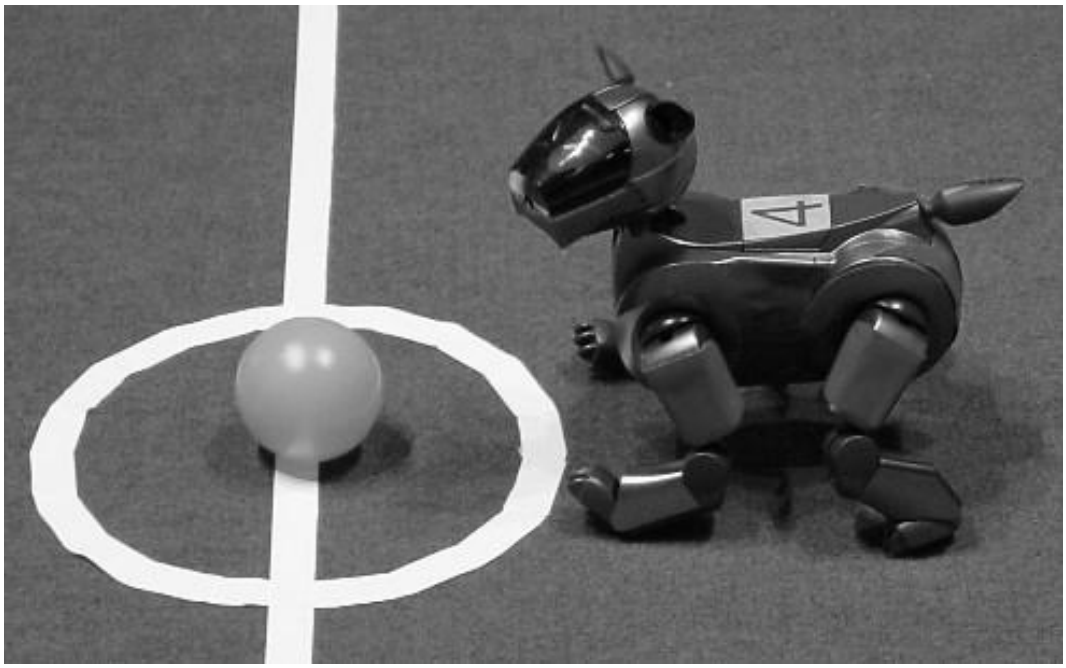


Abbildung 1.3: *Ein Sony-Roboter während eines RoboCup-Spiels*

1.4.3 Der verwendete Roboter

Verwendet wird eines der Robotermodelle, die von Sony zusammen mit spezieller Software unter dem Namen *AIBO* entwickelt worden sind. Es handelt sich um ein Produkt aus der Unterhaltungselektronik, das als eine Art künstliches Haustier konzipiert wurde. Der Roboter besitzt vier Beine und ist mit diesen in der Lage, sich in der Ebene omnidirektional zu bewegen und zu rotieren. Die Umgebung kann über eine Vielzahl von Sensoren erfasst werden, der wichtigste ist eine am vorderen Ende des Kopfes integrierte CCD-Farbkamera.

Das jeweils technisch fortgeschrittenste Modell wird in der Sony-Liga eingesetzt. Dies war während der Implementierung dieser Diplomarbeit der *Sony ERS-210A*, im Folgenden als *Sony-Roboter* bezeichnet und zu sehen in Abbildung 1.3.

Seine Dimensionen betragen im normalen Stand circa $15 \times 28 \times 25$ cm (Breite / Länge / Höhe). Der Kern des Roboters verfügt neben einem Mikrocontroller über einen mit 400 Megahertz getakteten MIPS-Prozessor sowie über 32 Megabyte Hauptspeicher. Zur Kommunikation mit dem Roboter, beziehungsweise zwischen einzelnen Robotern, kann optional eine Funk-Netzwerkkarte (nach IEEE 802.11b) eingebaut werden.

Eigene Software für den Sony-Roboter kann auf einem externen PC entwickelt werden. Als Betriebssystem auf dem Roboter kommt das von Sony entwickelte Echt-

zeitbetriebssystem *Aperios* zusammen mit der Schnittstelle *OPEN-R* [Sony Corporation, 2003a] zum Einsatz.

1.4.4 Software-Umgebung

Die implementierte Verhaltenssteuerung wurde in das Gesamtsystems des *GermanTeam* [Röfer et al., 2004] integriert. Das GermanTeam ist eine Zusammenarbeit von vier Universitäten: der Humboldt Universität zu Berlin, der Universität Bremen, der Technischen Universität Darmstadt sowie der Universität Dortmund. Gemeinsam wird an einem modularen System gearbeitet, welches sämtliche, zur Steuerung eines Sony-Roboters benötigten, Bereiche abdeckt. Neben der Verhaltenssteuerung existieren unter anderem auch Module zur Bildverarbeitung, Selbstlokalisierung oder zur Steuerung der Bewegungen. In das System integriert sind ein Simulator sowie umfangreiche Möglichkeiten zur Kommunikation mit dem Roboter. Seit 2001 nimmt das GermanTeam regelmäßig am RoboCup in der Sony-Liga teil.

Der detaillierte Aufbau des Systems, die Verwendung der Verhaltenssteuerung sowie die damit erzielten Ergebnisse werden ausführlich im Abschnitt 6.1 beschrieben.

Kapitel 2

Architektur

Dieses Kapitel beschreibt die Einbettung der Verhaltensarchitektur in vorhandene Umgebungen, die Grundprinzipien der Verhaltensauswahl, den konkreten inneren Aufbau der Architektur sowie die Möglichkeiten der Verhaltensmodellierung.

2.1 Einbettung in ein Gesamtsystem

Um die Verhaltenssteuerung auf einem Roboter verwenden zu können, sind zwei Konvertierungsschritte nötig. Zum einen muss das Weltmodell des Roboters in die von den Potentialfeldern verwendete Repräsentation überführt werden und zum anderen muss das Ergebnisverhalten interpretiert und entsprechend zur Ausführung gebracht werden, wie in Abbildung 2.1 zu sehen ist. Die beiden dazu benötigten Konverter müssen für jede Plattform speziell implementiert werden.

2.1.1 Konvertierung des Weltmodells

Die innerhalb der Architektur verwendete Schnittstelle zur Repräsentation der Umwelt ist sehr allgemein gehalten, so dass die Daten aus Weltmodellen verschiedenster Robotersysteme adäquat überführt werden können.

Die Informationen über den Zustand eines Objektes der Umgebung werden in einem einzelnen Datentyp gekapselt, der direkt an die Verhaltenssteuerung weitergegeben werden kann. Folgende Daten werden hierzu für jedes Objekt benötigt:

Objektbezeichner Bei der Modellierung des Verhaltens wird jedem Objekt ein eindeutiger Bezeichner zugeordnet. Um die aktuellen Daten für ein Objekt zu übergeben, muss dieser zur internen Identifikation angegeben werden.

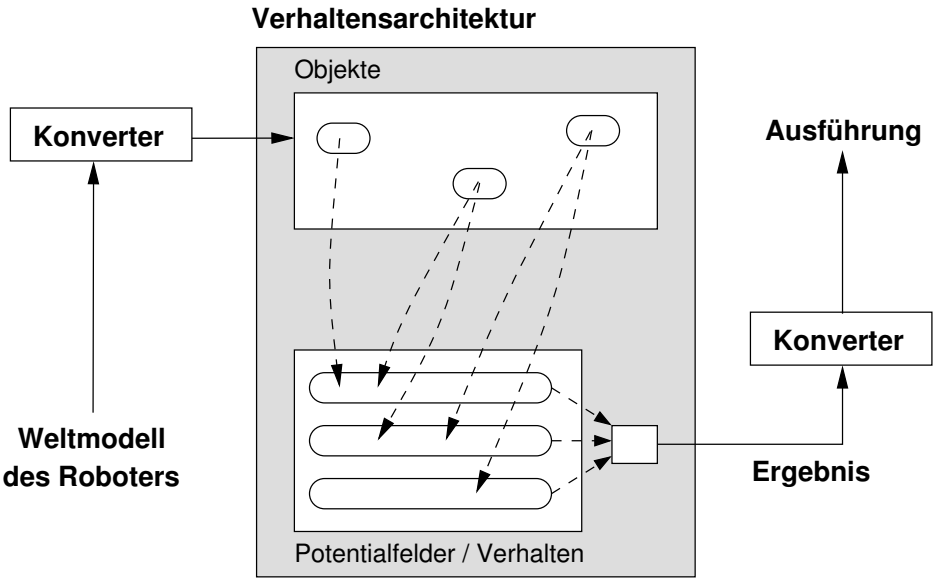


Abbildung 2.1: *Einbettung der Verhaltenssteuerung in ein Gesamtsystem.*

Pose In der Pose sind die Position, Rotation und Geschwindigkeit des Objekts zusammengefasst. Existiert zu dem Objekt eine Wahrscheinlichkeitsverteilung, so kann diese zusätzlich angegeben werden.

Status Der Status ist ein boolescher Wert, der beschreibt, ob zum aktuellen Zeitpunkt Informationen über das Objekt bekannt sind und diese verwendet werden können. Ist dies nicht der Fall, wird das Objekt intern deaktiviert und für keinerlei Berechnungen verwendet.

Die Eigenposition des Roboters wird separat behandelt, dabei entfallen der Objektbezeichner und der Status. Lediglich die Pose des Roboters wird an die Verhaltenssteuerung übergeben.

Bei Systemen, die über Lokalisierungsverfahren ein Weltmodell aufbauen, welches absolute Positionen aller Objekte der Umgebung einschließlich der des Roboters enthält, wie beispielsweise die verwendete Testplattform, können deren Daten, sofern die Objekte innerhalb der Verhaltenssteuerung ebenfalls modelliert worden sind, direkt vom Konverter übernommen werden.

Ein Einsatz in Systemen, die über keinerlei Verfahren zur Objektidentifizierung und -lokalisierung verfügen, ist allerdings ebenso möglich. Hierzu muss deren Eigenposition auf den Koordinatenursprung gesetzt, Sensorereignissen jeweils ein Objekt zugeordnet sowie die relativen Messdaten der Sensorik in die Objektbeschreibung eingesetzt werden.

Eine detailliertere Beschreibung des verwendeten Datentypen findet sich in der Referenz in Abschnitt A.3.3.

2.1.2 Interpretation des Ergebnisses

Das Ergebnis der Verhaltenssteuerung ist wiederum in einem einzelnen Datentyp zusammengefasst, die wichtigsten Informationen sind folgende:

Verhaltensname Der Name des ausgewählten Verhaltens, entsprechend der in der Modellierung angegebenen Bezeichnung.

Subaktion Ein Verhalten kann, wie später in Abschnitt 5.3.2 beschrieben wird, mehrere mögliche Unterverhalten haben. In einem solchen Fall wird der entsprechende Bezeichner ebenfalls vermerkt.

Bewegungsparameter Die Parameter beinhalten eine Bewegungsrichtung, eine Geschwindigkeit sowie einen Winkel, um den sich der Roboter drehen soll.

Darüber hinaus enthält das Ergebnis noch einige weitere, in den meisten Fällen nicht benötigte, Detailinformationen, unter anderem einen Zeitstempel und die Bewertung des Ergebnisses. Genaueres hierzu findet sich in Abschnitt A.3.4.

Für den Fall, dass keines der Verhalten ausgeführt werden kann, ist eine entsprechende eindeutige Kennzeichnung des Ergebnisses vorgesehen.

Der Konverter muss nun die Verhaltensnamen auf tatsächliche Aktionen des Roboters abbilden. Dies können weitere komplexe Einzelverhalten aber auch direkte Aufrufe der Aktuatorik sein. Ebenso muss darüber entschieden werden, ob die Bewegungsparameter für das jeweilige Verhalten relevant sind und weiterverarbeitet werden sollen oder ob diese, vielleicht auch nur teilweise, ignoriert werden können.

2.2 Verhaltensauswahl

2.2.1 Verwendung mehrerer getrennter Potentialfelder

Das Potentialfeld-Verfahren ist nach [Arkin, 1998] das klassische Beispiel für sich überlagernde Verhalten. Innerhalb eines Gesamtfeldes können durch die Zuordnung einzelner Felder zu Objekten oder Messungen in der Umgebung mehrere Verhalten gleichzeitig, also einander überlagernd, ausgeführt werden. Dies kann zum Beispiel das Laufen zu einem Ball unter gleichzeitiger Vermeidung anderer Roboter sein.

Diesem Schema folgend ist allerdings schnell eine Grenze des Ansatzes erreicht, was sich an einem Szenario aufzeigen lässt, in dem ein Roboter gleichzeitig mehrere Ziele verfolgt. So bewegt sich ein Torwart-Roboter in einem Fußballspiel stets zu zwei verschiedenen Zielen: Er muss sich in seinem Tor positionieren, aber auch zu einem Ball in Tornähe laufen um diesen klären zu können. Ordnet man nun die beiden attraktiven Felder von Tor und Ball einem gemeinsamen Gesamtfeld zu, kommt es zu ungünstigen Überlagerungen. Im schlimmsten Fall existiert ein Bereich, in dem sich die Felder vollständig aufheben, der Roboter bewegt sich dann in ein lokales Minimum und kann keines der beiden Ziele erreichen.

Dieses Problem könnte durch einen externen Mechanismus, der das jeweils günstigste Zielobjekt auswählt und andere Ziele gegebenenfalls deaktiviert, behoben werden. Da die Verhaltenssteuerung jedoch eigenständig arbeiten soll, wird ein anderer Weg gewählt. Für verschiedene Bewegungsverhalten, also beispielsweise ein Verhalten für die Positionierung im Tor sowie eines für die Bewegung zum Ball, sollen verschiedene voneinander unabhängige Potentialfelder verwendet werden können. Zur späteren Ausführung wird durch ein internes Bewertungsverfahren das jeweils beste ausgewählt. Dieses Vorgehen schließt selbstverständlich die eingangs erwähnten Möglichkeiten der Überlagerung innerhalb eines Feldes nicht aus.

Des Weiteren liegt der Bewertung von Aktionen ein abweichendes Berechnungsschema zu Grunde. Deshalb sind diese ohnehin in separaten Feldern zu behandeln. Hierbei ist es allerdings durchaus möglich, innerhalb eines Feldes zwischen mehreren Aktionen zu wählen, wie in Abschnitt 5.3.2 gezeigt wird.

2.2.2 Maximum-Auswahl-Architektur

Um nun die einzelnen Verhalten in einer gemeinsamen Architektur zusammenzufassen ist die *Maximum-Auswahl-Architektur* aus [Arkin, 1998] gewählt worden, die auf den Arbeiten von [Maes, 1989] basiert.

Wie in Abbildung 2.2 zu sehen, arbeitet dabei eine beliebige Anzahl von Verhalten unabhängig voneinander und erzeugt eine Menge von Ergebnissen $\{v_1, \ldots, v_n\}$. Zur Ausführung kommt letztlich lediglich ein einziges Verhalten V. Hierzu wird das Verhalten v_i (mit $i \in \{1, .., n\}$) ausgewählt, welches den höchsten Aktivierungswert $a(v_i)$ hat[1]. Dieser Wert wird von jedem Verhalten zusammen mit dem Ergebnis berechnet. Das zu verwendende Berechnungsverfahren muss für jedes Verhalten explizit angegeben werden, die verschiedenen Möglichkeiten werden in entsprechenden

[1] An dieser Stelle sei darauf hingewiesen, dass bei der im Rahmen dieser Diplomarbeit erfolgten Umsetzung der Architektur, stets den *niedrigsten* erzeugten Werten die *höchste* Aktivierung zugeordnet ist. Dieser Zustand ergibt sich aus der Struktur der Potentialfelder und verändert die Funktionsweise der Architektur nicht.

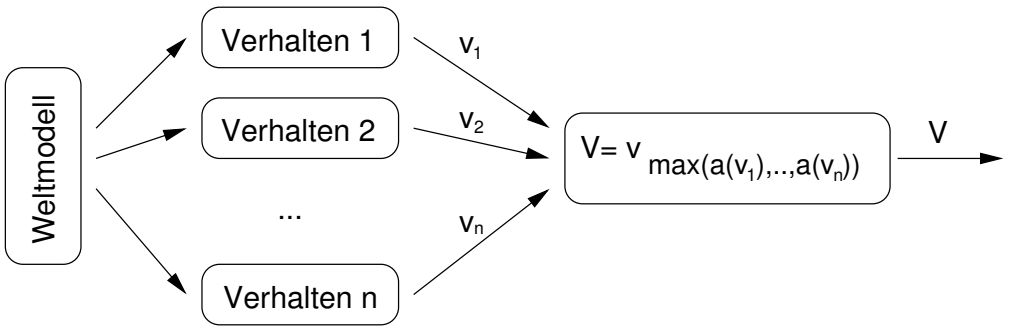


Abbildung 2.2: *Die Maximum-Auswahl-Architektur.*

Abschnitten der Kapitel zur Bewegungssteuerung (4.1.4) und zur Aktionsauswahl (5.1.5) beschrieben.

2.2.3 Verfeinerung der Verhaltensauswahl

Die Architektur sieht zunächst zwar keine feste Hierarchie oder das gegenseitige Blockieren einzelner Verhalten wie in der *Subsumption-Architektur* [Brooks, 1986] vor, sämtliche Verhalten arbeiten gleichberechtigt nebeneinander. Es wurden jedoch mehrere Mechanismen implementiert, die verwendet werden können, um das Zusammenspiel der Verhalten detaillierter zu modellieren. Standardmäßig sind sämtliche Verfahren deaktiviert, so dass rein reaktiv immer das Verhalten mit der augenblicklich besten Bewertung ausgewählt wird.

Kombination von Verhalten

Ein Verhalten V_A kann mit mehreren weiteren Verhalten V_{B_1} bis V_{B_n} derart kombiniert werden, dass immer wenn V_A ausgewählt wird, das Ergebnis mit den Ergebnissen von V_{B_1} bis V_{B_n} kombiniert wird. Das Gesamtergebnis behält dabei den V_A zugeordneten Verhaltensnamen, die Bewegungsparameter werden jedoch über alle kombinierten Verhalten gemittelt. Eine naheliegende Anwendung dieses Verfahrens ist die Überlagerung von Freiheitsgraden; sie wird in Abschnitt 4.4.2 beschrieben.

Ebenso wird es hierdurch möglich, über verschiedene Aktionsfelder Situationen zu unterscheiden und diese jeweils mit einem speziellen Bewegungsverhalten zu kombinieren, welches die dazu passenden Bewegungsparameter bestimmt.

Blockierung und Beibehaltung einzelner Verhalten

Jedem Verhalten können mehrere Attribute zugewiesen werden, die dessen Aktivierung steuern. Diese Attribute sind jeweils zeitabhängig, wobei die Zeit sowohl in Millisekunden als auch in der Anzahl der Aufrufe der Verhaltenssteuerung beschrieben werden kann.

Ein Verhalten kann, nachdem es ausgewählt worden ist, für eine festgelegte Zeit seine *Aktivierung beibehalten*. Während dieses Zeitraums sind alle anderen Verhalten blockiert.

Ebenso kann bei jedem Verhalten, unabhängig davon, ob es ausgewählt worden ist, das *Ergebnis beibehalten* werden, so dass beispielsweise rechenaufwändigere Einzelverhalten nur in gewissen Zeitabständen neu ausgewertet werden müssen.

Einem Verhalten kann eine *maximale Aktivierungszeit* zugeordnet werden. Dadurch kann vermieden werden, dass einzelne Verhalten zu lange ausgeführt werden.

Um sicherzustellen, dass ein Verhalten, nachdem es seine maximale Aktivierungszeit erreicht hat, nicht wieder von Neuem ausgewählt wird, ist es möglich, dieses für eine Mindestzeit *nach der Ausführung zu blockieren*.

Auswahlmechanismen

Schwankungen im Weltmodell durch vereinzelte Fehlmessungen der Sensorik, die zu einem plötzlichen Auftauchen eines zuvor nicht wahrgenommenen Objektes beziehungsweise zu dessen Verschwinden führen, können sich direkt auf die Auswahl des aktuell günstigsten Verhaltens auswirken. Um die Auswirkung solcher Fehler zu mindern sind zwei Mechanismen hinzugefügt worden, die jeweils die Menge der n zuletzt erzeugten Ergebnisse puffern und aus dieser Menge das tatsächlich auszuführende Verhalten bestimmen.

Dies ist zum einen die Auswahl des *häufigsten aus n* Verhalten. Dabei wird das jüngste Ergebnis des häufigsten Verhaltens zurückgegeben.

Zum anderen kann als Bedingung angegeben werden, dass ein Verhalten *mindestens n-mal in Folge* ausgewählt worden sein muss. Es wird, solange kein neues Verhalten diese Bedingung erfüllt, stets das jüngste Ergebnis der letzten Folge zurückgegeben.

2.3 Innerer Aufbau der Architektur

Wie bereits erwähnt, enthält die Architektur eine Anzahl einzelner Verhalten, eine eigene Repräsentation der Umwelt sowie diverse Selektionsmechanismen. In Ab-

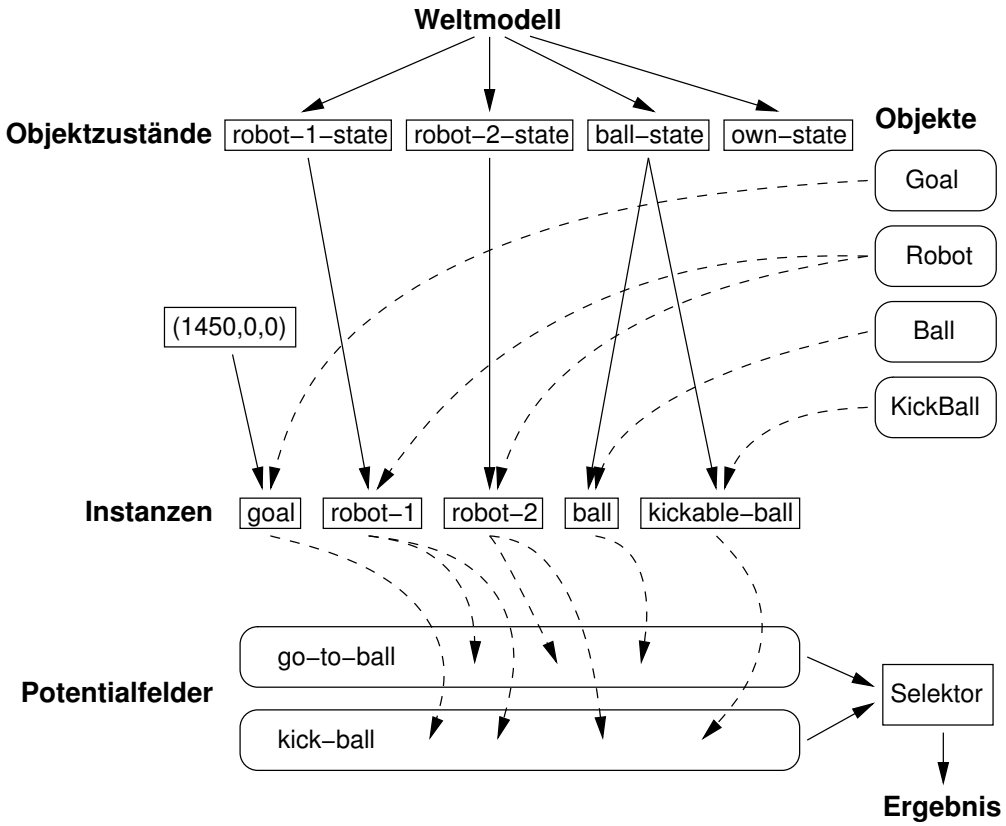


Abbildung 2.3: *Der innere Aufbau der Architektur am Beispiel eines einfachen Verhaltens zum Fußball spielen.*

bildung 2.3 wird der innere Aufbau der Architektur mit allen Elementen und deren Zusammenhängen dargestellt. Diese werden im Folgenden erläutert.

2.3.1 Objektzustände

Ein Objektzustand enthält die aus der Konvertierung des Weltmodells gewonnenen Daten über den akuellen Zustand eines Objektes der Umgebung. Er dient intern als Datenquelle für eine beliebige Anzahl von Instanzen.

2.3.2 Objekte und deren Instanzen

Im Rahmen der Verhaltensmodellierung werden *Objekte* und deren *Instanzen* getrennt betrachtet.

Objekte dienen dabei als Vorlagen oder Schablonen für Instanzen. Im Allgemeinen erzeugt jedes Objekt ein eigenes Potentialfeld, sämtliche Informationen über dessen Form sind Teil der Objektbeschreibung. Da Objekte einen sehr komplexen inneren Aufbau und eine große Menge möglicher Parameter haben, werden sie ausführlich in einem eigenen Kapitel (3) beschrieben.

Die Trennung wurde vorgenommen um die Modellierung zu vereinfachen. Sie ermöglicht die Wiederverwendung einer Objektbeschreibung für eine Vielzahl von Instanzen. So kann beispielsweise ein Roboter einer gegnerischen Mannschaft einmal modelliert und anschließend mehrfach instantiiert zu werden.

Instanzen beweglicher Objekte ist stets ein Objektzustand zugeordnet. Von diesem können dann zur Laufzeit die jeweils aktuellen Daten für die Berechnung bezogen werden. Ist der Roboter in der Lage, sich in seiner Umgebung zuverlässig zu lokalisieren und sind die Positionen einiger Objekte fest und zuvor bereits bekannt, wie beispielsweise die Tore auf einem Fußballfeld, so können diesen statt Objektzuständen feste Posen zugeordnet werden.

Für eine vereinfachte Zuordnung zu Potentialfeldern können Instanzen auch in Gruppen zusammen gefasst werden.

2.3.3 Potentialfelder

Ein Potentialfeld respäsentiert ein Verhalten. Das Feld selbst setzt sich dabei zusammen aus den Einzelfeldern der zugeordneten Objekte. Es existieren zwei Arten von Potentialfeldern: Bewegungsfelder (siehe Kapitel 4) und Aktionsfelder (siehe Kapitel 5). Bei jedem Aufruf der Verhaltenssteuerung werden sämtliche Verhalten ausgeführt und erzeugen ein Ergebnis zusammen mit einer Bewertung.

2.3.4 Selektor

Gemäß den Bewertungen der Ergebnisse der einzelnen Potentialfelder sowie den in Abschnitt 2.2.3 beschriebenen Verfahren, wählt der Selektor ein Verhalten aus und liefert dieses als Resultat der Verhaltenssteuerung zurück.

2.4 Modellierung des Verhaltens

Um ein Verhalten für einen Roboter mit Hilfe der vorgestellten Architektur zu modellieren, müssen die verwendeten Objektzustände, Objekte, Instanzen und Verhalten sowie deren Zusammenhänge vom Benutzer beschrieben werden. Gleiches gilt für die Parametrisierung der Verhaltensauswahl.

2.4.1 Externe Modellierung

Die Implementierung der Verhaltensarchitektur und die auf ihr basierenden Verhalten sind voneinander getrennt. Jedes Verhalten wird in einer eigenen Beschreibungsdatei spezifiziert, die vom System geladen werden kann. Dadurch bleibt die Verhaltensarchitektur als solche stets unverändert.

Falls eine Kommunikation mit dem Roboter besteht und Daten an diesen versendet werden können, wäre es somit möglich, das Verhalten zur Laufzeit durch die Übertragung einer neuen Beschreibungsdatei zu verändern. Eine Veränderung des eigentlichen Steuerungsprogramms und ein damit einhergehender Neustart des Roboters können vermieden werden.

Allerdings können zur Laufzeit keine neuen Verhalten oder Objekte mehr hinzugefügt werden. Es müssen also vorab sämtliche möglichen Verhalten und die vollständige, für das Verhalten relevante, Umwelt modelliert werden. Die Architektur enthält jedoch Mechanismen, die zeitweise unbenötigte Teile des Verhaltens deaktivieren und entsprechend wieder aktivieren können.

2.4.2 Modellierung in XML

Die Beschreibungssprache für die Verhaltensmodellierung ist in XML [World Wide Web Consortium, 2000] definiert worden. Mittels einer entsprechenden *DTD* (*Document Type Definition*) ist eine Menge von Auszeichnungselementen festgelegt worden, die es ermöglichen, eine klar strukturierte Beschreibung eines Verhaltens anzufertigen. Abbildung 2.4 zeigt zwei Ausschnitte aus einer Beschreibungsdatei, eine ausführliche Referenz findet sich in Anhang B.

Durch die Wahl von XML an Stelle eines eigenen Formates kann bei der Erstellung von Beschreibungen auf die Unterstützung einer Vielzahl von Texteditoren zurückgegriffen werden. Neben den weit verbreiteten Funktionen zur Einrückung und Syntax-Hervorhebung sind einige Editoren in der Lage, unter Verwendung der DTD, dem Benutzer bei der korrekten Strukturierung zu helfen, in dem sie an jeder Position der Beschreibung angeben, welche Elemente oder Attribute möglich sind oder benötigt werden.

Der Ansatz, XML zur Verhaltensmodellierung zu nutzen, ist auch in anderen Arbeiten verfolgt worden [Lötzsch et al., 2004].

2.4.3 Transformation und Validierung

Da nicht für jede Roboter-Plattform ein XML-Parser zur Verfügung steht, beziehungsweise dessen Verwendung in einem eingebetteten System Schwierigkeiten

a)

```
<object name="Opponent-Robot" type="repulsive">
    <asymptotic-function range="200"
                         at-zero="1000"
                         const-interval="1"/>
    <point-field/>
    <circle radius="90"/>
</object>
```

b)

```
<motionfield name="go-to-ball">
   <return-const value="-1"/>
   <include name="ball"/>
   <include name="own-penalty-area"/>
   <include-group name="all-robots"/>
</motionfield>
```

Abbildung 2.4: *XML-Beschreibungen a) eines Roboters und b) eines Verhaltens zur Bewegung zu einem Ball*

bereiten könnte, werden die XML-Dateien vor der Verarbeitung in ein anderes Format transformiert, welches vom Roboter eingelesen werden kann. Dieses Format hat das Ziel, möglichst einfach einzulesen zu sein, eine direkte Bearbeitung durch einen Benutzer ist nicht vorgesehen. Abbildung 2.5 zeigt Ausschnitte aus einer solchen Beschreibungsdatei.

Zur Umwandlung wurde die Transformationssprache XSLT [World Wide Web Consortium, 1999] verwendet, mit der sich XML-Dateien in beliebige andere Formate übersetzen lassen. Es ist ebenfalls möglich, mittels XSLT eine Dokumentation des Verhaltens in Form von HTML-Seiten erstellen zu lassen. Dies ist bereits ansatzweise realisiert worden.

Zusammen mit der Transformation wird eine Validierung des Verhaltens durchgeführt. Dabei werden zum einen dessen syntaktische Korrektheit geprüft und zum anderen die Übereinstimmung der verwendeten Elemente und deren Anordnung mit den Vorgaben der DTD verglichen. Dies ermöglicht eine sehr frühe Erkennung von syntaktischen und strukturellen Fehlern noch vor der Ausführung auf dem Roboter. Inhaltliche Fehler, wie beispielsweise ungültige Attributwerte, können allerdings erst zur Laufzeit beim Einlesen der Beschreibungsdatei entdeckt werden.

Zur Durchführung dieser Schritte wurden die beiden freien Programme `xsltproc`

a)

```
object;Opponent-Robot;repulsive;none;
asymptotic-function;200;1000;1;0;0;
point-field;
circle;100;true;
```

b)

```
motionfield;go-to-ball;false;true;-1;-1;field;calls;0;
calls;0;milliseconds;-1;const;-0.01;EOL;ball;
own-penalty-area;include-group;all-robots;EOL;
```

Abbildung 2.5: *Transformierte Beschreibungen a) eines Roboters und b) eines Verhaltens zur Bewegung zu einem Ball*

(Transformation) und `xmllint` (Validierung) von [Veillard, 2003] verwendet, die auf einer Vielzahl von Plattformen zur Verfügung stehen.

Kapitel 3

Objekte in Potentialfeldern

Dieses Kapitel behandelt die den Potentialfeldern zugeordneten Objekte. Es wird das Verfahren zur Berechnung des Feldes eines Objekts beschrieben, ebenso wie die drei Hauptbestandteile, aus denen es sich zusammensetzt: die geometrische Form, die mathematische Funktion sowie die Form des Feldes. Das Kapitel schließt mit einem Verfahren zur Berechnung des Feldes auf der Basis eines probabilistischen Weltmodells.

3.1 Berechnung des Feldes eines Objekts

Zu jedem Objekt kann zum einen die Wirkung des Vektorfeldes, welches auch als das Potentialfeld des Objektes bezeichnet wird, und zum anderen der Wert der Potentialfunktion, die nach [Johannson and Saffiotti, 2002] als eine elektrische Ladung interpretiert werden kann, jeweils an einer beliebigen Position in der Umgebung bestimmt werden.

Die Berechnungen sind abhängig von der geometrischen Form des Objektes, der Form des Feldes sowie von einer Funktion. Zu jedem dieser drei Teile sind mehrere Alternativen implementiert worden, die in beliebigen Kombinationen einem Objekt zugeordnet werden können. Es ist theoretisch möglich, über diese Diplomarbeit hinaus, noch eine Reihe weiterer Alternativen hinzuzufügen.

3.1.1 Der Wert der Potentialfunktion

Während in der Mathematik eine Potentialfunktion $\varphi(x; y)$ in einem ebenen Potentialfeld stets abhängig von einer Position ist und der Gradient über partielle Ableitungen zu bestimmen ist [Papula, 1999], wird in dieser Arbeit von einem verein-

fachten Modell ausgegangen, in dem eine Potentialfunktion $f(x)$ verwendet wird, die lediglich abhängig von der Entfernung zu dem Objekt ist, welchem das Potentialfeld zugeordnet ist. Dies entspricht dem Vorgehen von [Latombe, 1991].

Im Folgenden sei $\vec{PO}$ ein Vektor von einer beliebigen Position P zu dem Objekt, sowie f die dem Objekt zugeordnete Funktion. Die genaue Bestimmung von $\vec{PO}$ ist von der jeweiligen Form des Feldes abhängig; die Formen werden in Abschnitt 3.3 beschrieben, die möglichen Funktionen f in Abschnitt 3.2.

Der Wert der Potentialfunktion φ an der Position P kann somit folgendermaßen bestimmt werden:

$$\varphi(P) = f(|\vec{PO}|) \tag{3.1}$$

Der klassische Ansatz der Wegplanung benötigt die Potentialfunktion nicht direkt, sondern lediglich deren Ableitung. Innerhalb des hier vorgestellten Ansatzes wird sie jedoch zum einen zur der Vermeidung lokaler Minima (Abschnitt 4.3) sowie zum anderen zur der Bewertung von Aktionen verwendet (Abschnitt 5.1.5).

3.1.2 Das Potentialfeld

Aufgrund der zuvor erläuterten Modellierung kann die Wirkung des Potentialfeldes an einer Position P, also der Feldvektor $\vec{v}(P)$, nicht mehr direkt als Gradient von $\varphi(P)$ bestimmt werden. Stattdessen wird das im Folgenden beschriebene Berechnungsverfahren aus [Latombe, 1991] verwendet.

Als Grundlage der Berechnung dient der zu normierende Vektor $\vec{PO}$, welcher die Richtung von $\vec{v}$ vorgibt. Dieser Vektor wird skaliert mit dem Wert der ersten Ableitung der Potentialfunktion f in Abhängigkeit von der Länge von $\vec{PO}$. Der Funktionswert bestimmt die Orientierung und die Länge von $\vec{v}$.

Für den Vektor $\vec{v}$, der auf die Position P wirkt, ergibt sich daher folgende Gleichung:

$$\vec{v}(P) = f'(|\vec{PO}|)\frac{\vec{PO}}{|\vec{PO}|} \tag{3.2}$$

3.2 Funktionen

In der hier verwendeten Modellierung werden Funktionen im Allgemeinen durch zwei Parameter bestimmt: die Reichweite r der Funktion sowie den Funktionswert z an der Stelle Null. Aus diesen beiden Werten lassen sich die konkreten Funktionsparameter errechnen. Alle Funktionen sind definiert und differenzierbar auf der

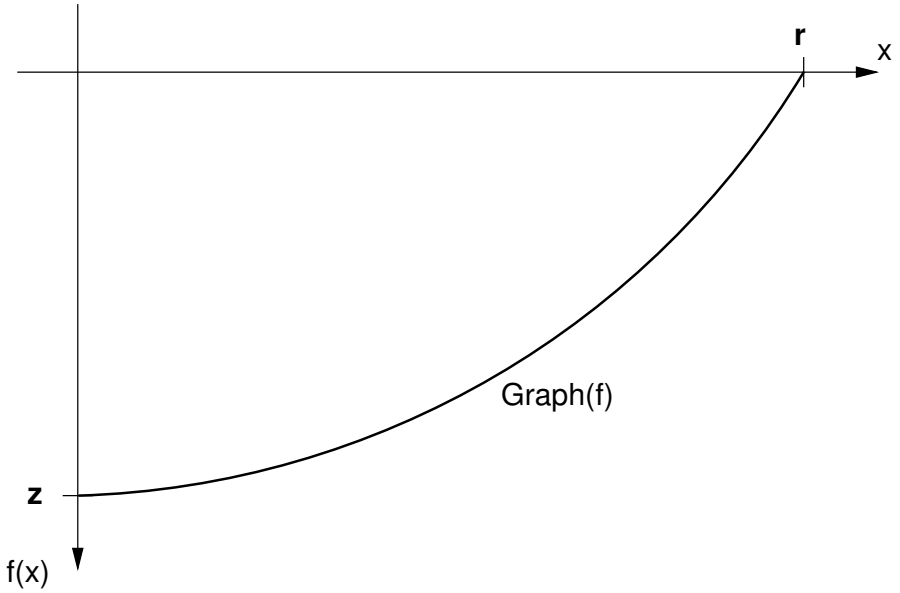


Abbildung 3.1: *Der Graph einer allgemeinen Funktion f. Die Stelle r ist immer die einzige Nullstelle. An der Stelle Null hat die Funktion stets den Wert z.*

Menge der positiven reellen Zahlen einschließlich der 0. Da der Wirkungsradius eines Objektes durch r begrenzt wird, ist folgendes festgelegt worden:

$$f(x) = f'(x) = 0, \forall x \geq r \tag{3.3}$$

Hieraus folgt zudem, das gemäß der Gleichung 3.2 an sämtlichen Positionen außerhalb des Wirkungsradius lediglich Nullvektoren erzeugt werden, also kein Potentialfeld mehr existiert.

Der Funktionswert z stellt den Extremwert der Funktion dar und begrenzt somit auch den Wertebereich der Funktion. Es gilt:

$$f(0) = z \tag{3.4}$$

$$0 \leq |f(x)| \leq |z| \tag{3.5}$$

Eine Beispielfunktion ist in Abbildung 3.1 zu sehen. Allein das Vorzeichen von z bestimmt, ob ein Objekt repulsiv oder attraktiv ist. Bei negativem z sind alle Funktionswerte $f(x)$ negativ und das Objekt ist attraktiv. Entsprechend ist das Objekt repulsiv bei positivem Vorzeichen.

3.2.1 Die quadratische Funktion

Zur Modellierung einer Zielposition wird oftmals eine ganzrationale Funktion zweiten Grades, auch Parabel genannt, verwendet. Da der Wert der 1. Ableitung und

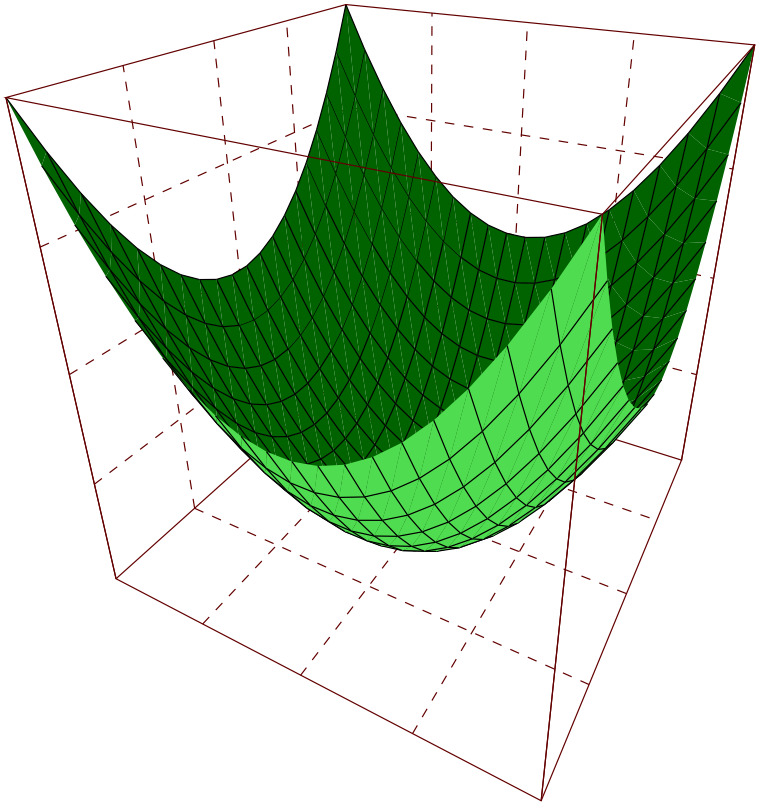


Abbildung 3.2: *Der Graph einer quadratischen Funktion $f_{quad}(d)$ mit $d = \sqrt{x^2 + y^2}$.*

somit auch die Länge von $\vec{v}$ in Richtung der Zielposition, also für einen gegen Null strebenden Wert von x, ebenfalls gegen Null konvergiert, verleiht die Funktion dem Bewegungsverhalten Stabilität [Latombe, 1991, Khatib, 1986]. Bei größer werdender Entfernung vom Objekt nimmt der Betrag des Gradienten jedoch auch stetig zu, siehe Abbildung 3.2.

Eine quadratische Funktion f_{quad} hat die allgemeine Formel:

$$f_{quad}(x) = ax^2 + bx + c \tag{3.6}$$

Für die Ableitung f'_{quad} gilt:

$$f'_{quad}(x) = 2ax + b \tag{3.7}$$

Die verwendete Funktion soll folgende Bedingungen erfüllen:

$$f_{quad}(0) = z \tag{3.8}$$
$$f_{quad}(r) = 0 \tag{3.9}$$
$$f'_{quad}(0) = 0 \tag{3.10}$$

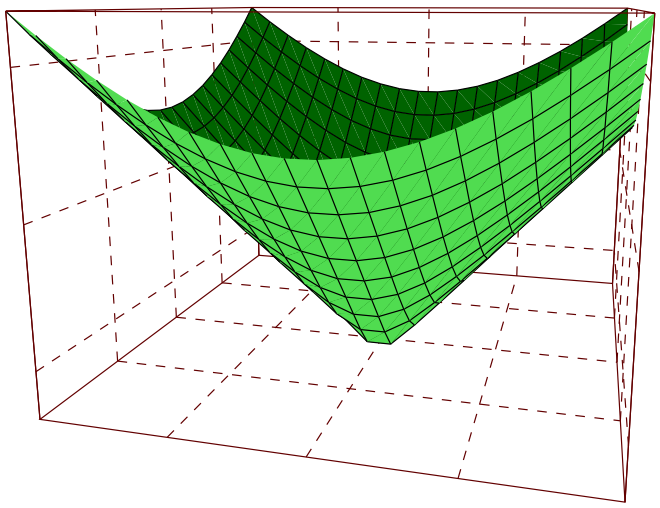


Abbildung 3.3: *Der Graph einer linearen Funktion* $f_{lin}(d)$ *mit* $d = \sqrt{x^2 + y^2}$.

Daraus folgt:

$$c = z \tag{3.11}$$
$$ar^2 + br + c = 0 \tag{3.12}$$
$$b = 0 \tag{3.13}$$

Somit ergeben sich folgende Gleichungen für die quadratische Funktion f_{quad} und ihre Ableitung f'_{quad}:

$$f_{quad}(x) = \frac{-z}{r^2}x^2 + z \tag{3.14}$$
$$f'_{quad}(x) = \frac{-2z}{r^2}x \tag{3.15}$$

3.2.2 Die lineare Funktion

Lineare Funktionen werden ebenfalls zur Modellierung von Zielpositionen verwendet. Im Unterschied zur Parabel bleibt der Wert der ersten Ableitung, und somit die Länge des Gradienten, jedoch stets konstant [Latombe, 1991]. Dies ermöglicht ein gleichförmiges Bewegungsverhalten in Richtung der Zielposition. Ein Graph einer solchen Funktion ist in Abbildung 3.3 zu sehen.

Eine lineare Funktion f_{lin} hat die allgemeine Formel:

$$f_{lin}(x) = ax + b \tag{3.16}$$

Die verwendete Funktion muss folgende Bedingungen erfüllen:

$$f_{lin}(0) = z \tag{3.17}$$
$$f_{lin}(r) = 0 \tag{3.18}$$

Daraus folgt:

$$b = z \tag{3.19}$$
$$ar + b = 0 \tag{3.20}$$

Somit ergeben sich folgende Gleichungen für die lineare Funktion f_{lin} und ihre Ableitung f'_{lin}:

$$f_{lin}(x) = \frac{-z}{r}x + z \tag{3.21}$$
$$f'_{lin}(x) = \frac{-z}{r} \tag{3.22}$$

3.2.3 Eine gebrochenrationale Funktion

Hindernisse werden als repulsive Objekte in Potentialfeldern modelliert; dazu werden im Allgemeinen gebrochenrationale Funktionen verwendet [Latombe, 1991], die folgende Eigenschaften haben:

$$\lim_{x\to\infty} f(x) = 0 \tag{3.23}$$
$$\lim_{x\to 0} f(x) = \pm\infty \tag{3.24}$$

Dadurch nimmt auch die Steigung in der Nähe eines Objektes stetig zu, das Objekt wirkt stark abstoßend. Bei größer werdender Entfernung konvergieren Funktionswerte und Steigung gegen Null, der Einfluss des Potentialfeldes eines repulsiven Objektes auf entfernte Positionen ist verschwindend gering. Abbildung 3.4 zeigt eine solche Funktion.

Die in dieser Arbeit verwendete gebrochenrationale Funktion f_{frac} hat die allgemeine Formel:

$$f_{frac}(x) = \frac{a}{x} + b, \forall x \neq 0 \tag{3.25}$$

Da f_{frac} an der Stelle 0 nicht definiert ist, kann die zuvor aufgestellte Bedingung $f(0) = z$ nicht mehr zur Bestimmung der Parameter für die Berechnung verwendet werden. Es wird daher ein neuer, bei der Modellierung anzugebender, Parameter ϵ eingeführt, so dass gilt:

$$f_{frac}(x) = z, \forall x \leq \epsilon \tag{3.26}$$

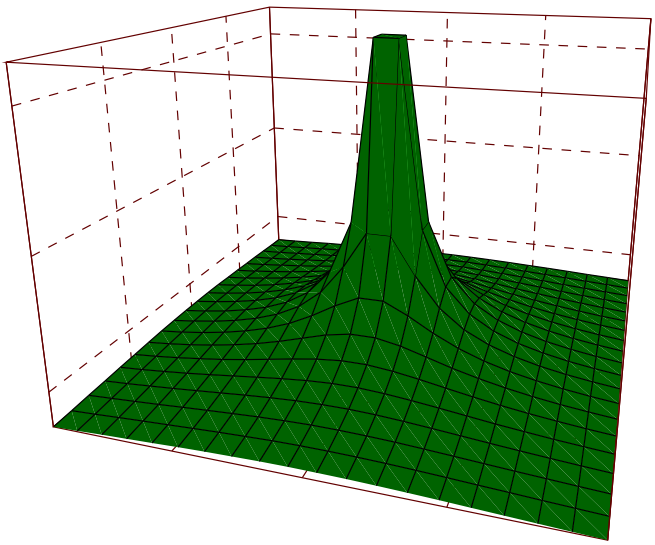


Abbildung 3.4: *Der Graph einer gebrochenrationalen Funktion* $f_{frac}(d)$ *mit* $d = \sqrt{x^2 + y^2}$.

Daraus ergeben sich nun folgende Bedingungen für die Funktion:

$$f_{frac}(\epsilon) = z \tag{3.27}$$
$$f_{frac}(r) = 0 \tag{3.28}$$

Es folgt:

$$\frac{a}{\epsilon} + b = z \tag{3.29}$$
$$\frac{a}{r} + b = 0 \tag{3.30}$$

Somit ergeben sich folgende Gleichungen für die gebrochenrationale Funktion f_{frac} und ihre Ableitung f'_{frac}:

$$f_{frac}(x) = \frac{z}{(\frac{1}{\epsilon} - \frac{1}{r})}\frac{1}{x} - \frac{z}{\frac{r}{\epsilon} - 1} \tag{3.31}$$
$$f'_{frac}(x) = \frac{-z}{(\frac{1}{\epsilon} - \frac{1}{r})}\frac{1}{x^2} \tag{3.32}$$

3.2.4 Die soziale Funktion

Eine weitere Funktion ist die von [Reif and Wang, 1995] entwickelte *soziale Funktion*, die in Szenarien mit einer Vielzahl von autonomen Robotern zur Beschreibung

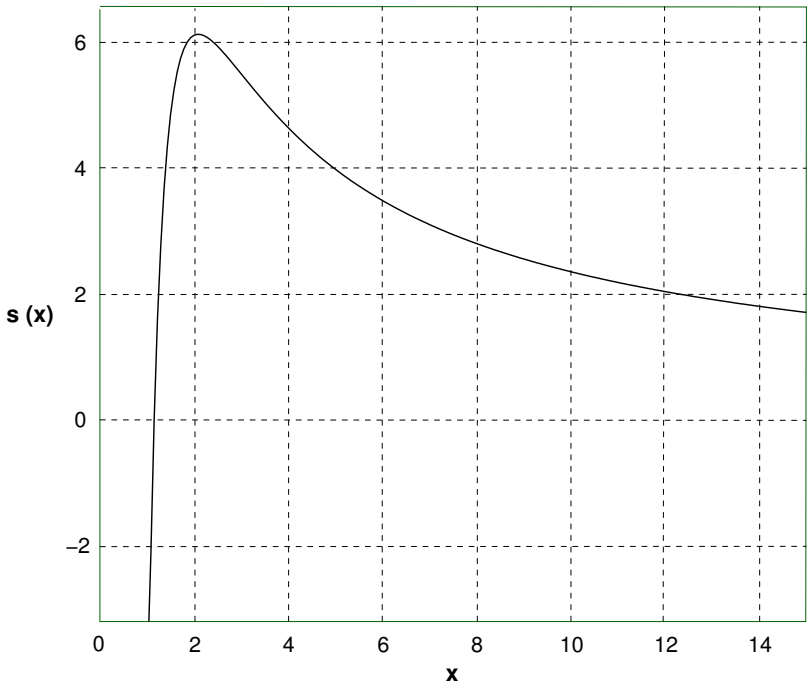


Abbildung 3.5: *Der Graph einer sozialen Beispielfunktion* $s\,(x) = -20.0/x^{3.0} + 15.0/x^{0.8}$.

von Positionierungsbeziehungen verwendet werden kann. Dabei wird die Beziehung eines Roboter zu einem anderen jeweils durch eine einzige Funktion beschrieben, die sein Abstandsverhalten charakterisiert. Diese Zuordnung ist unidirektional, so dass zwei verschiedene Roboter sich auch unterschiedlich einander gegenüber verhalten können. Im Rahmen dieser Arbeit ist es möglich, diese Funktion beliebigen Objekten zuzuordnen.

Eine soziale Funktion s ist folgendermaßen definiert:

$$s(x) = -\frac{c_1}{x^{\sigma_1}} + \frac{c_2}{x^{\sigma_2}} \quad , \forall x \neq 0 \tag{3.33}$$

$$c_1, c_2 \geq 0, \quad \sigma_1 > \sigma_2 > 0$$

Sie besteht aus der repulsiven Komponente $-\frac{c_1}{x^{\sigma_1}}$, die für $\lim_{x \to 0}$ dominiert, so dass die Funktion s in der Nähe des Objektes abstoßend wirkt, und der attraktiven Komponente $\frac{c_2}{x^{\sigma_2}}$, die bei größeren Entfernungen dominant ist und dadurch, jedoch immer schwächer werdend, anziehend wirkt, wie in Abbildung 3.5 zu sehen ist.

Die Funktion hat eine gesonderte Stellung, da sie nicht über einen Extremwert z und eine Reichweite r beschrieben werden kann, was einen unbegrenzten Wirkungsradius zur Folge hat. Stattdessen sind die zu setzenden Parameter die repulsive Konstante c_1, der repulsive Exponent σ_1, die attraktive Konstante c_2 und der attraktive

Exponent σ_2. Deren Wahl bestimmt das Verhalten der Funktion bei bestimmten Entfernungen. Eine markante Stelle ist die Entfernung d, an der sich die repulsive und die attraktive Komponente gegenseitig aufheben, also ein Gleichgewichtszustand erreicht ist. Es gilt:

$$d = (c_1/c_2)^{\frac{1}{\sigma_1 - \sigma_2}} \tag{3.34}$$

Ausgehend von einer bestimmten Entfernung, die der Roboter zu einem anderen Objekt haben soll, können mittels dieser Gleichung die Parameter gewählt werden.

Da die soziale Funktion zur Berechnung der Länge eines Feldvektors verwendet wird, entspricht sie, im Kontext des zuvor beschriebenen Modells, der ersten Ableitung einer noch zu bestimmenden Funktion f_{soc}. Da s an der Stelle Null nicht definiert ist, muss auch hier, wie schon bei den gebrochenrationalen Funktionen, eine Konstante ϵ verwendet werden, so dass gilt:

$$f'_{soc}(x) = \begin{cases} s(x) & ,\forall x > \epsilon \\ s(\epsilon) & ,\forall x \leq \epsilon \end{cases} \tag{3.35}$$

Um die Funktion auch als Potentialfunktion nutzen zu können, muss f_{soc} durch Bildung einer Stammfunktion bestimmt werden. Dabei werden der repulsive und der attraktive Teil der Funktion getrennt betrachtet, ferner kann eine Konstante k hinzugefügt werden, so dass gilt:

$$f_{soc}(x) = \begin{cases} f_{rep}(x) + f_{att}(x) + k & ,\forall x > \epsilon \\ f_{rep}(\epsilon) + f_{att}(\epsilon) + k & ,\forall x \leq \epsilon \end{cases} \tag{3.36}$$

Für die beiden Funktionen f_{rep} und f_{att} müssen aufgrund der Exponenten σ_1 und σ_2 jeweils zwei Fälle unterschieden werden:

$$f_{rep}(x) = \begin{cases} c_1 \cdot \ln(x) & ,falls\ \sigma_1 = 1 \\ (c_1/(1-\sigma_1))x^{1-\sigma_1} & ,\sigma_1 \neq 1 \end{cases} \tag{3.37}$$

$$f_{att}(x) = \begin{cases} c_2 \cdot \ln(x) & ,falls\ \sigma_2 = 1 \\ (c_2/(1-\sigma_2))x^{1-\sigma_2} & ,\sigma_2 \neq 1 \end{cases} \tag{3.38}$$

Die Parametrisierung während der Modellierung ist bei der sozialen Funktion zwar aufwändiger als bei anderen Funktionen, sie bietet jedoch interessante Möglichkeiten zur Erzeugung eines gemeinsamen Verhaltens größerer Gruppen von Robotern. Einige ausführliche Beispiele, wie die gleichmäßige Verteilung über eine Fläche oder das Versammeln an einem Punkt, werden in [Reif and Wang, 1995] vorgestellt.

3.3 Feldformen

Das Potentialfeld und die Potentialfunktion wirken auf die Umgebung eines Objekts. Die davon betroffenen Regionen sowie die Richtung der Vektoren ergeben

sich, neben der Reichweite der Funktion, aus der Form des Feldes. Die verschiedenen Formen unterscheiden sich im jeweiligen Verfahren zur Bestimmung des eingangs des Kapitels erläuterten Vektors $\vec{PO}$ von einer beliebigen Position P zum Objekt.

3.3.1 Zentrierte Felder

Die einfachste Form eines Feldes ist das zentrierte Feld. Es wirkt, ausgehend von der Position O des Objekts, gleichmäßig in alle Richtungen und wird zumeist zur Beschreibung von Zielpositionen oder Hindernissen unbekannter Form verwendet. Eine dem Objekt eventuell zugewiesene Form wird dabei vollständig ignoriert. Der Vektor $\vec{PO}$ kann sehr effizient über die Ortsvektoren von O und P bestimmt werden:

$$\vec{PO} = \vec{O} - \vec{P} \tag{3.39}$$

Abbildung 3.6a zeigt ein zentriertes repulsives Feld.

3.3.2 Felder um geometrische Objekte

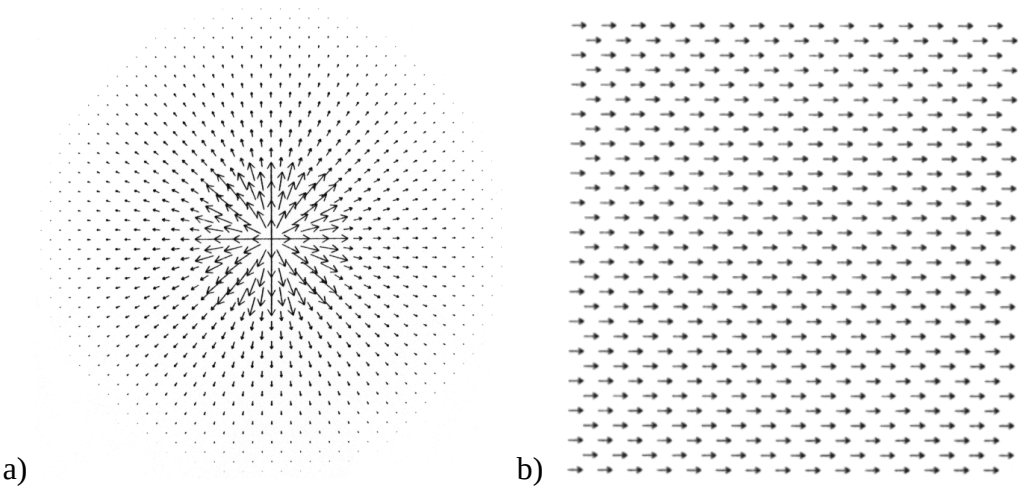
Ist die Form eines Objekts bekannt, so kann sie zur Berechnung eines Feldes verwendet werden, welches eine geometrische Figur umschließt. Dabei wird zur Bestimmung von $\vec{PO}$ nicht mehr auf die Position O zurückgegriffen, sondern eine Position Q bestimmt, welche den zu P nächsten Punkt auf der Figur repräsentiert. Somit gilt:

$$\vec{PO} = \vec{Q} - \vec{P} \tag{3.40}$$

Das Verfahren zur Bestimmung von Q ist abhängig von der jeweils verwendeten Figur, siehe hierzu Abschnitt 3.4.

Handelt es sich bei der Figur um eine Strecke oder ein konvexes Polygon, so bilden sich Regionen, in denen Vektoren parallel zueinander und orthogonal zur nächstgelegenen Kante der Figur verlaufen. Dies entspricht der in [Arkin, 1998] speziell aufgeführten Form eines *uniformen Feldes*, die in dieser Arbeit nicht direkt umgesetzt wurde, aber durch Figuren – insbesondere durch Strecken – beschrieben werden kann. Abbildung 3.6b zeigt ein solches uniformes Feld.

Da das Potentialfeld und die Potentialfunktion lediglich für die Umgebung des Objekts berechnet werden, verbleibt das Innere der Figur zunächst undefiniert. Da aber nicht nur feste Hindernisse, die der Roboter theoretisch nicht durchdringen kann, sondern auch größere Regionen, die nicht betreten werden oder aber, nach einer vorhergehenden Falschlokalisation, schnellstmöglich wieder verlassen werden sollten, modelliert werden können, müssen auch im Inneren der Region entsprechende



a) b)

Abbildung 3.6: *a) Ein zentriertes repulsives Feld sowie b) ein uniformes Feld. (Die Bilder wurden entnommen aus [Arkin, 1998])*

Vektoren erzeugt werden. Ebenso sollte, im Rahmen späterer Bewertungen einzelner Positionen, stets ein sinnvoller Wert für die Potentialfunktion berechnet werden können. Aus diesen Gründen wurden mehrere Konventionen festgelegt:

Bei attraktiven Objekten werden im Inneren der Figur lediglich Nullvektoren erzeugt, da nach dem Erreichen einer Zielregion keine weiteren Bewegungen mehr nötig sind. Im Gegensatz zu [Latombe, 1991] werden in repulsiven Regionen weiterhin Vektoren erzeugt, die ein Verlassen ermöglichen. Dies kann, ebenso wie die Erzeugung des Feldes außerhalb der Figur, über die Bestimmung des nächstgelegenen Punktes Q auf der Figur realisiert werden. Die Potentialfunktion bleibt, sowohl bei attraktiven als auch bei repulsiven Objekten, im Inneren einer Figur stets konstant und liefert den Wert von $f(0)$.

3.3.3 Kreisausschnitte

Felder in Form von Kreisausschnitten wurden eingeführt, um bestimmte Regionen entlang der Ausrichtung von Objekten zu modellieren. Im Rahmen der Aktionsauswahl könnten derartige Felder beispielsweise eingesetzt werden, um in einem Fußballspiel Regionen vor Mitspielern als positiv, sowie Regionen vor Gegenspielern als negativ zu bewerten. Ebenso können Potentialfelder erzeugt werden, die es ermöglichen, sich einem attraktiven Objekt unter Berücksichtigung seiner Bewegungsrichtung zu nähern, zum Beispiel um einen herannahenden Ball abzufangen. Ein ähnliches Verfahren wird auch in [Arkin, 1998] zur vorausschauenden Vermeidung beweglicher Hindernisse vorgestellt. Abbildung 3.7a zeigt ein solches repul-

sives Feld.

Der Aufbau des Feldes im Kreisausschnitt erfolgt in zwei Schritten und stellt eine Erweiterung des in 3.1 beschriebenen Schemas dar. Sowohl die Potentialfunktion als auch das Potentialfeld stehen zum einen, wie auch die anderen Felder, im Bezug zur Entfernung zum Objekt, zum anderen aber auch im Bezug zur Entfernung zur Mitte des Kreisausschnitts.

Im ersten Schritt wird zunächst ein zentriertes Feld ausgehend von der Objektposition O erzeugt, das sich allerdings auf die Positionen innerhalb des gegebenen Kreisausschnitts beschränkt. Sei $\vec{r}$ der Richtungsvektor des Objekts und α der Winkel des Kreisausschnitts, dann gilt:

$$\varphi_l(P) = f(|\vec{PO}|) \quad ,\forall\ P \mid \measuredangle(\vec{r}, \vec{PO}) <= \frac{\alpha}{2} \tag{3.41}$$

$$\vec{v_l}(P) = f'(|\vec{PO}|)\frac{\vec{PO}}{|\vec{PO}|} \quad ,\forall\ P \mid \measuredangle(\vec{r}, \vec{PO}) <= \frac{\alpha}{2} \tag{3.42}$$

In einem zweiten Schritt wird, ausgehend von den vorherigen Ergebnissen, ein weiteres Feld erzeugt, welches von der Mitte des Kreisausschnitts bis an dessen Rand verläuft. Dazu wird zum einen ein Vektor $\vec{PC}$ bestimmt, der, ausgehend von einer Position P, orthogonal zur Winkelhalbierenden des Kreisausschnitts verläuft, und zum anderen eine zweite Funktion f_c verwendet, die den Werteverlauf dieses Feldes charakterisiert. Der Typ der Funktion ist bei der Modellierung anzugeben, die beiden benötigten Parameter r und z werden automatisch bestimmt:

$$z = \varphi_l(|\vec{PO}|) \tag{3.43}$$

$$r = \frac{\alpha}{2}|\vec{PO}| \tag{3.44}$$

Mittels dieser zweiten Funktion kann nun durch folgende Gleichungen ein weiteres Feld abhängig vom Winkel zur Mitte des Kreisausschnitts $\measuredangle(\vec{r}, \vec{PO})$ erzeugt werden:

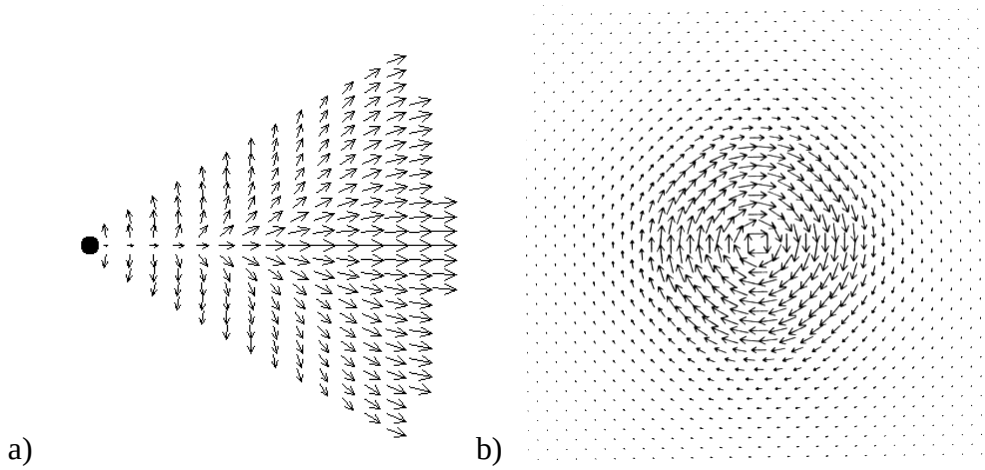
$$\varphi_c(P) = f_c(\measuredangle(\vec{r}, \vec{PO})) \tag{3.45}$$

$$\vec{v_c}(P) = f_c'(\measuredangle(\vec{r}, \vec{PO}))\frac{\vec{PC}}{|\vec{PC}|} \tag{3.46}$$

Durch die Kombination der Felder ergibt sich für das Gesamtfeld:

$$\varphi(P) = \varphi_c(P) \tag{3.47}$$

$$\vec{v}(P) = \vec{v_l} + \vec{v_c} \tag{3.48}$$



a) b)

Abbildung 3.7: *a) Ein repulsives Feld in einem Kreisausschnitt und b) ein tangential verlaufendes Feld (Bild entnommen aus [Arkin, 1998]).*

Dieses Berechnungsverfahren ermöglicht einen kontinuierlichen Verlauf der Werte der Potentialfunktion innerhalb des Kreisausschnitts. Der einzige Extremwert ist stets $\varphi(O)$, die Funktion nähert sich in Richtung der drei Ränder streng monoton dem Wert Null. Das entsprechende Vektorfeld verläuft, wie in Abbildung 3.7a zu sehen ist, abhängig von O und der Mitte des Kreisausschnitts. Die Vektorlängen sind dabei abhängig von den gewählten Funktionen f und f_c.

3.3.4 Tangentiale Felder

Zur Umrundung von Objekten, beispielsweise um ein Objekt von allen Seiten zu untersuchen oder um beim Fußball einen Ball zu umrunden, werden tangentiale Vektorfelder benötigt. Diese können, kombiniert mit den bisher vorgestellten Feldformen, erzeugt werden, indem die errechneten Vektoren eines Feldes entweder um $\frac{\pi}{2}$ oder um $-\frac{\pi}{2}$ rotiert werden. Auf den Wert der Potentialfunktion hat diese Rotation keinerlei Auswirkung, die Vektoren behalten ebenfalls ihre ursprüngliche Länge. Abbildung 3.7b zeigt ein solches Feld.

3.4 Geometrische Figuren

Jedem Objekt kann eine geometrische Figur zugeordnet werden. Diese kann als Basis zur Errechnung des umliegenden Feldes dienen. Ebenso wird sie zur Kollisionserkennung bei der Durchführung von Aktionen verwendet. Es wurden drei geometrische Primitive implementiert: Strecke, konvexes Polygon und Kreis.

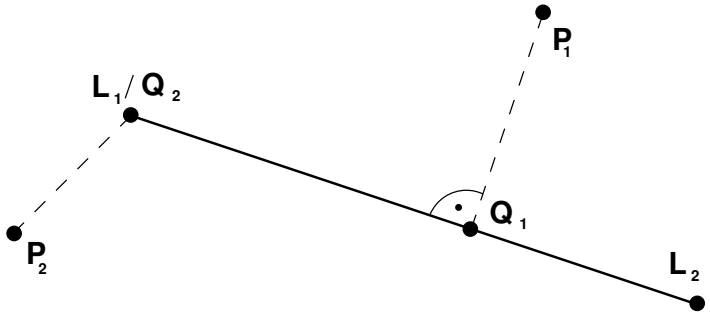


Abbildung 3.8: *Die zu zwei Punkten P_1 und P_2 jeweils nächsten, auf einer Strecke $|L_1L_2|$ liegenden Punkte Q_1 und Q_2.*

Dieser Abschnitt stellt die Primitive vor und beschreibt die jeweiligen, für den Aufbau eines Feldes wichtigen, Funktionen zur Berechnung des zu einem beliebigen Punkt P nächstgelegenen Punktes Q auf der Figur.

3.4.1 Strecken

Strecken können zur Modellierung von Wänden oder anderen Begrenzungen der Umgebung verwendet werden. Ebenso dienen sie der Erzeugung uniformer Felder. Eine Strecke wird durch ihre zwei Endpunkte L_1 und L_2 beschrieben..

Der auf einer Strecke zur einer beliebigen Position P nächstgelegene Punkt Q ist entweder dessen senkrechte Projektion, falls diese existiert, oder der nächstgelegene der beiden Endpunkte, siehe hierzu Abbildung 3.8.

3.4.2 Konvexe Polygone

Ist die Form eines Objekts bekannt und soll auch entsprechend von der Verhaltenssteuerung genutzt werden, kann sie über ein konvexes Polygon beschrieben oder angenähert werden. Durch die Beschränkung auf diese spezielle Form von Polygonen sind effizientere interne Berechnungen möglich, wie beispielsweise für den Test des Enthaltenseins eines Punktes im Polygon. Komplexere konkave Objekte müssen daher auf mehrere Objekte aufgeteilt und somit durch mehrere konvexe Polygone beschrieben werden.

Bei der Modellierung wird ein Polygon durch eine Folge von Punkten $\{K_1, \ldots, K_n\}$ mit $n \geq 3$ beschrieben.

Der zu einer Position P nächstgelegene Punkt Q auf einem konvexen Polygon wird aus einer Menge von Punkten, bestehend aus $\{K_1, \ldots, K_n\}$ sowie sämtlichen mög-

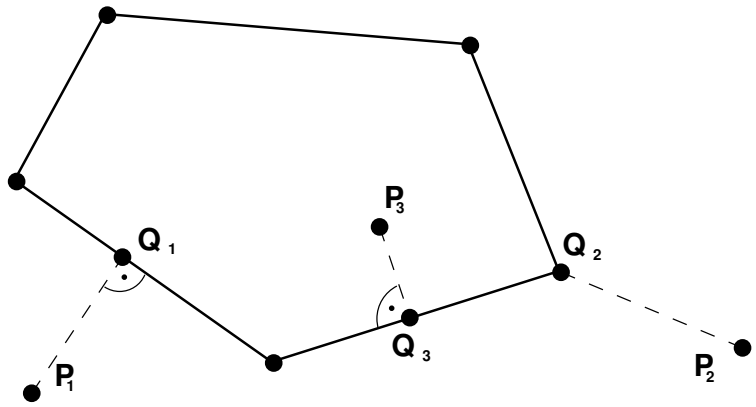


Abbildung 3.9: *Drei Punkte P_n und ihre jeweils nächstgelegenen Punkte Q_n auf dem Polygon.*

lichen senkrechten Projektionen von P auf Kanten des Polygons, bestimmt. Dies ermöglicht auch die Bestimmung von Q ausgehend von Positionen innerhalb des Polygons, wie in Abbildung 3.9 zu sehen ist.

3.4.3 Kreise

Kreise lassen sich, neben der Modellierung runder Objekte, auch im Zusammenhang mit Objekten unbekannter Form verwenden. Zum einen können durch sie Hindernisse beschrieben werden, deren genaue Form im Vorfeld nicht bekannt ist, die jedoch, um etwaige Kollisionen bei der Planung von Aktionen zu vermeiden, trotzdem mit einer Form versehen werden sollen. Zum anderen kann die Form zwar bekannt sein, die genaue Ausrichtung von der Sensorik aber nicht erkannt werden, so dass das Objekt über die Grundfläche seines Rotationskörpers beschrieben wird. Durch die jeweilige Annäherung der Form durch einen Kreis kann ein Feld erzeugt werden, das gleichmäßig verläuft und effizient zu berechnen ist.

Der Ortsvektor einer Projektion Q eines beliebigen Punktes P auf einen Kreis an der Position O mit dem Radius r kann durch folgende Formel bestimmt werden:

$$\vec{Q} = \frac{r}{|\vec{PO}|} \vec{PO} \tag{3.49}$$

Dies ist auch in Abbildung 3.10 zu sehen.

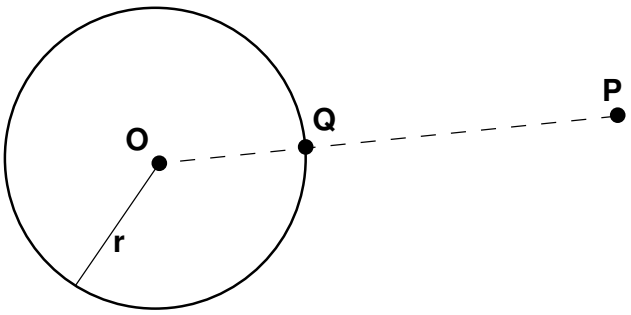


Abbildung 3.10: *Der dem Punkt P nächste, auf einem Kreis an der Position O liegende Punkt ist Q.*

3.4.4 Objekte ohne Form

In einigen Fällen ist eine Zuordnung einer geometrischen Form nicht sinnvoll, beispielsweise bei einem anzufahrenden Zielpunkt. Es wird dann von einem punktförmigen Objekt ausgegangen, die Enfernung des Roboters zum Objekt ist somit immer gleich der Entfernung zur Position des Objekts. Ein Objekt ohne physische Ausdehnung bleibt bei der Kollisionserkennung unberücksichtigt.

3.5 Objekte in einem probabilistischen Weltmodell

Das in dieser Arbeit angewandte Verfahren zur Modellierung von Objekten ermöglicht die direkte Verarbeitung einer Menge von verschiedenen Hypothesen über einen Objektzustand. Die entsprechenden Daten können dabei aus probabilistischen Lokalisierungsverfahren übernommen werden.

3.5.1 Probabilistische Lokalisierungsverfahren

Robotersysteme, die über ein Weltmodell verfügen, welches sowohl die eigene als auch die Positionen anderer Objekte in absoluten Koordinaten beinhaltet, erzeugen dieses mittels verschiedener Lokalisierungsverfahren aus den Sensordaten. Ein weit verbreitetes Verfahren zur Selbstlokalisierung ist die *Monte-Carlo-Lokalisierung* von [Fox et al., 1999]. Dabei handelt es sich um ein probabilistisches Verfahren, welches eine Vielzahl von Hypothesen über die Eigenposition, eine sogenannte Partikelmenge, verfolgt und diesen anhand von Sensordaten Wahrscheinlichkeiten zuordnet. Ein solches Verfahren ist auch von [Schulz et al., 2001] zur Verfolgung der Bewegungen von Objekten in der Umgebung des Roboters verwendet worden. Ein davon abweichender ebenfalls probabilistischer, Ansatz ist das *Multiple-*

Hypothesis-Tracking, dass von [Schmitt et al., 2002] in der RoboCup-Domäne zur Lokalisierung fremder Roboter verwendet wird.

Obwohl diese Verfahren intern stets eine Menge möglicher Objektzustände betrachten, wird zur anschließenden Verarbeitung der Daten in einer Verhaltenssteuerung zumeist nur die jeweils wahrscheinlichste Hypothese ausgewählt, sämtliche anderen Hypothesen verbleiben unbeachtet.

3.5.2 Integration in Potentialfelder

Da bei den, in dieser Arbeit vorgestellten, Verfahren zur Bewegungssteuerung und Aktionsbewertung keinerlei konkrete Repräsentation von Objekten im Sinne von [Brooks, 1991] nötig ist, sondern lediglich Vektoren und Werte von Potentialfunktionen, unabhängig von deren Zuordnung zu Objekten, verarbeitet werden, kann auch ein probabilistisches Weltmodell auf der Ebene der Objektmodellierung verwendet werden. Dabei wird eine Menge von Annahmen aus der Lokalisierung übernommen, zu jeder einzelnen Hypothese können der Wert der Potentialfunktion und das Potentialfeld entsprechend der Modellierung des Objekts bestimmt und abhängig von der Wahrscheinlichkeit in ein Gesamtergebnis eingebracht werden.

Die Möglichkeit der Verarbeitung von Objekten mit einer Wahrscheinlichkeitsverteilung führt auch zu einer Anwendung im Bereich der Manipulation der Umgebung, wie in Abschnitt 5.2.4 beschrieben wird.

Im Folgenden werden die nötigen Berechnungsverfahren für die Verarbeitung eines probabilistischen Weltmodells vorgestellt.

3.5.3 Wahrscheinlichkeitsverteilte Eigenposition

Existiert zur Position P des Roboters eine Menge von Hypothesen $\{P_0, \ldots, P_n\}$, denen jeweils eine Wahrscheinlichkeit $\{w_0, \ldots, w_n\}$ zugeordnet ist, während die Position des Objekts eindeutig ist, können $\varphi(P)$ und $\vec{v}(P)$ mittels der in Abschnitt 3.1 definierten Funktionen bestimmt werden:

$$\varphi(P) = \sum_{i=0}^{n} w_i \, \varphi(P_i) \tag{3.50}$$

$$\vec{v}(P) = \sum_{i=0}^{n} w_i \, \vec{v}(P_i) \tag{3.51}$$

Dabei gilt für die Wahrscheinlichkeiten:

$$0 \leq w \leq 1 \quad , \forall w \in \{w_0, \ldots, w_n\} \tag{3.52}$$

Die Bedingung $\sum_{i=0}^{n} w_i = 1$ ist nicht zwingend notwendig und, abhängig vom verwendeten Lokalisierungsverfahren, auch nicht immer sinnvoll. Bei der Modellierung der Funktionsparameter des Objekts sollte daher auf die zu erwartenden Wahrscheinlichkeitswerte Rücksicht genommen werden.

3.5.4 Wahrscheinlichkeitsverteilte Objektpositionen

Im umgekehrten Fall, also einer eindeutigen Position P des Roboters und einer Menge von Hypothesen $\{O_0, \ldots, O_n\}$ mit Wahrscheinlichkeitswerten $\{v_0, \ldots, v_m\}$ für die Position des Objekts, können $\varphi(P)$ und $\vec{v}(P)$ folgendermaßen bestimmt werden:

$$\varphi(P) = \sum_{j=0}^{m} v_j \, f(|P\vec{O}_j|) \tag{3.53}$$

$$\vec{v}(P) = \sum_{j=0}^{m} v_j \, f'(|P\vec{O}_j|) \frac{P\vec{O}_j}{|P\vec{O}_j|} \tag{3.54}$$

Für die Wahrscheinlichkeitswerte gelten dabei die gleichen Bedingungen wie zuvor für $\{w_0, \ldots, w_n\}$.

3.5.5 Kombination von Wahrscheinlichkeitsverteilungen

Es verbleibt der Fall, dass sowohl zur Position des Roboters als auch zu der des Objekts jeweils eine Menge von Hypothesen vorliegt. Dies führt zu einer Kombination der zuvor beschriebenen Gleichungen:

$$\varphi(P) = \sum_{i=0}^{n} \sum_{j=0}^{m} w_i v_j \, f(|P_i\vec{O}_j|) \tag{3.55}$$

$$\vec{v}(P) = \sum_{i=0}^{n} \sum_{j=0}^{m} w_i v_j \, f'(|P_i\vec{O}_j|) \frac{P_i\vec{O}_j}{|P_i\vec{O}_j|} \tag{3.56}$$

Kapitel 4

Bewegung im Potentialfeld

Dieses Kapitel beschreibt die eingesetzten Verfahren zur Bewegungsplanung mit Potentialfeldern. Neben einer Vorstellung des grundlegenden Verfahrens werden Relativbewegungen, die Behandlung lokaler Minima sowie einige Erweiterungen vorgestellt.

4.1 Das Verfahren zur Bestimmung der Bewegung

Die Bewegung eines Roboters setzt sich zusammen aus seiner Bewegungsrichtung, einer Rotation und einer Geschwindigkeit, die abhängig von den Potentialfeldern der umgebenden Objekte berechnet werden. Des Weiteren muss für ein Bewegungsverhalten ein Aktivierungswert für die spätere Auswahl durch die Verhaltenssteuerung bestimmt werden.

4.1.1 Berechnung des Bewegungsvektors

Sei $\{O_1, .., O_n\}$ die Menge der einem Bewegungsverhalten zugeordneten und zum Zeitpunkt der Berechnung aktivierten Objekte, dann kann zu jedem O_i mit $1 <= i <= n$ für eine beliebige Position P mittels der in Abschnitt 3.1.2 beschriebenen Gleichung ein Vektor $\vec{v_i}(P)$ berechnet werden, der auf die Position P wirkt. Dieses Verfahren kann auf die Position des Roboters angewendet und die Vektoren anschließend summiert werden [Latombe, 1991]:

$$\vec{v} = \sum_{i=1}^{n} \vec{v_i}(P) \tag{4.1}$$

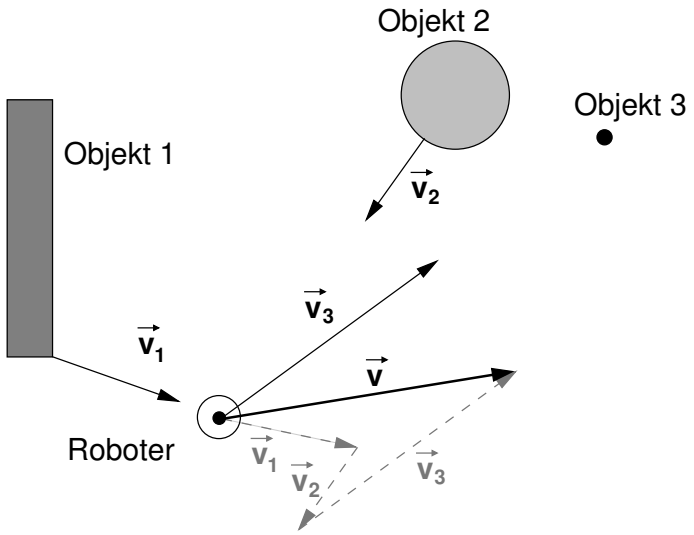


Abbildung 4.1: *Ein Roboter bestimmt seine Bewegungsrichtung durch Summierung der Feldvektoren der ihn umgebenden Objekte. In diesem Fall sind die Objekte 1 und 2 Hindernisse und erzeugen die abstoßenden Vektoren $\vec{v_1}$ und $\vec{v_2}$. Das Ziel ist Objekt 3, welches den anziehenden Vektor $\vec{v_3}$ erzeugt. Der daraus resultierende Richtungsvektor ist $\vec{v}$.*

Das Ergebnis ist ein Vektor $\vec{v}$, der eine absolute Bewegungsrichtung im Potentialfeld beschreibt. In Abbildung 4.1 wird dies an einem Beispiel verdeutlicht. Da die Verhaltenssteuerung letztlich eine relative Bewegungsrichtung erzeugen soll, muss $\vec{v}$ anschließend um den negierten Winkel der Ausrichtung des Roboters im verwendeten allozentrischen Weltmodell rotiert werden.

Die kontinuierliche Anwendung dieses Verfahrens führt den Roboter unter Vermeidung von Hindernissen durch seine Umgebung. Es handelt sich um einen Gradientenabstieg, da das Aufsummieren der Vektoren der Bildung des Gesamtgradienten der summierten Potentialfunktionen der Objekte entspricht.

Auf die allgemein übliche, explizite Benennung einer attraktiven Zielposition [Latombe, 1991, Khatib, 1986] wird hier verzichtet. Diese Zuordnung wird auf der Modellierungsebene durchgeführt, aus der Sicht der Bewegungssteuerung sind sämtliche Objekte lediglich Erzeuger von Vektoren.

4.1.2 Rotation und Geschwindigkeit

Die Rotation α des Roboters in der Ebene entspricht dem Winkel zwischen dem relativen Richtungsvektor $\vec{v}$ und der positiven x-Achse des relativen Koordinatensystems des Roboters und kann daher mittels der *atan2*-Funktion aus den Kompo-

nenten von $\vec{v}$ berechnet werden:

$$\alpha = \mathit{atan2}\,(\vec{v}_y, \vec{v}_x) \tag{4.2}$$

Die zusätzliche Berechnung der Rotation auf Basis der Bewegungsrichtung erscheint zunächst redundant, erfährt aber durch die mögliche Deaktivierung von Freiheitsgraden, wie in Abschnitt 4.4.2 beschrieben, eine besondere Bedeutung.

Die Geschwindigkeit des Roboters entspricht der Länge des erzeugten Richtungsvektors, ist also stets $|\vec{v}|$. Das bedeutet beispielsweise, dass bei einer über eine quadratische Funktion beschriebenen Zielposition bei größeren Entfernungen hohe Geschwindigkeiten errechnet werden, die bei zunehmender Nähe gegen Null konvergieren. Sollen die berechneten Geschwindigkeiten zur Steuerung des Roboters verwendet werden, anstatt sich stets mit konstanter Geschwindigkeit zu bewegen oder diese über ein externes Verfahren zu bestimmen, so sollte bei der Modellierung der Funktionsparameter der Objekte auf den gewünschten Wertebereich der Geschwindigkeiten Rücksicht genommen werden. Da die hier erzeugte Geschwindigkeit keine Maßeinheit hat, muss der an die Verhaltenssteuerung angebundene Ergebnis-Konverter (siehe Abschnitt 2.1.2) eine geeignete Abbildung auf den zulässigen Wertebereich der Ausführungsschicht vornehmen.

4.1.3 Parametrisierung

Da das Verfahren zur Bewegungsberechnung unveränderlich und nicht direkt parametrisierbar ist, verbleibt die Modellierung der Objekte als die einzige Möglichkeit zur Beeinflussung des Bewegungsverhaltens. Eine ansonsten unveränderte Umgebung kann somit bei veränderten Feldparametern verschiedene Bewegungsfolgen hervorrufen. Dies ist auch in Abbildung 4.2 zu sehen.

4.1.4 Berechnung des Aktivierungswertes

Die Bestimmung einer Bewegung über ein Potentialfeld entspricht einem einzelnen Verhalten innerhalb der in Abschnitt 2.2.2 beschriebenen Architektur. Daher muss auch ein Aktivierungswert berechnet werden können, der die Dringlichkeit der Ausführung des Bewegungsverhaltens kennzeichnet. Es sind dazu bisher zwei Verfahren implementiert worden:

Zuweisung eines konstanten Aktivierungswertes

Die Zuweisung eines konstanten Aktivierungswertes führt zu einer, unabhängig vom Zustand der Umwelt, gleichbleibenden Bewertung. Dies kann in zwei ver-

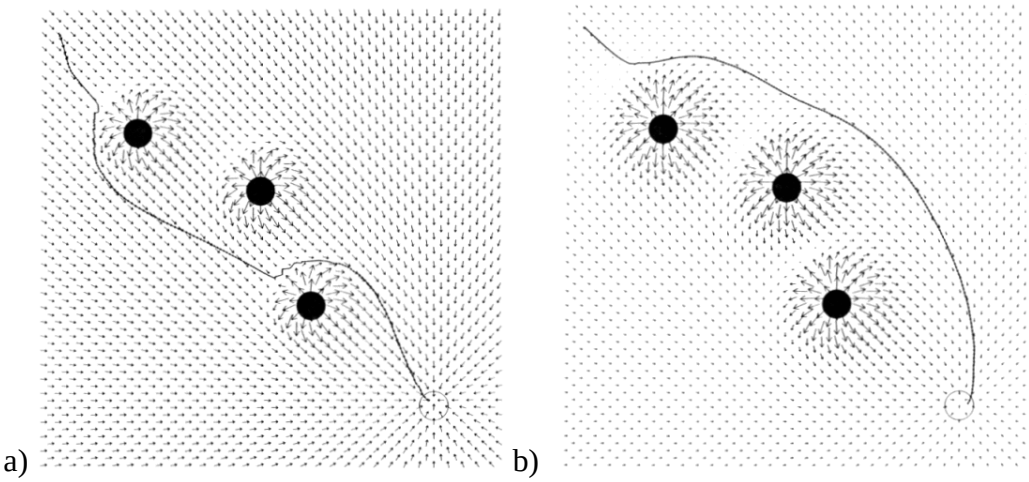


Abbildung 4.2: *Auswirkung der Objekt-Parameter auf die Bewegungsrichtung. a) Die Zielposition ist stark anziehend und die zu meidenden Objekte nur schwach abstoßend, b) die Zielposition ist schwach anziehend und die Objekte stark abstoßend. (Die Bilder wurden entnommen aus [Arkin, 1998])*

schiedenen Szenarien sinnvoll sein. Zum einen kann durch eine solche Zuweisung ein Verhalten zum Standardverhalten deklariert werden, das immer dann ausgeführt wird, wenn kein anderes Verhalten einen höheren Aktivierungswert hat. So kann beispielsweise ein Torwart immer auf ein Positionierungsverhalten zurückgreifen, solange die Verhalten zum Abfangen oder Schießen des Balls nicht durchführbar sind oder sehr schlecht bewertet werden. Zum anderen kann ein besonders niedriger Aktivierungswert zugewiesen werden, der eine direkte Auswahl des Verhaltens verhindert, um es dann später gegebenenfalls indirekt über das in Abschnitt 2.2.3 beschriebene Verfahren zur Verhaltenskombination auszuführen.

Negative Länge des Gradienten

Des Weiteren kann die negative Länge des Gradienten, also $-|\vec{v}|$, zur Bewertung eines Bewegungsverhaltens verwendet werden. Dadurch wird beispielsweise ein Ausweichverhalten besser bewertet, wenn sich der Roboter in der Nähe eines stark repulsiven Hindernisses befindet und eine entsprechende Ausweichbewegung dringend notwendig ist. Der Betrag des Vektors muss negiert werden, da für das Verfahren zur Verhaltensauswahl stets der niedrigste Wert die höchste Aktivierung darstellt.

4.2 Relativbewegungen

Zusätzlich zu den bisher beschriebenen Möglichkeiten, sich zu einem zuvor definierten Ziel hin- und von Hindernissen wegzubewegen, soll die Verhaltenssteuerung um Relativbewegungen erweitert werden, der Roboter sich also relativ zu einem oder mehreren anderen Objekte positionieren, beziehungsweise gemeinsam mit anderen Robotern Formationen bilden können.

4.2.1 Autonomes Bilden von Formationen

Mittels der Modellierung repulsiver und attraktiver Felder ist es möglich, Gruppen von Robotern derart zu koordinieren, dass sie sich an einem bestimmten Punkt sammeln oder sich gleichmäßig über eine Fläche verteilen. Die Ausrichtung in konkreten Formation, wie das dynamische Bilden und Beibehalten einer Reihe, ist mit den bisher vorgestellten Verfahren nicht möglich.

Hierzu wird in [Reif and Wang, 1995] ein theoretisches Modell vorgestellt, in dem die angestrebten Positionen der Roboter in einer Formation auf einen Graphen abgebildet werden, in dem die Kanten durch soziale Funktionen repräsentiert werden, die die zum Erreichen der Formation nötigen Bewegungsvektoren erzeugen. Ein bereits in zahlreichen Versuchen erprobtes Verfahren wird von [Balch and Arkin, 1999] verfolgt. Basierend auf dem Ansatz der Motor-Schemata [Arkin, 1989] ist eine Anzahl fest definierter Formationen, wie eine Linie oder eine V-Form, jeweils als ein spezielles Schema implementiert worden, welches durch Überlagerung in ein Gesamtverhalten integriert werden kann. Beiden Ansätzen gemein ist das Konzept eines oder mehrerer *Führungsroboter*, die sich eventuell untereinander, aber nicht mit den anderen Robotern, koordinieren und deren Position als Referenz zur Bildung der Formation verwendet wird.

4.2.2 Integration in die Bewegungssteuerung

Das im Rahmen dieser Diplomarbeit verwendete Verfahren orientiert sich an [Balch and Arkin, 1999], indem spezielle Objekte zur Erzeugung von Relativbewegungen verwendet werden, die einem Verhalten, ebenso wie die in Kapitel 3 beschriebenen Objekte, beliebig zugeordnet werden können.

Es handelt sich dabei jedoch um keine fest vorgegebenen Formationen, sondern vielmehr um allgemeine Vorgaben zur Positionierung relativ zu anderen Objekten, mittels derer sich beliebige Formationen zusammensetzen lassen. Dieses Vorgehen

ermöglicht sowohl das gemeinsame koordinierte Bewegen einer Gruppe von Robotern als auch eine Positionierung im Bezug auf Objekte, wie beispielsweise einen Ball oder ein Tor in einem Fußballspiel.

Im Folgenden werden die implementierten Verfahren im Detail vorgestellt. Dabei ist zu beachten, dass jeweils die relative Nähe zu anderen Objekten nicht behandelt wird, da diese über deren eigene Potentialfelder gesteuert wird.

4.2.3 Positionierung zwischen anderen Objekten

Zur Positionierung zwischen anderen Objekten existieren aus Sicht der Modellierung zwei verschiedene Beschreibungsmöglichkeiten. Dies ist zum einen die Positionierung zwischen zwei Objekten (entsprechend dem englischen *between*) sowie zum anderen das Positionieren innerhalb einer Menge von Objekten (entsprechend dem englischen *among*). Da das interne Berechnungsverfahren jeweils gleich ist, werden sie an dieser Stelle gemeinsam beschrieben.

Der Bereich, den ein Roboter erreichen muss, um der gewünschten Relation zu entsprechen, kann als eine geometrische Figur beschrieben werden. Durch das Hinzufügen einer Funktion, die den Verlauf der Potentialfunktion sowie die Längen der Bewegungsvektoren bestimmt, kann die Positionierung zwischen Objekten auf die in Abschnitt 3.3.2 beschriebenen Felder um geometrische Objekte abgebildet werden. Dies hat zum einen den Vorteil der Wiederverwendung bereits vorhander Komponenten, zum anderen ergibt sich die Möglichkeit, neben einem Potentialfeld auch den Wert der Potentialfunktion berechnen zu können, eine für die Vermeidung lokaler Minima elementare Eigenschaft (siehe Abschnitt 4.3.2).

Es verbleibt die Bestimmung des von mehreren Objekten begrenzten Bereiches. Wie in Abbildung 4.3 zu sehen ist, kann um eine Menge von Objekten herum eine konvexe Hülle gebildet werden. Dazu ist es nötig, zunächst eine Punktmenge zu erzeugen, aus der dann die späteren Eckpunkte eines konvexen Polygons bestimmt werden können. Diese Menge besteht aus den signifikanten Punkten der geometrischen Figuren der Objekte, zwischen die sich der Roboter begeben soll, also den End- und Eckpunkten der Strecken und Polygone, den Mittelpunkten der Kreise[1] sowie den Positionen der Objekte ohne geometrische Form. Zur Bestimmung der konvexen Hülle einer Punktmenge ist zunächst das Verfahren des *Einwickelns* aus [Sedgewick, 1992] implementiert worden. Aufgrund der in der Testdomäne sehr geringen Anzahl von Objekten ist dessen Aufwandsklasse zu vernachlässigen. Bei einem Einsatz in komplexeren Domänen sowie bei einer eventuellen Erweiterung

[1] Durch die Verwendung des Kreismittelpunktes an Stelle einer Annäherung der gesamten Kreisform entspricht dieses Verfahren nicht der Bildung einer korrekten konvexen Hülle, ist aber sehr effizient durchzuführen.

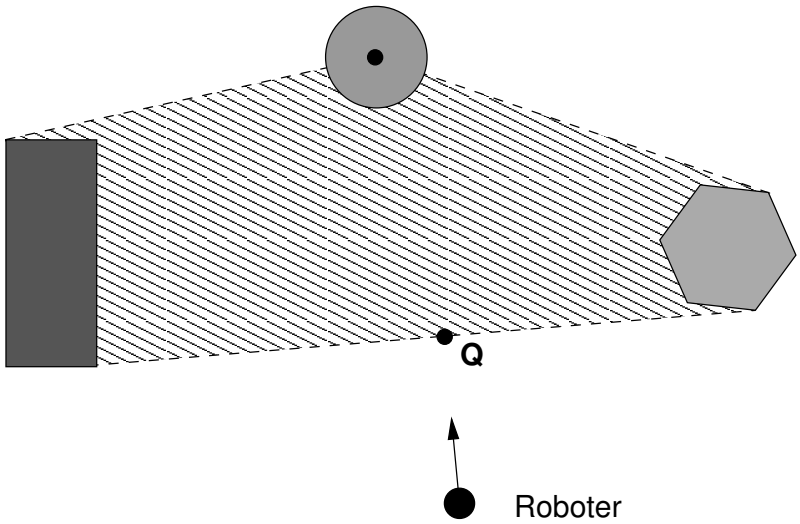


Abbildung 4.3: *Durch die Bildung der konvexen Hülle um eine Menge von Objekten kann ein Potentialfeld um eine geometrische Figur erzeugt werden. Durch dieses Feld kann ein Bewegungsvektor zu einer Position Q bestimmt werden, die zwischen den Objekten liegt.*

der Verhaltenssteuerung auf drei Dimensionen sollte ein effizienteres Verfahren eingesetzt werden.

Da sich die Positionen der referenzierten Objekte laufend verändern können, beziehungsweise nicht zu jedem Zeitpunkt vollständig bekannt sind, muss die Berechnung des konvexen Polygons bei jedem Aufruf des Bewegungsverhaltens neu durchgeführt werden. Zur Positionierung zwischen anderen Objekten müssen mindestens die Positionen zweier dieser Objekte bekannt sein. Ist dies nicht der Fall, wird lediglich ein Nullvektor erzeugt, da eine relative Positionierung nicht möglich ist.

Auch bei der Bewegung zwischen zwei Objekte kann ein konvexes Polygon erzeugt werden, falls beispielsweise beide Objekte über Polygone beschrieben worden sind, wie in Abbildung 4.4a zu sehen ist. Handelt es sich bei den Objekten um Kreise, besitzen sie keine geometrische Form oder ergeben sich einige spezielle Anordnungen von Strecken, wird anstelle eines Polygons eine Strecke zwischen den Objekten erzeugt, siehe Abbildung 4.4b.

Abschließend sei darauf hingewiesen, dass durch dieses Positionierungsverfahren im Inneren einer angesteuerten Regionen, entsprechend den Definitionen in Abschnitt 3.3.2, kein Potentialfeld erzeugt wird. Ferner kann durch die Zuordnung einer repulsiven Funktion ein gegenteiliges Verhalten modelliert werden, bestimmte dynamische Regionen also gemieden beziehungsweise verlassen werden.

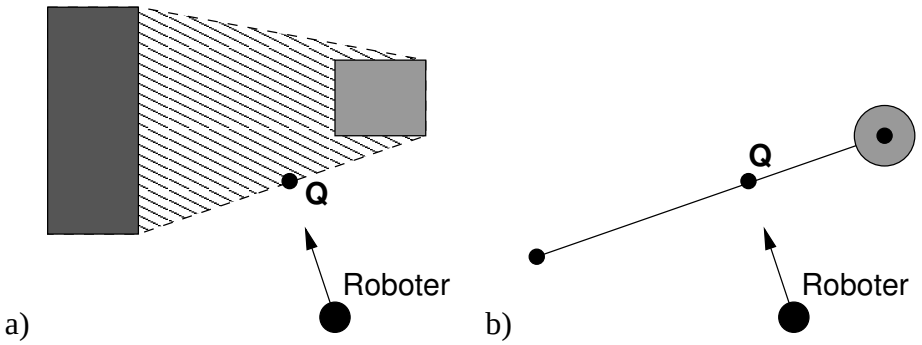


Abbildung 4.4: *Bei der Positionierung zwischen zwei Objekten ist es sowohl möglich, dass die Zielregion a) durch ein konvexes Polygon als auch b) durch eine Strecke beschrieben wird.*

4.2.4 Einen relativen Winkel einnehmen

Die zweite implementierte Relativbewegung ist die Positionierung in einem relativen Winkel zu einem anderen Objekt, die es beispielsweise ermöglicht, mehrere Roboter in einer Linie oder hintereinander anzuordnen. Mittels einer Vielzahl verschiedener Winkelrelationen ist theoretisch der Aufbau beliebiger komplexer Formen durch mehrere Roboter möglich.

Ebenso wie im vorherigen Abschnitt wird das Verfahren zur Bestimmung der Bewegungsrichtung auf ein Feld um ein geometrisches Objekt abgebildet. Dabei wird, ausgehend von dem Objekt, zu dem die relative Positionierung erfolgen soll, eine Strecke konstruiert, auf der sämtliche Punkte in einem zuvor festgelegten Winkel α zum Objekt liegen.

Dieser Winkel kann sowohl in Bezug auf ein absolutes als auch auf das relative Koordinatensystem des Objekts angegeben werden. In einem absoluten Koordinatensystem haben alle Punkte auf einer Halbgeraden, die ausgehend vom Objekt parallel zur x-Achse verläuft, einen Winkel von $0°$ zum Objekt. Entsprechend könnten zum Beispiel Roboter in einem Fußballspiel, mittels einer relativen Ausrichtung von $\pm 90°$ zueinander, nebeneinander quer über das Feld positioniert werden. Wird ein relatives Koordinatensystem verwendet, so entspricht ein Winkel von $0°$ einer Position in Richtung der aktuellen Rotation des Objekts. Ist diese dem Roboter bekannt, ist es beispielsweise möglich, sich stets rechts, links oder hinter einem Objekt zu positionieren.

Die Länge der konstruierten Strecke wird automatisch bestimmt, so dass von der Position des Roboters immer eine orthogonale Projektion auf einen Punkt Q auf der Strecke möglich ist, wie in Abbildung 4.5 zu sehen ist.

Ist die Position des referenzierten Objekts nicht bekannt, also in der internen Model-

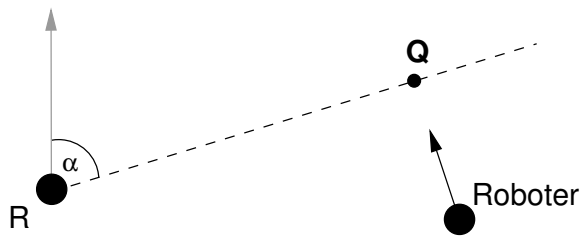


Abbildung 4.5: *Zur Positionierung in einem relativen Winkel α zu einem Objekt R wird eine entsprechende Strecke konstruiert, die ein Potentialfeld erzeugt, welches den Roboter zu dem nächstgelegenen Punkt Q auf der Strecke führt.*

lierung der Verhaltenssteuerung deaktiviert, kann keine Bewegung berechnet werden und es wird ein Nullvektor erzeugt.

4.2.5 Auswahl zwischen verschiedenen Relativbewegungen

Des Weiteren ist es möglich, eine Reihe von Relativbewegungen zu beschreiben und die jeweils günstigste auszuwählen. Innerhalb einer Gruppe mehrerer Roboter, die eine komplexe Formation bilden sollen, können dadurch beispielsweise mehrere redundante Relationen zu verschiedenen Robotern beschrieben werden, um dann, auch unter Berücksichtigung der Tatsache, dass ein Teil der Relativbewegungen nicht möglich ist, da die referenzierten Objekte deaktiviert sind, die momentan beste auszuwählen. Die möglichen Bewertungskriterien für Relativbewegungen sind folgende:

Minimale Entfernung Da die Relativbewegungen auf geometrische Figuren abgebildet werden, lässt sich die Entfernung vom Roboter zu den entsprechenden Regionen berechnen, so dass bei der Auswahl die nächstgelegene Position bevorzugt werden kann.

Maximaler Gradient Die Auswahl über den maximalen Gradienten entspricht dem in Abschnitt 4.1.4 beschriebenen Bewertungsverfahren, hier allerdings unter Verwendung positiver Beträge.

Minimaler Gradient Sind sämtlichen Relativbewegungen quadratische Funktionen zugeordnet worden, die in der Nähe eines Objekts gegen Null konvergieren, entspricht die Bewegung mit dem geringsten Gradienten der nächstgelegenen Region.

Maximale Priorität Bei der Modellierung können einzelnen Relativbewegungen Prioritäten zugeordnet werden. Dies ermöglicht bei der späteren Auswahl eine Bevorzugung der Bewegung mit der höchsten Priorität.

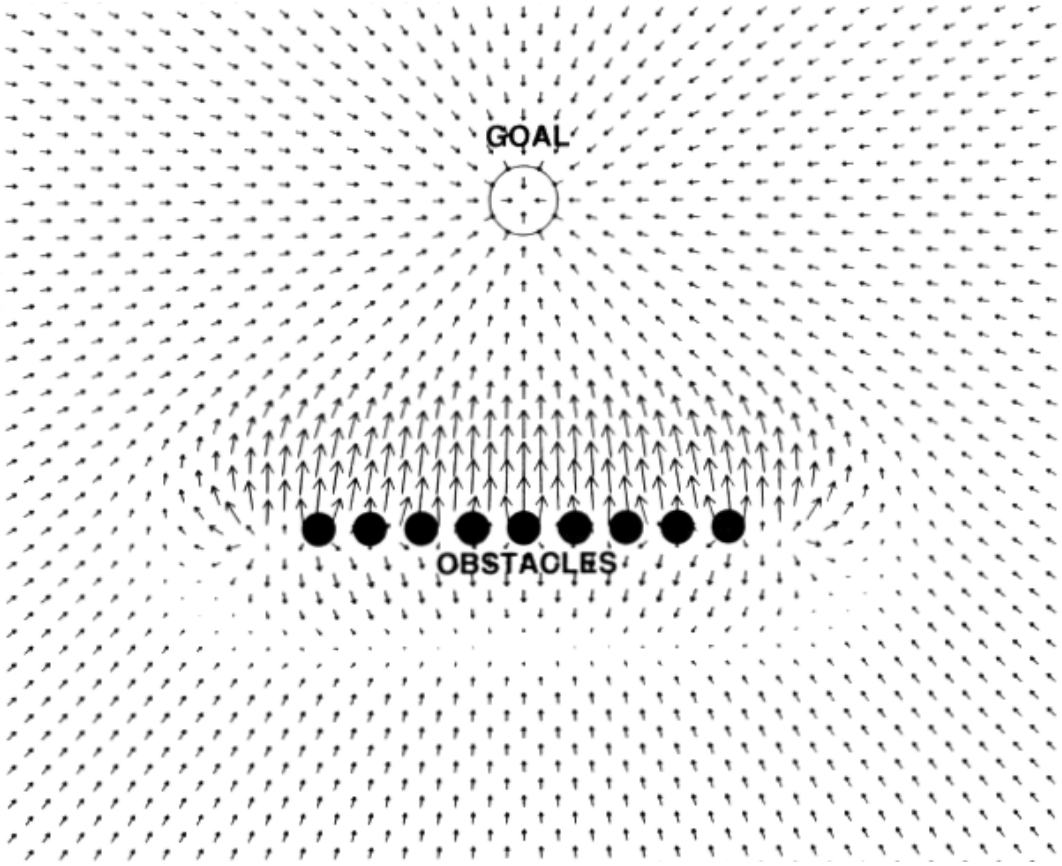


Abbildung 4.6: *Auf der der Zielposition abgewandten Seite der Hindernisse befindet sich ein Bereich, in dem die Vektorlängen gegen Null konvergieren. (Bild entnommen aus [Arkin, 1998])*

Ebenso wie bei allen anderen Relativbewegungen, wird in dem Fall, dass keine der Bewegungen ausgewählt werden kann, ein Nullvektor erzeugt.

4.3 Behandlung lokaler Minima

Lokale Minima stellen ein dem Potentialfeld-Ansatz inhärentes Problem dar. Dieser Abschnitt beschreibt ein Verfahren, welches die Navigation aus einem solchen Minimum ermöglicht.

4.3.1 Das Problem lokaler Minima

Aus dem in Abschnitt 4.1.1 beschriebenen Verfahren zur Berechnung der Bewegungsrichtung ergibt sich ein zentrales Problem des Potentialfeld-Ansatzes. Es können Konstellationen existieren, in denen sich die Feldvektoren der einzelnen Objekte bei der Addition gegenseitig aufheben und das Ergebnis ein Vektor der Länge Null ist. Aus einem solchen Vektor können weder eine Geschwindigkeit noch eine Bewegungsrichtung bestimmt werden, so dass der Roboter die aktuelle Position nicht mehr verlassen kann. In Abbildung 4.6 ist eine solche Situation dargestellt.

Die Bezeichnung *lokales Minimum* bezieht sich dabei auf den Wert der aufsummierten Potentialfunktionen der Objekte an solchen Positionen. In der Praxis befindet sich ein Roboter selten exakt in lokalen Minima, da diese, zumindest in diesem Ansatz, lediglich die Ausdehnung eines Punktes oder einer Geraden haben können. Der Gradient der Potentialfunktion ist in der näheren Umgebung allerdings stets zum Minimum gerichtet, so dass der Roboter, dessen Bild von der Umwelt zumeist zusätzlich Schwankungen in den Sensormessungen unterworfen ist, um eine Position oszilliert.

Dieses Problem ist schon seit der ersten Beschreibung des Potentialfeld-Ansatzes in [Khatib, 1986] bekannt und wurde seither, unter anderem in [Koren and Borenstein, 1991], ausführlich diskutiert. Letztere zeigen zudem das Problem enger Passagen auf, welches beispielsweise beim Roboterfußball häufig in der Form auftritt, dass zwei andere Roboter derart nah zueinander stehen, dass zwischen ihnen zwar theoretisch ein möglicher Weg wäre, ihre repulsiven Felder jedoch eine Passage unmöglich machen. Dies kann als der Spezialfall eines lokalen Minimums angesehen werden.

Es existiert keine allgemeine Standardlösung, wohl aber diverse Strategien, mit diesem Problem umzugehen. Ein sehr simples, in [Arkin, 1998] beschriebenes Verfahren ist die permanente Addition eines Zufallsvektors zu dem errechneten Bewegungsvektor, um sich somit von einer stabilen Position in einem lokalen Minimum zu entfernen. Dies führt jedoch zu einem suboptimalen Verhalten außerhalb lokaler Minima und ermöglicht zudem lediglich das Verlassen eines punktförmigen lokalen Minimums. In [Balch and Arkin, 1993] werden lokale Minima durch die Vermeidung zuvor besuchter Positionen vermieden. In Domänen wie dem Roboter-Fußball, in denen sich die Zielposition des Roboters sowie dessen Umgebung permanent verändert, erscheint dies allerdings nicht sinnvoll. In derartigen hochdynamischen Umgebungen werden lokale Minima mitunter auch unter der Annahme der schnellen Auflösung eines Minimums gänzlich ignoriert [Vail and Veloso, 2003].

Derartig reaktive, nur die lokale Umgebung des Roboters betrachtende, Verfahren können zwar in einigen Fällen erfolgreich ein lokales Minimum überwinden, garantieren jedoch keinen Erfolg. Als einziges garantiert effektives Verfahren gilt die

```
function A*-SEARCH (start, goal) returns nextPosition
    searchTree ← empty
    currentNode ← start
    while not REACHED (currentNode, goal)
    do
        newNodes ← EXPAND(currentNode)
        ESTIMATECOSTS(newNodes,currentNode,goal)
        APPEND(searchTree, newNodes)
        currentNode ← FINDBESTNODE(searchList)
    end
return BEGINNINGOFPATHTO (currentNode)

function ESTIMATECOSTS (nodes, lastNode, goal)
    for every node in nodes
    do
        g ← COMPUTEPATHCOSTS(node, lastNode)
        h ← COMPUTEHEURISTICTO(node, goal)
        ASSIGNCOSTS (node, g, h)
    end
```

Abbildung 4.7: *Der allgemeine A*-Algorithmus*

Planung eines vollständigen Pfads zum Ziel unter der Betrachtung der vollständigen Umgebung, welches daher auch im Rahmen dieser Diplomarbeit umgesetzt wurde.

4.3.2 Wegplanung mit dem A*-Algorithmus

Bereits in [Latombe, 1991] wird die Verwendung einer Bestensuche zur Überwindung lokaler Minima beschrieben. Aufgrund des sehr hohen Rechenaufwands, im Vergleich zur Navigation mit dem Potentialfeld-Verfahren, sind in der Vergangenheit zumeist reaktive Verfahren, wie die im vorherigen Abschnitt erläuterten, bevorzugt worden. Durch die Verfügbarkeit größerer Rechenkapazitäten und den Entwurf effizienter Suchstrategien ist es allerdings inzwischen möglich, eine vollständige Wegplanung für einen Roboter in Echtzeit durchzuführen, siehe [Behnke, 2004].

Das allgemein bevorzugte Suchverfahren ist der A*-Algorithmus von [Hart et al., 1968], ein vollständiges und optimales Verfahren zur informierten Bestensuche. Der Algorithmus erzeugt, beginnend mit einem Knoten, der die aktuelle Position repräsentiert, einen Suchbaum, der solange erweitert wird, bis ein Knoten gefunden

worden ist, der den Zielkriterien entspricht (siehe Abbildung 4.7). Zur Erweiterung des Suchbaums wird stets der Knoten mit den niedrigsten Kosten ausgewählt, der noch nicht erweitert worden ist. Abhängig von der Struktur des Suchraums und dem verwendeten Erweiterungsverfahren werden die Nachbarknoten dieses Knotens bestimmt, deren Kosten berechnet und sie im Baum unterhalb des zuletzt gewählten Knotens eingehängt. Die Kosten eines Knotens setzen sich aus zwei Komponenten zusammen: Den bisherigen Pfadkosten zu diesem Knoten sowie den geschätzten, noch anfallenden Kosten zum Ziel. Diese Kombination ermöglicht, verglichen mit anderen Algorithmen, eine sehr effiziente Suche. Ausführlichere Informationen, zusammen mit Beweisen der einzelnen Eigenschaften, finden sich in [Russell and Norvig, 1995].

Zur Verwendung des A*-Algorithmus für die Wegplanung im Potentialfeld müssen mehrere Komponenten des Verfahrens speziell an die Umgebung angepasst werden. Dies sind die Erweiterung der Knoten, die Berechnung von Pfadkosten sowie das Schätzen zukünftiger Kosten mittels einer geeigneten Heuristik.

Dynamische Diskretisierung des Raumes

Da der Algorithmus Wege als eine Abfolge von diskreten Positionen, die jeweils einem Knoten im Suchbaum entsprechen, betrachtet, der Roboter sich jedoch in einem kontinuierlichen Raum bewegt, muss eine Diskretisierung vorgenommen werden, mittels der Punkte in der Umgebung des Roboters bestimmt werden können. Die Erweiterung eines Knotens entspricht somit der Bestimmung einer endlichen Menge neuer Positionen in seiner Umgebung.

Eine naheliegende Strategie ist die Rasterung der Umgebung [Latombe, 1991] durch ein festes Gitter, welches eine Aufteilung des Raumes in eine Menge gleichgroßer Zellen bewirkt, die jeweils eine fest definierte Anzahl an Nachbarzellen haben. Dieses Vorgehen bringt jedoch mehrere, sich gegenseitig bedingende, Probleme mit sich. Bevor mit der Wegplanung begonnen werden kann, muss die Zellgröße in Abhängigkeit vom gewünschten Detailgrad und dem, für die Repräsentation der Gesamtmenge der Zellen benötigten, zur Verfügung stehenden Arbeitsspeicher fest gewählt werden. Dies kann einerseits zu einer sehr groben, für die Repräsentation der näheren Umgebung des Roboters nicht ausreichenden, Zellgröße führen, aber auch eine sehr große Menge von Zellen zur Folge haben, die eine Suche nach einem Weg ineffizient werden lässt. Zur Umgehung dieser Probleme ist von [Behnke, 2004], aufbauend auf der Arbeit von [Kambhampati and Davis, 1986], ein Verfahren zur Wegplanung mit dem A*-Algorithmus in einem Potentialfeld entwickelt worden, das mit verschiedenen Zellgrößen arbeitet. Durch die Verwendung sehr kleiner Zellen in der Nähe des Roboters und großer Zellen zur Repräsentation ferner Regionen, deren Zustand zumeist ohnehin nicht exakt zu bestimmen ist und sich

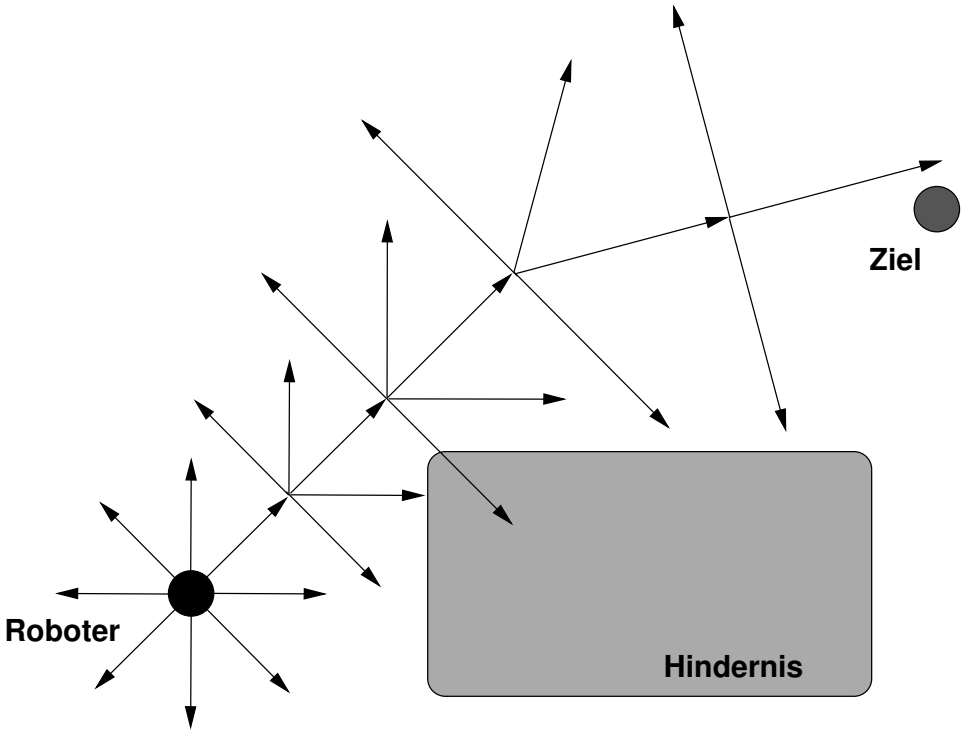


Abbildung 4.8: *Ein dynamischer Suchbaum. Mit zunehmender Entfernung vom Ausgangspunkt der Suche nimmt die Anzahl der Verzweigungen ab und die Länge der Kanten zu. Des Weiteren sind einige Kanten durch Optimierungsverfahren entfernt worden.*

bis zur Ankunft des Roboters wahrscheinlich verändert, konnte eine echtzeitfähige Wegplanung realisiert werden.

Der im Rahmen dieser Diplomarbeit implementierte Ansatz verzichtet auf eine Rasterung der Umgebung und erzeugt, ähnlich dem Verfahren der *Rapidly-Exploring Random Trees*, unter anderem beschrieben in [Bruce and Veloso, 2002], eine dynamische Baumstruktur. Durch die Bestimmung der Kantenlängen und des Verzweigungsgrades in Abhängigkeit von der Entfernung zum Ausgangspunkt der Suche kann, ebenso wie in [Behnke, 2004], eine Diskretisierung mit verschiedenen Auflösungen erzielt werden, ohne allerdings zuvor eine feste Anzahl an Auflösungen definieren zu müssen. Abbildung 4.8 zeigt einen solchen Baum.

Zur Steuerung des Prozesses der Wegplanung ist bei der Modellierung eine Reihe von Parametern anzugeben: Die minimale und maximale Anzahl an Verzweigungen pro Knoten[2], b_{min} und b_{max} mit $b_{min} \leq b_{max}$, die minimale und maximale Länge der Kanten des Baums, r_{min} und r_{max} mit $r_{min} \leq r_{max}$, der Radius r_{near} des Nahbe-

[2]Durch spätere Optimierungen zur Verkleinerung des Suchbaums kann die tatsächliche Anzahl an Verzweigungen unter dem vorgegebenen Mindestwert liegen.

reichs des Roboters sowie der Radius r_{far}, der eine Entfernung beschreibt, jenseits der ausschließlich die vorgegebenen Maximalwerte verwendet werden.

Mittels dieser Parameter können bei der Erweiterung eines Knotens die Anzahl b sowie die Entfernung r der neu zu erzeugenden Knoten, in Abhängigkeit von der Entfernung d zum Anfangspunkt der Suche, bestimmt werden. Es gilt:

$$b = \begin{cases} b_{min} & , d \leq r_{near} \\ b_{max} & , d \geq r_{far} \\ b_{max} - (\frac{d - r_{near}}{r_{far} - r_{near}})(b_{max} - b_{min}) & , sonst \end{cases} \tag{4.3}$$

$$r = \begin{cases} r_{min} & , d \leq r_{near} \\ r_{max} & , d \geq r_{far} \\ r_{min} + (\frac{d - r_{near}}{r_{far} - r_{near}})(r_{max} - r_{min}) & , sonst \end{cases} \tag{4.4}$$

Nach der Berechnung der Werte werden b neue Knoten, jeweils im Abstand von $2\pi/b$, auf einem Kreis mit dem Radius r um den aktuellen Knoten erzeugt.

Um die Anzahl der Knoten im Suchbaum möglichst gering zu halten und eine möglichst gleichmäßige Verteilung zu forcieren, sind zwei Optimierungen hinzugefügt worden. Da die erweiterten Knoten stets auf Kreisen liegen, kommt es bei der Erweiterung eines Knotens zu einer Überlappung mit dem Kreis des Vaterknotens. Innerhalb dieses Bereichs müssen allerdings keine neuen Knoten mehr erzeugt werden, da dieser als bereits abgedeckt gilt. Siehe hierzu Abbildung 4.9. Die gleiche Eliminierungsstrategie wird zudem auf alle bisher erweiterten Knoten angewendet. Dadurch kann vermieden werden, dass sich in einer Region mehrere Teilbereiche des Suchbaums überlagern. Während der Test in Bezug auf den Vaterknoten sehr effizient über relative Winkel durchgeführt werden kann, muss im zweiten Fall eine Liste über alle bisher erweiterten Knoten geführt werden, die dann auf eine eventuelle Überlappung getestet werden können. Dies stellt zwar einen zusätzlichen Rechenaufwand dar, führt aber zu einem deutlich kleineren Suchbaum und ermöglicht dadurch eine insgesamt effizientere Suche.

Berechnung der Pfadkosten

Die Pfadkosten g zu einem Knoten an der Position P_k setzen sich zusammen aus dem Pfadkosten g_p des Vaterknotens an der Position P_p sowie den, von einer Funktion k berechneten Kosten für den Weg von P_p zu P_k:

$$g = g_p + k\,(P_p, P_k) \tag{4.5}$$

Für die Wurzel des Suchbaums gilt $g = g_p = 0$.

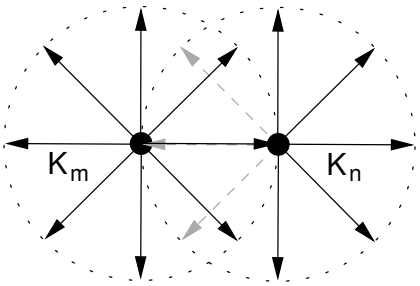


Abbildung 4.9: *Bei der Erweiterung des Knotens K_n werden keine neuen Knoten erzeugt, die innerhalb des bereits vom Vaterknoten K_m abgedeckten Bereichs liegen.*

Sei d die Entfernung zwischen den beiden Knoten sowie δ die Differenz der Werte der aufsummierten Potentialfunktionen an den beiden Positionen[3]:

$$\delta = \varphi(P_p) - \varphi(P_k) \tag{4.6}$$

$$d = |\vec{P_p P_k}| \tag{4.7}$$

Dann kann die Funktion k folgendermaßen definiert werden:

$$k = \begin{cases} d & ,\delta \leq 0 \\ d + \delta & ,\delta > 0 \end{cases} \tag{4.8}$$

Zur Berechnung der Pfadkosten wird immer zumindest die Länge d der zurückgelegten Strecke verwendet. Ist der Wert der Potentialfunktion an der Position P_k höher als an P_p, so ist dies als eine Verschlechterung zu bewerten und die Differenz δ wird zu den Kosten addiert. Dies ist darin begründet, dass die Potentialfunktion an der Zielposition im Allgemeinen den niedrigsten Wert hat; bei größerer werdender Entfernung oder in der Nähe von Hindernissen nimmt dieser Wert zu. Man kann sich bildlich vorstellen, dass ein Aufstieg zu einer höheren Position aufwändiger ist, als der Abstieg in ein Tal.

Es sei darauf hingewiesen, dass der A*-Algorithmus bei diesem Verfahren zur Berechnung von g letztendlich nicht den kürzesten Weg zum Ziel sucht, sondern den günstigsten in Abhängigkeit von den Potentialfunktionen der Objekte in der Umgebung.

Die Heuristik

Zur Abschätzung der noch folgenden Kosten h auf dem Weg von P_k zur Zielposition Z wird eine Heuristik benötigt, die möglichst effizient versucht, den Wert h

[3]Die Möglichkeit der Bildung eines Gesamtpotentials $\varphi(P)$ an einer beliebigen Position P sei hier vorausgesetzt. Eine ausführliche Beschreibung erfolgt in Abschnitt 5.1.3

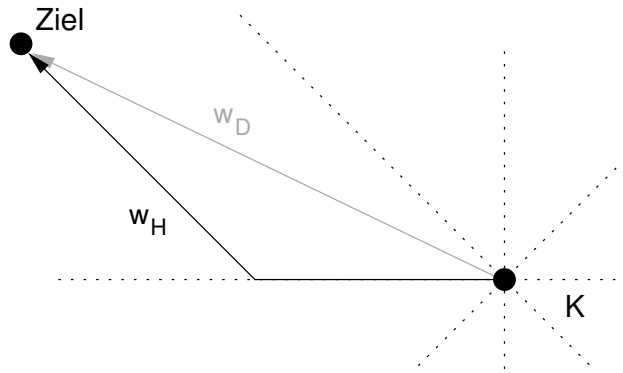


Abbildung 4.10: *Die Länge des verbleibenden Weges zum Ziel kann sowohl über die direkte Verbindung w_D als auch über einen zusammengesetzten Weg w_H entlang der Kanten des Suchbaums angenähert werden.*

an die tatsächlich zu erwartenden Kosten anzunähern. Eine wichtige Vorbedingung des A*-Algorithmus ist dabei jedoch die Zusicherung, dass h niemals die Kosten überschätzt.

Zur Einhaltung dieser Bedingung muss davon ausgegangen werden, dass die Werte der Potentialfunktion in Richtung des Ziels stetig fallen, die Kosten also nur von der Länge des zurückzulegenden Weges abhängen. In einer ersten Version wurde daher folgende Heuristik verwendet:

$$h = |\vec{P_k Z}| \tag{4.9}$$

Dies führte jedoch, im Zusammenspiel mit der verwendeten Funktion zur Berechnung der Pfadkosten, zu einer zu optimistischen Bewertung vieler Position, was eine starkes Anwachsen des Suchbaums zur Folge hatte. Um eine genauere Annäherung des Weges zu erreichen, wurde in der letztendlich verwendeten Heuristik die Tatsache berücksichtigt, dass der geplante Weg stets entlang der Kanten zwischen den Knoten verläuft. Interpretiert man diese Kanten als Vektoren, so muss die Zielposition entweder auf der Verlängerung eines Vektors liegen – in diesem Fall kann auch die zuvor beschriebene Heuristik verwendet werden – oder aber zwischen zwei Vektoren, so dass mittels dieser beiden Vektoren ein Weg berechnet werden kann, der länger ist, als die direkte Distanz, aber gleichzeitig niemals länger sein kann, als der später tatsächlich verlaufende Weg. Diese Konstruktion ist in Abbildung 4.10 zu sehen.

Seien $\vec{v_1}$ und $\vec{v_2}$ die Vektoren entlang zweier benachbarter Kanten, dann kann h folgendermaßen berechnet werden:

$$h = |n \cdot \vec{v_1}| + |m \cdot \vec{v_2}| \tag{4.10}$$

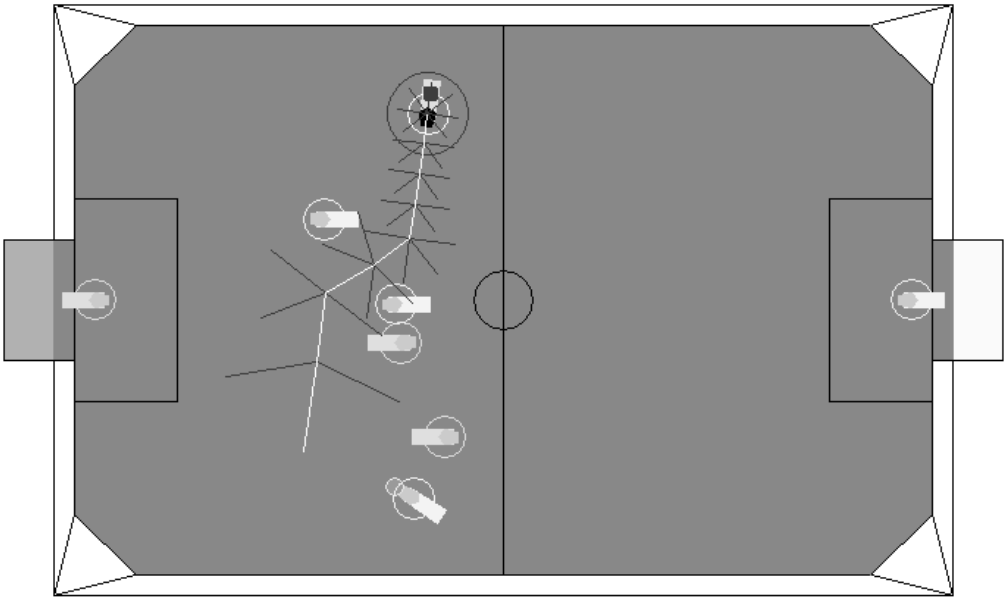


Abbildung 4.11: *Planung eines Weges zu einer relativen Position. In diesem Beispiel wird der günstigste Weg zur Positionierung eines Roboters zwischen Ball und Tor bestimmt. Die helle Linie beschreibt den gewählten Weg entlang der Kanten des Suchbaums, die dunklen Kanten kennzeichnen weitere evaluierte Wege.*

Für die Werte m und n muss dabei gelten:

$$n \cdot \vec{v_1} + m \cdot \vec{v_2} = \vec{P_k Z} \tag{4.11}$$

4.3.3 Einbettung und Verwendung der Wegplanung

Um die Wegplanung effektiv zur Bewegungssteuerung innerhalb der vorgestellten Architektur nutzen zu können, bedarf es noch einiger zusätzlicher Definitionen und Erweiterungen.

Da es theoretisch möglich ist, dass einem Feld mehrere attraktive Objekte und somit auch mehrere potentielle Ziele zugeordnet worden sein können, was im allgemeinen Fall nicht sinnvoll ist, aber für spezielle Aufgaben durchaus nötig sein kann, muss für die Verwendung der Wegplanung stets ein explizites Zielobjekt angegeben werden. Dabei kann es sich auch um ein Objekt zur Beschreibung einer Relativbewegung handeln, in diesem Fall wird als Zielposition die nächstgelegene Position innerhalb der angesteuerten Region verwendet, wie in Abbildung 4.11 zu sehen ist. Da eine Zielposition mit dem verwendeten Verfahren in den seltensten Fällen exakt erreicht werden kann, wird lediglich eine gewisse Nähe zum Ziel angestrebt; es

muss bei der Modellierung der Wegplanung eine maximale Entfernung angegeben werden, die zwischen einem Knoten und dem Ziel liegen darf, damit dieses noch als gefunden gelten kann.

Nachdem ein Pfad gefunden worden ist, kann ein absoluter Richtungsvektor $\vec{v}$ berechnet werden. Dieser entspricht im Suchbaum der Kante von der Wurzel zum nächsten Knoten auf dem Weg zum Ziel. Ein relativer Richtungsvektor sowie die Rotation können gemäß der Verfahren aus Abschnitt 4.1 bestimmt werden. Die Geschwindigkeit stellt allerdings einen Sonderfall dar, da sie bisher aus dem Gradienten des Feldes errechnet wurde und gerade dieser bei der Vermeidung lokaler Minima nicht mehr betrachtet werden darf. Die Wegplanung kann somit nur einen, bei der Modellierung anzugebenden, konstanten Wert für die Geschwindigkeit setzen.

Die Eigenbewegung des Roboters sowie Veränderungen in der Umwelt können starke Schwankungen des geplanten Weges und dadurch auch des erzeugten Vektors $\vec{v}$ zur Folge haben, da der Weg bei jedem Aufruf des Verhaltens neu berechnet wird. Zur Stabilisierung der Bewegungsrichtung ist daher ein Verfahren aus [Behnke, 2004] übernommen worden. Nach der Bestimmung eines Vektors wird hinter dem Roboter, entgegengesetzt zur aktuellen Bewegungsrichtung, ein repulsives Objekt eingefügt, welches eine Beibehaltung des zuletzt verwendeten Anfangsweges forciert. Hierzu kann bei der Modellierung ein beliebiges Objekt zusammen mit einer Entfernung definiert werden. In dem in Abbildung 4.11 gezeigten Beispiel ist ein solches Stabilisierungselement verwendet worden. Dessen Position ist als dunkler Punkt auf dem Rücken des Roboters gekennzeichnet.

Da auf einem Roboter-System stets nur eine begrenzte Menge an Arbeitsspeicher zur Verfügung steht, kann die maximale Anzahl der Knoten in einem Suchbaum angegeben werden, der entsprechende Speicher wird stets vor der ersten Ausführung der Wegplanung reserviert. Ist das Maximum erreicht, wird der bis zu diesem Zeitpunkt am besten bewertete Weg für die weiteren Berechnungen verwendet. Der verwendete Algorithmus ist jedoch dahingehend optimiert worden, einen möglichst kleinen Suchbaum zu erzeugen, so dass das Verfahren auch auf Systemen mit relativ wenig Speicher gute Ergebnisse erzielen kann. Genaue Zahlen zur Performanz und zum Ressourcenverbrauch der Wegplanung finden sich in Abschnitt 6.3. Abbildung 4.12 zeigt die Navigation auf einem Spielfeld der Sony-Liga.

4.3.4 Flexible Verwendung einer Wegplanung

Da die Planung eines vollständigen Weges in jedem Fall aufwändiger ist als das herkömmliche Potentialfeld-Verfahren und nicht auf jedem System eine hinreichende Rechenleistung permanent zur Verfügung steht, liegt es nahe, eine Wegplanung nur dann einzusetzen, wenn sich der Roboter in einem lokalen Minimum befindet, einer

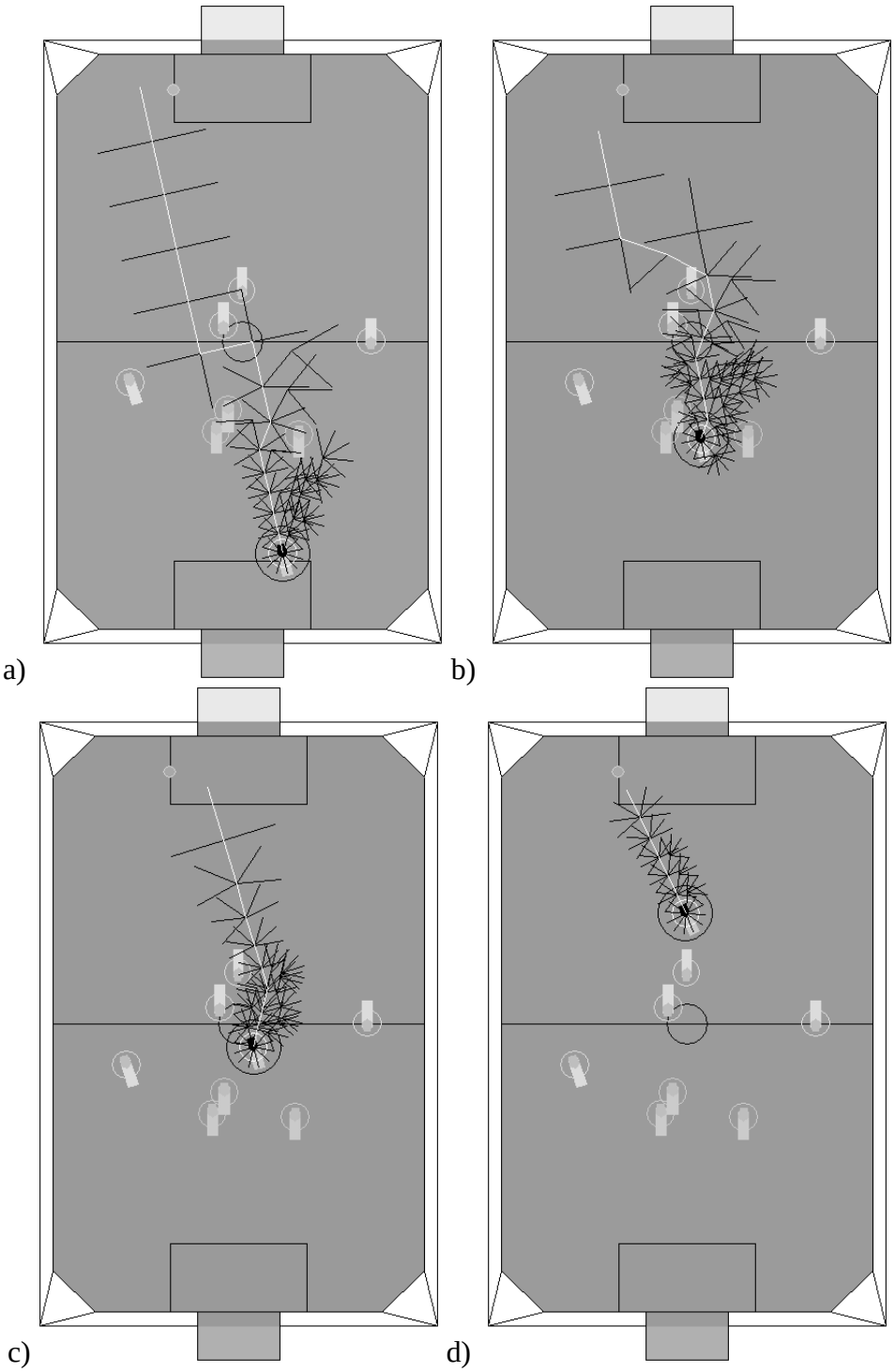


Abbildung 4.12: *Ein Beispiel der Wegplanung mit dem A*-Algorithmus: Ein Sony-Roboter plant einen Weg von einem Ende des Spielfelds zu einem Ball am anderen Ende. Dabei wird, wie im Verlauf der Abbildungen von a) bis d) zu sehen ist, permanent ein neuer Suchbaum aufgebaut.*

Situation, in der eine Wegplanung Priorität gegenüber der Performanz des Systems haben sollte. Um dies umzusetzen, müssen das Betreten und das Verlassen eines lokalen Minimums berechenbar sein.

Automatisches Erkennen eines lokalen Minimums

In einem lokalen Minimum ist die Steigung des Potentialfelds gleich Null. Diese Bedingung ist in der Praxis allerdings nicht ausreichend, eine derartige Position zu erkennen, da ein Minimum zumeist eine sehr kleine Ausdehnung hat und die Roboterposition um diese oszilliert. Es liegt somit nahe, einen Mindestwert $gradient_{min}$ für die Gradientenlänge anzugeben und ein lokales Minimum als ein Unterschreiten dieses Werts zu definieren.

Um ein lokales Minimum nicht mit einem absoluten Minimum, dem Ziel der Bewegung, zu verwechseln, muss zudem die aktuelle Entfernung $dist$ zum Ziel mit einer zuvor definierten Mindestentfernung $dist_{min}$ verglichen werden. Für die Bedingung $isMin$ gilt also:

$$isMin = (|\vec{v}| < gradient_{min}) \wedge (dist > dist_{min}) \tag{4.12}$$

Dieser Test ist sehr schnell zu berechnen und kann daher permanent durchgeführt werden.

Automatisches Erkennen des Verlassens eines lokalen Minimums

Um festzustellen, dass ein lokales Minimum wieder verlassen worden ist, ist das Zutreffen der Negation der vorherigen Bedingung, also $\neg isMin$, ein notwendiges aber kein hinreichendes Kriterium. Der Wert $gradient_{min}$ grenzt eine Region um das lokale Minimum herum ab. Nach dem Verlassen dieser Region ist jedoch nicht garantiert, dass der Roboter, dem Gradienten des Feldes folgend, sich nicht wieder in das Minimum bewegt. Dies kann ein Oszillieren der Roboterposition an der Grenze der durch $gradient_{min}$ beschriebenen Region zur Folge haben. Ein lokales Minimum kann erst dann als überwunden gelten, wenn sich der Roboter an einer Position befindet, von der aus er mit dem Verfahren des Gradientenabstiegs einen Weg zum Ziel finden kann.

Ausgehend von dieser Definition ist ein Algorithmus (siehe Abbildung 4.13) entwickelt worden, der überprüft, ob von der aktuellen Position aus ein Gradientenabstieg zur Zielposition möglich ist. Dies geschieht durch die schrittweise Bildung eines Vektorzuges, der aus einer Reihe von Vektoren einer festen Länge $step$, die jeweils entlang des Gradienten an ihrer Position ausgerichtet sind, besteht. Erreicht

function NOLOCALMINIMUM (*maxDist, step,* $\vec{start}$, $\vec{goal}$) **returns** boolean value
    $\vec{pos} \leftarrow \vec{start}$
    dist ← 0
    while (*dist* < *maxDist*) **and not** REACHED ($\vec{pos}, \vec{goal}$)
    do
        $\vec{g} \leftarrow$ COMPUTEGRADIENTAT($\vec{pos}$)
        $\vec{pos} \leftarrow \vec{pos} + \vec{g} \cdot \frac{step}{|\vec{g}|}$
        dist ← *dist* + *step*
    end
return REACHED ($\vec{pos}, \vec{goal}$)

Abbildung 4.13: *Der Algorithmus* NOLOCALMINIMUM *testet, ob sich ein Roboter nicht mehr in einem lokalen Minimum befindet.*

der Vektorzug nicht das Ziel, bevor seine Gesamtlänge einen Wert $maxDist$ überschritten hat, muss angenommen werden, dass sich der Roboter nach wie vor in einem lokalen Minimum, in dem die erzeugten Vektoren um eine einzelne Position oszillieren, befindet oder droht, sich in ein solches zu bewegen. Für den Wert $step$ wird in der derzeitigen Implementierung der minimale Expansionsradius r_{min} eines Knotens in einem Suchbaum verwendet. Als Obergrenze $maxDist$ wird das Doppelte der Länge des zuletzt geplanten Weges gesetzt.

Während der Roboter annimmt, in einem lokalen Minimum zu sein und die Wegplanung verwendet, kann dieser Test in regelmäßigen Abständen – beispielsweise in jedem Zyklus oder immer nach n Zyklen – durchgeführt und gegebenenfalls die Wegplanung wieder deaktiviert werden.

Das Verfahren könnte theoretisch in etwas abgewandelter Form auch zur Vorhersage lokaler Minima verwendet werden, es ist jedoch relativ rechenaufwändig und in einer hochdynamischen Umgebung wahrscheinlich nicht sehr effektiv.

4.4 Sonstige Erweiterungen

Dieser Abschnitt beschreibt einige implementierte Erweiterungen der Bewegungssteuerung. Dies sind Mechanismen zur Erzeugung zufälliger Bewegungsrichtungen, die Überlagerung von Bewegungen sowie die Möglichkeit der Glättung der erzeugten Vektoren.

4.4.1 Erzeugung zufälliger Richtungsvektoren

In manchen Anwendungsszenarien kann ein zufälliges Bewegungsverhalten von Nutzen sein, zum Beispiel bei der Exploration unbekannter Flächen oder zur Vermeidung lokaler Minima. Ebenso denkbar wäre eine Verwendung als Testumgebung für andere Komponenten wie die Selbstlokalisierung oder die Motorsteuerung.

Umgesetzt wurde diese Möglichkeit als eine Art Bewegungsgenerator, der in einem Bewegungsfeld aktiviert werden kann. Eine Implementierung als ein Objekt, ähnlich der Verfahren in Abschnitt 4.2, erschien aus mehreren Gründen als nicht sinnvoll. Zum einen existiert bei der Erzeugung eines zufälligen Richtungsvektors keine Potentialfunktion deren Wert berechnet werden könnte. Eine Zuordnung zu Aktionsfeldern oder eine Bewertung bei der A*-Suche sind somit nicht möglich. Zum anderen wird pro Bewegungsfeld ohnehin maximal ein Zufallsvektor benötigt. Zur Berechnung eines Bewegungsvektors kann der Bewegungsgenerator nun entweder allein verwendet werden oder zusammen mit einer Menge von Objekten, deren Felder dann von einem gewissen Unsicherheitsfaktor überlagert werden.

Die Länge eines erzeugten Vektors sollte in einem ähnlichen Wertebereich liegen, wie die der Feldvektoren der ansonsten verwendeten Objekte. Dieser Bereich kann durch zwei Parameter l_{min} und l_{max} gesetzt werden, so dass für einen zufällig erzeugten Vektor $\vec{v_r}$ gilt:

$$l_{min} \leq |\vec{v_r}| \leq l_{max} \tag{4.13}$$

Würde bei jedem Aufruf des Verhaltenssteuerung ein vollständig neuer Vektor berechnet werden, was beispielsweise auf einem Sony-Roboter bis zu 25 mal pro Sekunde geschehen kann, wäre die Bewegungsrichtung des Roboters extrem starken Schwankungen unterworfen und er würde sich im ungünstigsten Fall gar nicht bewegen. Es liegt also nahe, neue Vektoren nur in gewissen Zeitabständen und in Abhängigkeit von den vorherigen Vektoren zu erzeugen, um die Bewegung zu stabilisieren.

Zur Messung von Zeitabständen stehen zwei Maßeinheiten zur Verfügung: Die Anzahl der Aufrufe und die Zeit in Millisekunden. Dies ermöglicht es, den zuletzt erzeugten Vektor stets für n Einheiten beizubehalten und währenddessen keine neuen Vektoren zu erzeugen. Die verwendete Einheit muss bei der Modellierung angegeben werden und sollte abhängig von der Umgebung, in der die Potentialfeld-Architektur eingebettet ist, gewählt werden. Sämtliche Berechnungen und Bedingungen sind unabhängig von der Wahl der Maßeinheit.

Des Weiteren können Veränderungen von Richtung und Länge des Vektors in Abhängigkeit von der letzten Berechnung eingeschränkt werden. Es sei δ_{dir} die maximale Richtungsabweichung zwischen zwei nacheinander erzeugten Vektoren, $rand$

eine Funktion, die einen Zufallswert zwischen zwei Zahlen liefert und α_n die Richtung des n-ten erzeugten Vektors, dann gilt:

$$\alpha_n = rand(\alpha_{n-1} - \delta_{dir}, \alpha_{n-1} + \delta_{dir}) \tag{4.14}$$

Anschließend muss α_n auf das Intervall $[-\pi, \pi]$ normiert werden.

Es sei δ_{len} die maximale Längendifferenz zwischen zwei nacheinander erzeugten Vektoren. Für die Länge des n-ten erzeugten Vektors $\vec{v_n}$ gilt:

$$|\vec{v_n}| = rand(max(|\vec{v_{n-1}}| - \delta_{len}, l_{min}), min(|\vec{v_{n-1}}| + \delta_{len}, l_{max})) \tag{4.15}$$

4.4.2 Überlagerung von Freiheitsgraden

Jedes Ergebnis eines Feldes besteht unter anderem aus zwei Freiheitsgraden, die jeweils deaktiviert werden können: die Bewegungsrichtung und die Rotation. Zusammen mit der in Abschnitt 2.2.3 beschriebenen Kombination von Verhalten ergibt sich die Möglichkeit, einzelnen Freiheitsgraden verschiedene Verhalten zuzuordnen und dadurch ein komplexeres Bewegungsverhalten zu erzeugen. Eine weitere Aufteilung des Bewegungsvektors in zwei Freiheitsgrade, die x- sowie die y-Richtung, ist nicht vorgenommen worden, da die Kombination verschiedener Vektorkomponenten in diesem Kontext nicht sinnvoll ist.

Eine Anwendung findet dieses Verfahren allerdings nur auf Roboter-Plattformen, die sich omnidirektional bewegen können und deren Bewegungsrichtung dadurch unabhängig von der Ausrichtung des Roboters ist. Differential angetriebene oder durch Antriebs- und Lenkachse gesteuerte Roboter müssen stets entsprechend der aktuellen Bewegungsrichtung rotiert sein.

Mittels der Überlagerung von Freiheitsgraden konnte ein Verhalten implementiert werden, welches einen Roboter unter der Vermeidung von Hindernissen zum Ball laufen lässt, währenddessen jedoch stets, unabhängig von der aktuellen Bewegungsrichtung, der Körper in Richtung des Ball ausgerichtet ist und dieser somit im Blickfeld bleibt. Die Möglichkeit einer hinreichenden Ballkontrolle vorausgesetzt, könnte beispielsweise ein weiteres Verhalten modelliert werden, welches den Roboter mit dem Ball zu einem Ziel dribbeln lässt und gleichzeitig den eigenen Körper stets zwischen den Ball und eventuelle Gegenspieler rotiert.

4.4.3 Glättung der Bewegungsvektoren

Die zuvor schon beschriebenen Auswirkungen von möglichen Schwankungen im Weltmodell können auch durch eine Glättung der Bewegungsvektoren teilweise aufgefangen werden. Dazu wird die Bewegungssteuerung mit einer gewissen Trägheit

versehen, die, abhängig von der Zeit, nur eine fest vorgegebene Abweichung eines Bewegungsvektors $\vec{v}_n$ von seinem Vorgänger $\vec{v}_{n-1}$ zulässt.

Sei t die Zeit in Sekunden, die zwischen der Erzeugung der beiden Vektoren vergangen ist, $|\vec{v}_n|$ und $|\vec{v}_{n-1}|$ die Längen der Vektoren sowie α_n und α_{n-1} deren jeweilige Ausrichtung im verwendeten Koordinatensystem. Dann lassen sich Abweichung der Länge δ_l und des Winkels δ_α folgendermaßen definieren:

$$\delta_l = \frac{|\vec{v}_n| - |\vec{v}_{n-1}|}{t} \tag{4.16}$$

$$\delta_\alpha = \frac{\alpha_n - \alpha_{n-1}}{t} \tag{4.17}$$

Unter der Vorgabe einer maximalen Längenabweichung $\delta_{l_{max}}$ pro Sekunde gilt für den Vektor $\vec{v}_n$:

$$\vec{v}_n = \begin{cases} \vec{v}_n \frac{|\vec{v}_{n-1}|+t\cdot\delta_{l_{max}}}{|\vec{v}_n|} & , \delta_l > \delta_{l_{max}} \\ \vec{v}_n \frac{|\vec{v}_{n-1}|-t\cdot\delta_{l_{max}}}{|\vec{v}_n|} & , \delta_l < -\delta_{l_{max}} \\ \vec{v}_n & , sonst \end{cases} \tag{4.18}$$

Ebenso kann eine maximale Winkelabweichung von $\delta_{\alpha_{max}}$ pro Sekunde realisiert werden; dabei sei $\otimes$ ein Rotationsoperator, der einen links stehenden Vektor um den rechts stehenden Winkel dreht:

$$\vec{v}_n = \begin{cases} \vec{v}_n \otimes (t \cdot \delta_{\alpha_{max}} - \delta_\alpha) & , \delta_\alpha > \delta_{\alpha_{max}} \\ \vec{v}_n \otimes (-t \cdot \delta_{\alpha_{max}} - \delta_\alpha) & , \delta_\alpha < -\delta_{\alpha_{max}} \\ \vec{v}_n & , sonst \end{cases} \tag{4.19}$$

Kapitel 5

Aktionsauswahl

Neben der Bewegungsplanung ist die Auswahl von Aktionen die zweite Anwendung für Potentialfelder. Dieses Kapitel beschreibt die möglichen Arten von Aktionen, deren Bewertung und Auswahl. Ebenso werden Verfahren zum Umgang mit Aktionssequenzen vorgestellt.

5.1 Aktionen und deren Bewertung

Die Auswahl von Aktionen basiert auf einem anderen Berechnungsschema als die Bewegungssteuerung. Daher wird zur Beschreibung eines Verhaltens zur Aktionsauswahl ein anderer Feldtyp benötigt. Dieser wird im Folgenden als *Aktionsfeld* bezeichnet. Einem solchen Feld sind, zusätzlich zu Objekten und dem zu verwendenden Bewertungsverfahren, Beschreibungen möglicher Aktionen zugeordnet.

5.1.1 Aktionen

Bevor die Verfahren zur Modellierung und Bewertung erläutert werden, bedarf es zunächst einer kurzen Beschreibung dessen, was im Rahmen dieser Arbeit als eine *Aktion* betrachtet wird.

Definition einer Aktion

Eine Aktion wird als eine aus der Sicht der Verhaltenssteuerung nicht mehr unterteilbare Handlung betrachtet. Eine solche Aktion kann beliebig komplex in ihrer späteren Ausführung sein, dies obliegt jedoch nicht mehr der Verhaltenssteuerung,

da diese Aktionen lediglich bewertet und auswählt. Beispiele für Aktionen sind *Schieße den Ball, Drücke den Schalter* oder auch *Bleibe stehen*.

Aktionstypen

Die möglichen Aktionen sind in zwei Klassen unterteilt: *Transformationen* und *Messungen*.

Eine Transformation beschreibt eine Manipulation der Umwelt in Form einer Veränderung der Position eines der Objekte der Umgebung oder des Roboters selbst.

Bei der Durchführung einer Messung bleibt die Umgebung unverändert, es wird lediglich an einer bestimmten Position eine Bewertung vorgenommen. Ursprünglich war für diese Art von Aktionen ein weiterer separater Feldtyp vorgesehen, aufgrund der großen Ähnlichkeit der Implementierung und der Tatsache, dass auch eine Messung einzig dem Zweck dient, eine Entscheidung für oder gegen ein bestimmtes Verhalten zu treffen, sind Messungen in die Aktionsauswahl integriert worden.

5.1.2 Grundprinzip der Aktionsbewertung

Die Bewertung von Aktionen durch Potentialfelder wird, im Unterschied zur Bewegungssteuerung, in vergleichsweise wenigen Veröffentlichungen behandelt. Zu Bewertungszwecken wird dabei im Allgemeinen kein Potentialfeld verwendet, sondern die Werte der zu einem solchen Feld gehörenden Potentialfunktionen.

Häufig wird, wie in [Meyer and Adolph, 2003] oder [Ball and Wyeth, 2004], die gesamte Umgebung durch ein Raster in Zellen eingeteilt und jede dieser Zellen mittels mehrerer, bestimmten Objekten zugeordneten, Potentialfunktionen bewertet. Nach der Bestimmung der am besten bewerteten Region kann eine passende Aktion, wie zum Beispiel das Schießen eines Balls an die entsprechende Stelle, ausgewählt werden. Dabei ist jedoch noch nicht gesichert, ob die Aktion überhaupt vollständig ausgeführt werden kann. Ein weiterer Nachteil ist die vorgenommene Rasterung, die zum einen das bereits in Abschnitt 4.3.2 beschriebene Problem der zu verwendenden Zellgrößen mit sich bringt, und zum anderen für eine große Anzahl von Positionen Bewertungen durchführt, von denen letztendlich die meisten nicht verwendet werden.

Einen dem entgegengesetzten, intuitiveren Ansatz stellen [Johannson and Saffiotti, 2002] vor: Ausgehend von den überhaupt möglichen Aktionen eines Roboters werden deren Konsequenzen berechnet und anschließend bewertet. Ist beispielsweise die Bahn eines Balls nach einem geraden Schuss bekannt, so kann, auch unter Beachtung etwaiger Kollisionen mit anderen Objekten, der Endzustand der Aktion

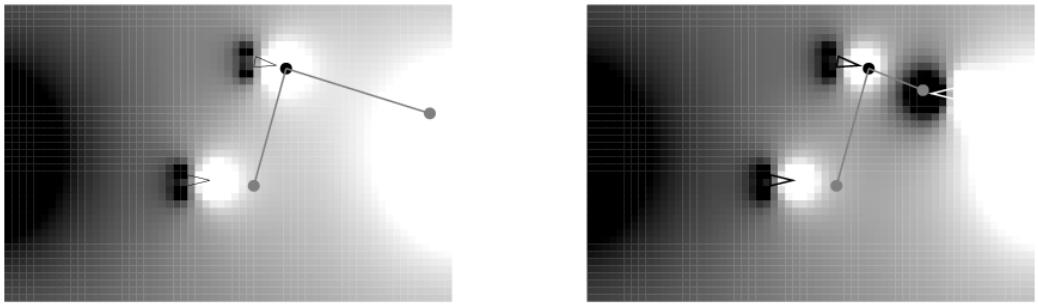


Abbildung 5.1: *Zwei Beispiele der Bewertung von Aktionen aus [Johannson and Saffiotti, 2002]. Die helleren Regionen repräsentieren eine positive Bewertung, die dunklen Regionen eine negative Bewertung. Evaluiert werden jeweils ein gerader sowie ein seitlicher Schuss eines Balls auf einem Spielfeld in der Sony-Liga.*

bestimmt werden. Eine Bewertung muss nur noch für die Position des Balls nach dem Schuss durchgeführt werden. Abbildung 5.1 zeigt ein entsprechendes Beispiel.

Aufgrund der höheren Effizienz und der Möglichkeit der direkten Abbildung von Aktionen wurde dieser Ansatz zur Aktionsbewertung ausgewählt.

5.1.3 Bestimmung des Potentials an einer Position

Die Bewertung einer Position P wird definiert als das Gesamtpotential $\varphi(P)$ des Aktionsfeldes an P. Dieses kann, analog zur Bestimmung der Potentialfelds, wie in Abschnitt 4.1.1 beschrieben, durch die Berechnung und Aufsummierung der Potentialfunktionen $\varphi_i(P)$ aller, dem Verhalten zugeordneten und aktivierten, Objekte $O_i \in \{O_1, .., O_n\}$ bestimmt werden:

$$\varphi(P) = \sum_{i=1}^{n} \varphi_i(P) \tag{5.1}$$

Dies entspricht der Überlagerung der Potentialfunktionen der Objekte und kann auch, nach [Johannson and Saffiotti, 2002], als die vorhandene *elektrische Ladung* an der Position P bezeichnet werden.

5.1.4 Bewertungsverfahren

Es wird zunächst davon ausgegangen, dass einem Aktionsfeld genau eine Aktion A zugeordnet ist[1], die mittels einer Funktion $\phi(A)$ bewertet werden kann. Für A gilt:

$$A \in \{M_O, M_R, T_O, T_R\} \tag{5.2}$$

Dabei sind M_O und M_R Messungen an der Position eines Objekts beziehungsweise an der Position des Roboters; T_O und T_R repräsentieren die Transformation eines Objekts beziehungsweise des Roboters selbst. Die Bewertung der einzelnen Aktionen wird im Folgenden beschrieben:

Bewertung von Messungen

Die Durchführung einer Messung an einer Position im Aktionsfeld entspricht der Bestimmung des Potentials an dieser Position. Die Bewertung $\phi(M_R)$ an der aktuellen Roboterposition R ist folgendermaßen definiert:

$$\phi(M_R) = \varphi(R) \tag{5.3}$$

Ebenso können Messungen an den Positionen beliebiger Objekte durchgeführt werden, was dem Konzept einer *Sonde* aus [Johannson and Saffiotti, 2002] entspricht. Die Bewertung ist unabhängig davon, ob es sich um Objekte handelt, die mittels eines zugeordneten Feldes auf ihre Umgebung wirken, oder um rein virtuelle passive Objekte, die nur zum Zweck der Messung modelliert worden sind. Aufgrund dieser Unabhängigkeit wird zur Bestimmung von $\phi(M_O)$ eine Funktion $\varphi_O(P)$ verwendet, die das Gesamtpotential aller Objekte bildet, wie in Abschnitt 5.1.3 beschrieben, dabei jedoch das Potential des Objekts O auslässt.

Somit kann eine Messung M_O folgendermaßen bewertet werden:

$$\phi(M_O) = \varphi_O(O) \tag{5.4}$$

Bewertung von Transformationen

Die Bewertung einer Transformation entspricht im Wesentlichen der Bewertung einer Messung, mit dem Unterschied, dass nicht die Position O eines Objekts bewertet wird, sondern die Position $T(O)$ nach einer durchgeführten Transformation T. Die möglichen Transformationen werden in Abschnitt 5.2 beschrieben.

[1] Möglichkeiten zum Vergleich mehrerer Aktionen innerhalb eines Feldes werden in Abschnitt 5.3 beschrieben.

Die Bewertung $\phi(T_O)$ einer beliebigen Transformation T eines Objektes O, wie beispielsweise eines seitlichen Schusses eines Balls, kann daher folgendermaßen bestimmt werden:

$$\phi(T_O) = \varphi_O(T(O)) \tag{5.5}$$

Ursprünglich nur für die Transformation anderer Objekte entworfen, lässt sich dieses Modell auch auf eine Transformation der Roboterposition R anwenden:

$$\phi(T_R) = \varphi(T(R)) \tag{5.6}$$

Anders als bei Messungen, lässt sich bei Transformationen nicht nur ein absoluter Wert eines Zustands bestimmen, sondern auch, unter Einbeziehung einer Bewertung des Ausgangszustandes vor der Transformation, die bewirkte Zustandsveränderung $\phi_G(A)$:

$$\phi_G(T_O) = \varphi_O(T(O)) - \varphi_O(O) \tag{5.7}$$
$$\phi_G(T_R) = \varphi(T(R)) - \varphi(R) \tag{5.8}$$

Abhängigkeit von der Ausführungszeit

Zusätzlich kann einer Aktion eine Ausführungszeit t_A (mit $t_A \geq 1$) zugeordnet werden. Durch eine Teilung von $\phi(A)$ durch t_A kann eine Bewertung in Abhängigkeit von der benötigten Ausführungszeit erfolgen, so dass bei einem Vergleich verschiedener Aktionen nicht allein das Resultat sondern die zeitliche Effizienz entscheidend ist.

5.1.5 Bestimmung eines Aktivierungswertes

Da ein Aktionsfeld einem Verhalten, gemäß der in Abschnitt 2.2.2 beschriebenen Verhaltensarchitektur, entspricht, muss ein Aktivierungswert a berechnet werden können. Dazu sind mehrere, auf den im vorherigen Abschnitt beschriebenen Bewertungsverfahren basierende, Möglichkeiten implementiert worden:

Wert von ϕ (A)

Die Bewertung $\phi(A)$ einer Aktion kann direkt als Aktivierungswert verwendet werden, so dass verschiedene Aktionsfelder anhand der Bewertung ihrer jeweiligen Aktionen miteinander verglichen werden können.

Bewertung der Zustandsveränderung ϕ_G (A)

Die Bewertung $\phi_G(A)$ einer Zustandsveränderung kann lediglich für Aktionsfelder, denen eine Transformation zugeordnet worden ist, sinnvoll verwendet werden. Da Messungen keine Zustandsveränderung bewirken, hätte der entsprechende Aktivierungswert a immer den Wert Null. Werden in einem Verhalten, parallel zur Aktionsauswahl, Bewegungsfelder verwendet, ist diese Art der Bewertung, vorausgesetzt die Aktion ist eine Transformation, zumeist am geeignetsten, da das Vorzeichen von $\phi_G(A)$ stets ein Indikator für eine Zustandsverbesserung (negatives Vorzeichen) oder -verschlechterung (positives Vorzeichen) ist.

Entfernungsabhängiger Wert von ϕ_G (A)

Dieses Verfahren kann, ebenso wie das im vorherigen Abschnitt beschriebene, lediglich im Zusammenhang mit Transformationen verwendet werden. Dabei wird der Wert $\phi_G(A)$ zusätzlich in Relation zu einer durch eine Transformation zurückgelegte Distanz gesetzt. Sei P die Position eines Objektes oder des Roboters vor und P_T die Position nach der Transformation, dann kann ein Aktivierungswert a folgendermaßen bestimmt werden:

$$a = \begin{cases} \frac{1}{|\vec{P_T P}|} \cdot \phi_G(A) & , |\vec{P_T P}| \neq 0 \\ 0 & , sonst \end{cases} \tag{5.9}$$

Dies führt zu einer schlechteren Bewertung von Transformationen über eine große Distanz.

Zuweisung eines konstanten Aktivierungswertes

Die Zuweisung eines konstanten Aktivierungswertes ist, ebenso wie bei einem Bewegungsverhalten, möglich, siehe hierzu Abschnitt 4.1.4.

5.2 Transformationen

Um die tatsächlich möglichen Aktionen eines Roboters abbilden zu können, ist eine Reihe verschiedener Transformationen implementiert worden. Deren Planung ist fester Bestandteil der Verhaltensarchitektur und wird nicht, wie beispielsweise bei [Johannson and Saffiotti, 2002], durch einen externen Planer durchgeführt. Für eine Transformation T gilt:

$$T \in \{T_F, T_O, T_G, R_F, R_O, R_G\} \tag{5.10}$$

Die einzelnen Elemente dieser Menge werden im Folgenden vorgestellt. Sie teilen sich auf in die Bereiche *Translation* und *Rotation*; theoretisch ist das verwendete Modell um beliebige Transformationen erweiterbar.

5.2.1 Translation

Sowohl die Position des Roboters als auch beliebiger anderer Objekte kann mittels einer Translation im Raum verschoben werden. Da beide Fälle in gleicher Form berechnet werden können, schließt die Bezeichnung *Objekt* in den folgenden drei Abschnitten stets den Roboter mit ein.

Fest vorgegebene Translation

Ist die Auswirkung einer Translation auf ein Objekt mit dem Ortsvektor $\vec{O}$ bereits im Vorfeld bekannt, wie beispielsweise die Position des Balls nach einem bestimmten Schuss, kann bei der Modellierung ein Vektor $\vec{t_f}$ angegeben werden, der diese Verschiebung im relativen Koordinatensystem des Roboters beschreibt. Für eine solche Translation T_F gilt somit:

$$T_F\left(\vec{O}\right) = \vec{O} + \vec{t_f} \tag{5.11}$$

Translation zu einem anderen Objekt

Mittels der Translation T_O zu einem anderen Objekt lassen sich komplexere Verschiebungen beschreiben, deren Parameter zum Zeitpunkt der Modellierung nicht bekannt sind, wie zum Beispiel der Pass eines Balls zu einem Mitspieler oder ein Schuss auf ein Tor. Die Verwendung einer solchen Transformation eignet sich natürlich nur für Systeme, deren Ausführungsschicht derartige Aktionen durchführen kann.

Sind der Ortsvektor $\vec{O}$ des zu verschiebenden Objekts sowie des Zielobjekts $\vec{P}$ bekannt, kann der Translationsvektor $\vec{t_o}$ und somit auch T_O bestimmt werden:

$$\vec{t_o} = \vec{P} - \vec{O} \tag{5.12}$$

$$T_O\left(\vec{O}\right) = \vec{O} + \vec{t_o} \tag{5.13}$$

Der Vektor $\vec{t_o}$ wird, sollte das Aktionsfeld zur Ausführung ausgewählt werden, von der Verhaltenssteuerung als Teil des Ergebnisses zurückgeliefert.

function FOLLOWGRADIENT (*maxDist*, *maxDeviation*, *step*, $\vec{start}$) **returns** $\vec{vector}$
$\vec{startDir} \leftarrow$ COMPUTEGRADIENTAT($\vec{start}$)
$\vec{stepVec} \leftarrow \vec{startDir}$
$\vec{stepVec} \leftarrow \frac{step}{|\vec{stepVec}|} \cdot \vec{stepVec}$
$\vec{pos} \leftarrow \vec{start} + \vec{stepVec}$
dist ← *step*
while (DEVIATIONBETWEEN ($\vec{stepVec}$, $\vec{startDir}$) < *maxDeviation*)
and (*dist* < *maxDist*)
do
$\vec{stepVec} \leftarrow$ COMPUTEGRADIENTAT($\vec{pos}$)
$\vec{stepVec} \leftarrow \frac{step}{|\vec{stepVec}|} \cdot \vec{stepVec}$
$\vec{pos} \leftarrow \vec{pos} + \vec{stepVec}$
dist ← *dist* + *step*
end
return ($\vec{pos} - \vec{stepVec} - \vec{start}$)

Abbildung 5.2: *Der Algorithmus* FOLLOWGRADIENT *verfolgt den Gradienten des Potentialfelds bis eine zuvor festgelegte Abweichung vom Ausgangsgradienten beziehungsweise eine Maximallänge erreicht worden ist.*

Da zur Bewertung von Aktionen deren Ausführungsdauer verwendet werden kann, muss bei der Modellierung zusätzlich die Geschwindigkeit[2] s der Translation angegeben werden, so dass die Zeit t_A berechnet werden kann:

$$t_A = \frac{|\vec{t_o}|}{s} \tag{5.14}$$

Translation entlang des Gradienten

Eine weitere komplexere Transformation, die das Potentialfeld der Umgebung nutzt, ist die Verfolgung des Gradienten ausgehend von der Position des Objekts. Ähnlich dem in Abschnitt 4.3.4 beschriebenen Verfahren zum Erkennen des Verlassens eines lokalen Minimums, wird eine Folge von Vektoren gleicher Länge entlang des jeweiligen Gradienten gebildet, bis eine maximale Entfernung überschritten, oder eine Position erreicht ist, an der der Gradient um einen zuvor festgelegten Maximal-

[2] Da die Verhaltensarchitektur keine festgelegte Längeneinheit verwendet und der Ausführungsdauer von Aktionen ebenfalls keine Zeiteinheit zugeordnet ist, gilt für s allgemein $\frac{Entfernung}{Zeit}$.

wert vom Ausgangsgradienten abweicht, was beispielsweise in der Nähe repulsiver Hindernisse der Fall ist.

Mittels des in Abbildung 5.2 beschriebenen Algorithmus lässt sich eine solche Translation $\vec{t_g}$ bestimmen, so dass T_G folgendermaßen definiert werden kann:

$$T_G(\vec{O}) = \vec{O} + \vec{t_g} \tag{5.15}$$

Für die Rückgabe von $\vec{t_g}$ sowie die Bestimmung der benötigten Zeit t_A gelten ebenfalls die im vorherigen Abschnitt beschriebenen Bedingungen.

5.2.2 Rotation

Eine Rotation beschreibt eine Drehung eines Objekts um den Roboter oder die Drehung des Roboters selbst.

Fest vorgegebene Rotation

Ebenso wie eine fest vorgegebene Translation, kann bei der Modellierung eine Rotation R_F mittels eines festen Winkels α_f beschrieben werden.

Zur Rotation des Roboters selbst muss dann lediglich α_f zu dem Rotationswinkel der Roboterpose addiert werden und das Ergebnis anschließend auf den intern für sämtliche Winkel verwendeten Wertebereich $[-\pi, \ldots, \pi]$ normiert werden.

Ebenso kann die Ausrichtung eines um den Roboter rotierten Objektes bestimmt werden. Eine solche Rotation führt allerdings zusätzlich zu einer Veränderung der Position des Objekts, die, unter Verwendung des in Abschnitt 4.4.3 eingeführten Rotationsoperators $\otimes$ und der Position des Roboters R, folgendermaßen bestimmt werden kann:

$$R_F(\vec{O}) = ((\vec{O} - \vec{R}) \otimes \alpha_f) + \vec{R} \tag{5.16}$$

Rotation zu einem anderen Objekt

Die Rotation R_O zu einem anderen Objekt entspricht der Transformation T_O zu einem anderen Objekt und folgt dem gleichen Berechnungsschema wie R_F. Es verbleibt die Bestimmung des Rotationswinkels α_o. Dieser kann mittels des Vektors $\vec{t_o}$ zu dem Objekt sowie der Eigenrotation α_R des Roboters bestimmt werden:

$$\vec{r} = \vec{t_o} \otimes (-\alpha_R) \tag{5.17}$$

$$\alpha_o = \text{atan2}(\vec{r}_y, \vec{r}_x) \tag{5.18}$$

Der Wert von α_o wird von der Verhaltenssteuerung als ein Teil des Ergebnisses zurückgeliefert. Die benötigte Zeit t_A der Rotation wird, entsprechend einer bei der Modellierung vorgegebenen Geschwindigkeit, automatisch bestimmt.

Rotation in Richtung des Gradienten

Zur Rotation R_G in Richtung des Gradienten, beispielsweise um den Roboter und ein Objekt entlang des Potentialfelds auszurichten, wird an der Position R des Roboters der Feldvektor $\vec{v}(R)$ gebildet. Aus dessen Winkel im absoluten Koordinatensystem und der Ausrichtung des Roboters lässt sich der Rotationswinkel α_g bestimmen:

$$\alpha_v = \text{atan2}\,(\vec{v}_y, \vec{v}_x) \tag{5.19}$$

$$\alpha_g = \alpha_v - \alpha_R \tag{5.20}$$

Mittels dieses Winkels kann eine Rotation durchgeführt werden. Die Art der Rückgabe von α_g sowie die Bestimmung von t_A unterscheiden sich nicht von R_O .

5.2.3 Begleiten der Transformation

Bei der Transformation anderer Objekte lassen sich zwei Fälle unterscheiden. Zum einen ist es möglich, dass der Roboter das Objekt bewegt und dabei seine eigene Position nicht verändert, wie beispielsweise beim Schuss eines Balls. Zum anderen existieren Aktionen, bei denen der Roboter dem transformierten Objekt folgen soll, wie es beim Verschieben von Objekten oder beim Dribbeln mit einem Ball der Fall ist.

Bei der Modellierung des Verhaltens ist es möglich, für jede Transformation eines Objekts anzugeben, ob eine entsprechende Transformation der Roboterposition ebenfalls durchgeführt werden soll. Dies ist insbesondere für die in Abschnitt 5.3 beschriebenen Aktionssequenzen unter dem Gesichtspunkt der Durchführbarkeit einer Aktion (siehe Abschnitt 5.2.5) wichtig.

5.2.4 Wahrscheinlichkeitsverteilung einer Transformation

Die Durchführung einer Transformation in der realen Welt entspricht nicht immer exakt den gewünschten Vorgaben. Neben kleineren Abweichungen, wie zum Beispiel das Abdriften eines geschossenen Balls, können Aktionen auch in einer bestimmten Anzahl von Fällen vollständig misslingen, wie die Ausführung eines Schusses, der den Ball verfehlt.

Die Transformation $T(O)$ eines Objekts O lässt sich jedoch mittels der Durchführung einer Messreihe auf eine Menge $\mathbb{T} = \{T_1(O), \ldots, T_n(O)\}$ von Transformationen abbilden, die jeweils einen möglichen Ausgang der Aktion repräsentieren. Dabei wird zusätzlich zu jedem $T_i \in \mathbb{T}$ eine Wahrscheinlichkeit p_i bestimmt. Das Ergebnis kann direkt auf die Modellierung einer Aktion übertragen werden, wobei allerdings zusätzlich $T_i \in \{T_F, R_F\}\ \forall\, T_i \in \mathbb{T}$ gelten muss, da die jeweiligen Transformationsparameter in einem vorhergehenden Versuch und nicht zur Laufzeit bestimmt werden.

Zur Bewertung einer solchen Aktion wird das zu transformierende Objekt auf ein, in Abschnitt 3.5 beschriebenes, Objekt mit einer Wahrscheinlichkeitsverteilung abgebildet. Jedes Ergebnis einer Transformation T_i zusammen mit einer Wahrscheinlichkeit p_i entspricht somit einer Hypothese über den Zustand des Objekts.

5.2.5 Durchführbarkeit der Transformation von Objekten

Eine notwendige Bedingung für die spätere Bewertung einer Aktion ist ihre Durchführbarkeit. Ist es gar nicht erst möglich, eine Aktion auszuführen, wird das dazugehörige Verhalten nicht bewertet und entsprechend als nicht durchführbar gekennzeichnet.

Während die Transformation des Roboters sowie eine Messung an dessen Position generell durchführbare Aktionen sind, sind Aktionen im Bezug auf andere Objekte stets von deren Aktivierung abhängig. Des Weiteren ist die Transformation solcher Objekte zusätzlich abhängig von deren relativer Position zum Roboter. So kann beispielsweise ein zwei Meter entfernt liegender Ball niemals von einem Sony-Roboter bewegt werden.

Um die Durchführbarkeit einer solchen Transformation zu testen, müssen die Bereiche relativ zum Roboter, in denen die jeweilige Transformation möglich ist, explizit definiert werden. Dies geschieht über die Angabe eines oder mehrerer konvexer Polygone. Die Ausführbarkeit einer Aktion entspricht also dem Enthaltensein der Objektposition in einem Polygon. Der hierzu verwendete Algorithmus stammt aus [Sedgewick, 1992].

Wie im Abschnitt 3.5 beschrieben, kann sowohl die Position eines Objekts als auch die des Roboters über eine Menge von Hypothesen beschrieben sein. Dies wirkt sich insofern auf die Transformation von Objekten aus, als dass der Durchführbarkeitstest für sämtliche Kombinationen von möglichen Objekt- und Roboterpositionen durchgeführt werden muss und eine Transformation nur für die jeweils gültigen Paare von Hypothesen ausgeführt wird.

5.2.6 Kollisionserkennung

Bei der Transformation eines Objekts oder des Roboters müssen auch die weiteren Objekte der Umgebung berücksichtigt werden. Ohne eine Kollisionserkennung würden aus der Sicht der Verhaltenssteuerung Schüsse während eines Fußballspiels andere Spieler durchdringen können, beziehungsweise würde ein Schuss aus kurzer Distanz auf ein Tor nicht an der Rückwand des Tores stoppen sondern einfach durch das Tor rollen. Da die Bewertung einer Transformation in einem direkten Zusammenhang mit der Position eines Objekts nach der Aktion steht, wäre ein solches Modell nicht hinreichend.

Zum Zweck der Kollisionserkennung wurde daher ein einfaches, schnell zu berechnendes Verfahren implementiert. Dabei wird ein zu transformierendes Objekt stets auf einen Punkt reduziert, seine geometrische Form also nicht beachtet. Ist dieses Vorgehen für einen bestimmten Anwendungszweck zu ungenau, so können beispielsweise während der Modellierung die Formen der anderen Objekte an den maximalen Radius des zu transformierenden Objekts angepasst werden. Eine Kollision kann nun festgestellt werden, indem der geplante Weg eines Objekts durch eine geometrische Figur beschrieben wird, die anschließend auf Schnittpunkte mit anderen Objekten untersucht werden kann. Dieses Vorgehen bezieht keine möglichen Veränderungen der Umwelt während der Ausführung der Transformation ein. Sind allerdings Geschwindigkeiten und ein vorhersehbares Verhalten der Objekte bekannt – während eines Fußballspiels kann beispielsweise davon ausgegangen werden, dass die Gegenspieler versuchen werden, einen geschossenen Ball abzufangen – so können entsprechend die Größen der geometrischen Figuren dieser Objekte derart modelliert werden, dass sie nicht mehr eine Form sondern die möglicherweise bedeckte Fläche repräsentieren. Dies entspräche einer Anpassung des Konfigurationsraums nach [Latombe, 1991].

Für die Translation und die Rotation sind verschiedene Verfahren zur Kollisionserkennung implementiert worden:

Translation

Eine Translation kann auf eine Strecke abgebildet werden, die die Position O des zu transformierenden Objekts mit der geplanten Endposition $T(O)$ verbindet. Durch die Berechnung sämtlicher Schnittpunkte mit den Figuren der Objekte der Umgebung kann der zu O nächstgelegene Kollisionspunkt $T_{max}(O)$ bestimmt werden, wie in Abbildung 5.3 zu sehen ist. Existiert kein Schnittpunkt, so gilt $T_{max}(O) = T(O)$. Für die spätere Bewertung der Translation wird stets $T_{max}(O)$ verwendet.

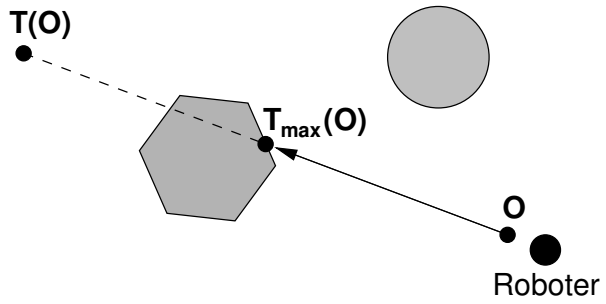


Abbildung 5.3: *Zum Zweck der Kollisionserkennung wird eine Translation auf eine Strecke abgebildet und die längstmögliche Translation $T_{max}(O)$ bestimmt.*

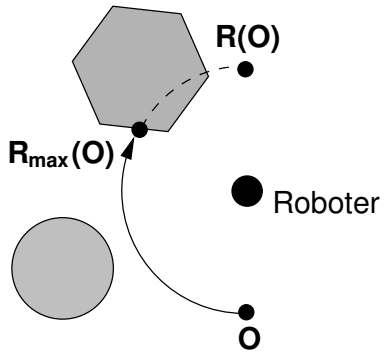


Abbildung 5.4: *Eine Rotation kann auf einen Kreisausschnitt abgebildet und somit die maximal mögliche Endposition $R_{max}(O)$ bestimmt werden.*

Rotation

Die Rotation eines Objekts um den Roboter kann auf einen Kreisausschnitt abgebildet werden, der durch die Objektposition O sowie die Endposition der geplanten Rotation $R\,(O)$ begrenzt wird, siehe Abbildung 5.4. Analog zum Kollisionstest der Translation kann eine Position $R_{max}(O)$ berechnet werden, die die maximal mögliche Rotation repräsentiert und die Basis für die spätere Bewertung der Aktion bildet.

Optimierung der Kollisionserkennung

Zur Bestimmung der für die zuvor beschriebenen Verfahren benötigten Schnittpunkte müssen die erzeugten Figuren mit sämtlichen Objekten der Umgebung auf mögliche Schnittpunkte getestet werden. Dies kann, insbesondere bei einer großen Anzahl von Objekten und einer Vielzahl zu berechnender Transformationen, zu einem erheblichen Rechenaufwand führen.

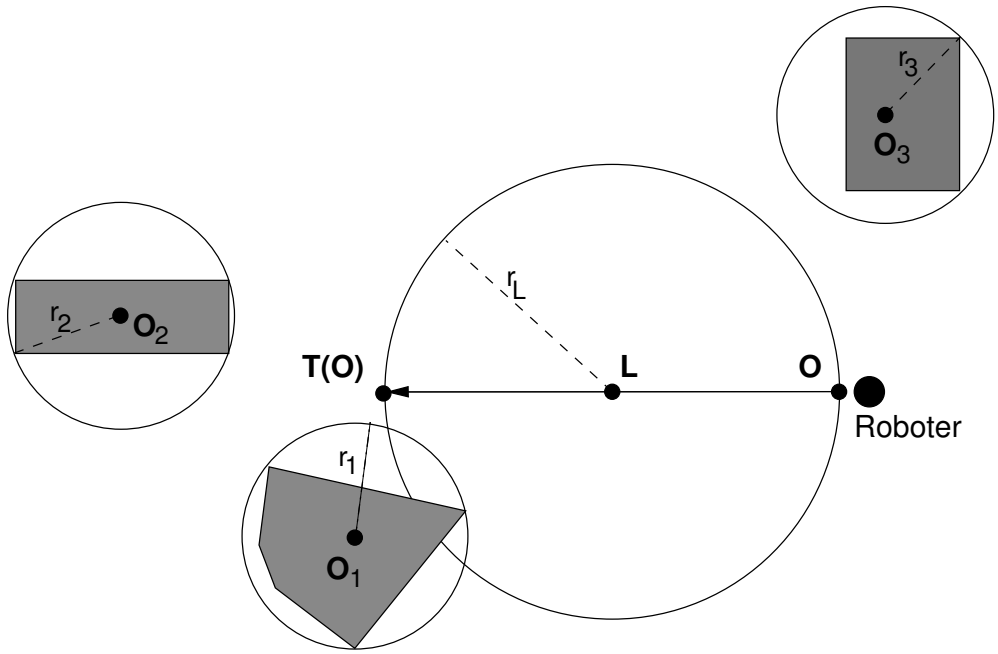


Abbildung 5.5: *Eine Strecke, welche eine Translation $T(O)$ beschreibt, wird von einem Kreis um deren Mittelpunkt L umgeben. Die Kreise um die Objekte der Umgebung werden auf einen Schnitt mit diesem Kreis getestet bevor weitere Verfahren zur Bestimmung von Schnittpunkten angewendet werden.*

Zur Reduzierung dieses Aufwands wird zu der internen Repräsentation eines jeden geometrischen Objekts eine Entfernung r hinzugefügt, die, ausgehend von dessen Position O, dem Radius einen Kreises um die gesamte Figur entspricht. Bevor nun zwei beliebig komplexe Objekte an den Positionen O_1 und O_2 auf gemeinsame Schnittpunkte untersucht werden, kann mittels einer einfachen Bedingung $mayIntersect$ anhand deren Radien r_1 und r_2 geprüft werden, ob überhaupt Schnittpunkte möglich sind:

$$mayIntersect\,(O_1, O_2) = (|\vec{O_1 O_2}| < r_1 + r_2) \tag{5.21}$$

Abbildung 5.5 zeigt dies am Beispiel einer Translation.

5.3 Aktionssequenzen

Die Bewertung von Aktionen aufgrund der zuvor bestimmten Endzustände ermöglicht bereits ein gewisses vorausschauendes Handeln. In einigen Fällen kann es jedoch nützlicher sein, eine zunächst suboptimal erscheinende Aktion auszuwählen,

da nach dieser eine weitere Aktion ausgeführt werden kann, die ein letztendlich besseres Ergebnis erzielt. Beispielsweise kann es beim Fußball nützlicher sein, sich mit dem Ball ein Stück zu drehen und dann zu schießen, als sofort zu schießen, obwohl der Schuss zunächst besser bewertet wird als die Drehung. Hierzu ist es nötig, Sequenzen von Aktionen repräsentieren und bewerten zu können. Dies ist bereits von [Johannson and Saffiotti, 2002] angedacht aber nicht realisiert worden.

Basierend auf der in diesem Kapitel beschriebenen Modellierung sind zwei Möglichkeiten zur Realisierung solcher Sequenzen implementiert worden: Die Beschreibung fester Aktionssequenzen sowie die Suche nach der bestmöglichen Aktionsfolge.

5.3.1 Feste Aktionssequenzen

Sollen neben einzelnen Aktionen auch feste Sequenzen bewertet werden, wie beispielsweise eine Rotation zum Tor mit einem anschließenden geraden Schuss des Balls, können Aktionsfelder modelliert werden, denen statt einer Aktion A eine Sequenz $A_1, A_2, \ldots, A_n$ zugeordnet ist. Dies können theoretisch beliebige Aktionen sein, die sich zudem auf verschiedene Objekte beziehen. Der wahrscheinlichste Anwendungsfall ist jedoch die Beschreibung einer Reihe von Transformationen desselben Objekts.

Bevor eine Bewertung vorgenommen werden kann, müssen zunächst die Auswirkungen sämtlicher Aktionen auf das Modell der Umwelt bestimmt werden, indem die Aktionen der Reihe nach ausgewertet werden. Ist dabei eine Aktion A_{m+1} (mit $0 \leq m \leq n$) gemäß den Definitionen aus Abschnitt 5.2.5 nicht durchführbar oder nicht mehr vorhanden, endet die Auswertung und A_m gilt als die letzte ausführbare Aktion. Für den Fall $m = 0$ wird das gesamte Verhalten als nicht ausführbar deklariert.

Die anschließende Bewertung ist definiert als der Wert von $\phi(A_m)$, da dieser letztendlich das durch die Ausführung der Sequenz erreichte Ergebnis repräsentiert. Soll ein Wert $\phi_G(A_m)$ gebildet werden, so geschieht dies unter Bezugnahme auf den Zustand des A_m zugeordneten Objekts vor der Auswertung der Sequenz.

Das an die Verhaltenssteuerung zurückgelieferte Ergebnis enthält, sofern vorhanden, die Parameter der Aktion A_1, da dies stets die nächste auszuführende Aktion ist.

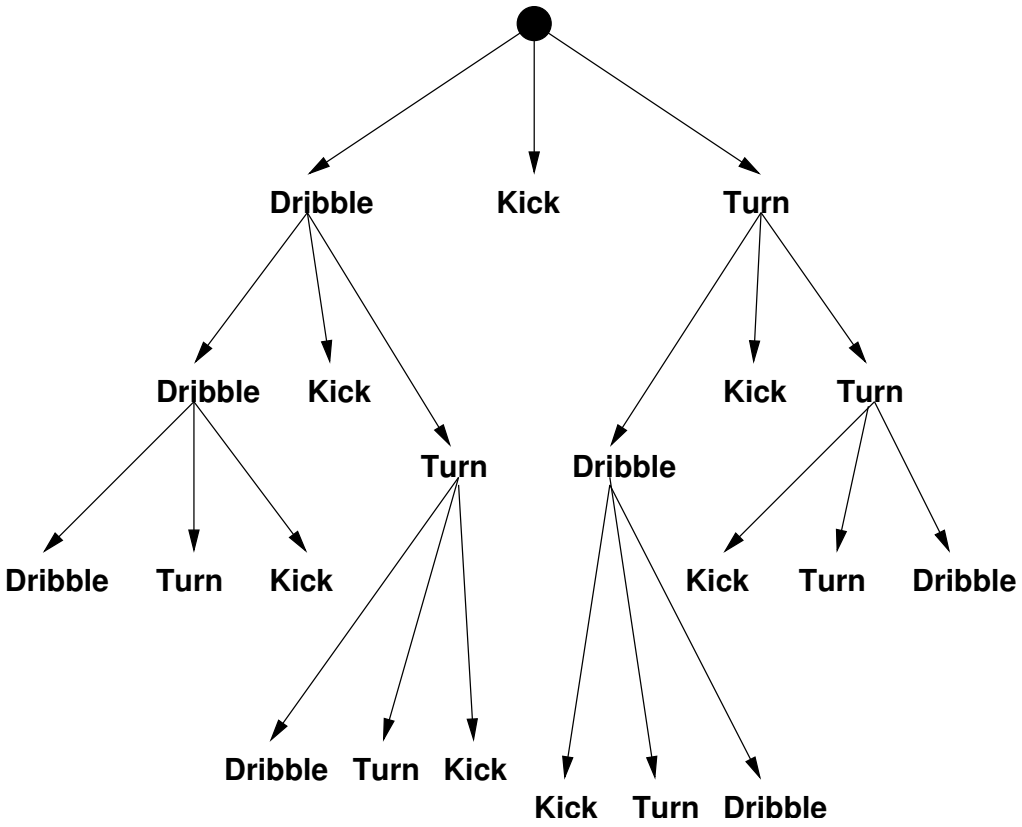


Abbildung 5.6: *Automatische Bildung möglicher Aktionsfolgen. Jeder Pfad von der Wurzel des Baums zu einem der Knoten entspricht einer möglichen Sequenz. Die drei Aktionen Kick, Dribble und Turn beziehen sich auf die Transformation eines Balls in einem Fußballspiel. Nach einer Auswertung von Kick ist keine weitere Aktion mehr durchführbar.*

5.3.2 Suche nach der besten Aktionsfolge

Ist ein Roboter in der Lage, eine Vielzahl von Aktionen auszuführen, die zudem zu Sequenzen zusammengesetzt werden könnten, so müsste, der bisherigen Modellierung folgend, eine große Menge einzelner Verhalten instantiiert werden. Die automatische Suche nach der besten Aktionsfolge ermöglicht es, einem Aktionsfeld eine Menge $\mathbb{A} = \{A_1, A_2, \cdots, A_n\}$ aller Aktionen zusammen mit der maximalen Länge d einer Sequenz zuzuordnen.

Entsprechend dieser Vorgaben können nun automatisch sämtliche durchführbaren Sequenzen $A_{s_1}, A_{s_2}, \ldots, A_{s_m}$ (mit $A_{s_i} \in \mathbb{A}$ und $1 \leq m \leq n$) gebildet werden. Abbildung 5.6 zeigt ein Beispiel mit drei Aktionen.

Zu jeder dieser Sequenzen kann eine Bewertung entsprechend des im vorherigen Abschnitt beschriebenen Verfahrens durchgeführt werden, so dass eine Gesamtbe-

wertung des Verhaltens auf dem Wert $\phi\left(A_{s_m}\right)$ beziehungsweise $\phi_G\left(A_{s_m}\right)$ der bestbewerteten Sequenz erfolgen kann. Ebenso werden im Ergebnis des Aktionsfelds der Name sowie die Parameter der ersten Aktion A_{s_1} dieser Folge vermerkt.

5.3.3 Größe des Suchbaums

Die Größe des bei der Bildung der Aktionssequenzen erzeugten Suchbaums nimmt, abhängig von der Suchtiefe d, exponentiell zu, so dass für die maximale Anzahl von Knoten K_{max} bei einer Anzahl n möglicher Aktionen gilt:

$$K_{max} = \sum_{i=1}^{d} n^i \tag{5.22}$$

Da das Verfahren intern als eine beschränkte Tiefensuche realisiert ist, muss allerdings lediglich Speicherplatz für $d \cdot n$ Knoten alloziert werden.

Sind in $\mathbb{A}$ Aktionen enthalten, die nicht immer durchführbar sind, beziehungsweise verändern Aktionen, wie beispielsweise der Schuss eines Balls, das Modell der Umwelt derart, dass keine weiteren Aktionen folgen können, so resultiert dies in einem kleineren Suchbaum. In dem in Abbildung 5.6 gezeigten Beispiel mit drei Aktionen und der Suchtiefe $d = 3$ wäre $K_{max} = 39$, de facto werden jedoch nur 21 Knoten erzeugt, da *Kick* eine Sequenz stets terminiert[3]. Sind einem Aktionsfeld ausschließlich solche sequenzterminierenden Aktionen zugeordnet, so führt dies, unabhängig von d, zu einem sehr kleinen Suchbaum.

Im allgemeinen Fall und unter zusätzlicher Betrachtung der Tatsache, dass sowohl die Position des Roboters als auch die Positionen eventuell zu transformierender Objekte über eine Anzahl von Hypothesen beschrieben sein können, ist das Verfahren zur automatischen Bildung von Aktionsfolgen lediglich zur Bewertung kurzer Sequenzen geeignet. Für längerfristige Betrachtungen, sofern diese in einer dynamischen Umgebung sinnvoll sind, sollten ensprechend spezialisierte Planungsverfahren verwendet werden.

5.3.4 Verkleinerung des Suchbaums

Aufgrund des exponentiellen Wachstums des Suchbaums kann eine effizientere Suche nach Aktionssequenzen nur durch dessen Verkleinerung, also durch das Abschneiden von Teilbäumen, erzielt werden. Ein solches Vorgehen wird als *Pruning*

[3] Der Sonderfall, dass durch eine sehr nahes Hindernis die Position des Ball nahezu unverändert bleibt und weitere Aktionen möglich sind, sei hier nicht betrachtet.

(engl. für Gehölzschnitt) bezeichnet. Zu diesem Zweck ist eine Heuristik implementiert worden, deren Anwendung bei der Modellierung ausgewählt werden kann.

Durch die Vorgabe der Bedingung, dass eine Aktionssequenz eine kontinuierliche Zustandsverbesserung realisieren soll, werden Knoten, die keine bessere Bewertung haben als ihr Vaterknoten, nicht mehr in den Suchbaum aufgenommen. Für zwei innerhalb einer Sequenz aufeinanderfolgende Aktionen A_{s_n} und $A_{s_{n+1}}$ muss also immer gelten:

$$\phi\left(A_{s_n}\right) > \phi\left(A_{s_{n+1}}\right) \quad \text{bzw.} \tag{5.23}$$

$$\phi_G\left(A_{s_n}\right) > \phi_G\left(A_{s_{n+1}}\right) \tag{5.24}$$

Da es mitunter auch nötig sein kann, zustandsverschlechternde Aktionen auszuführen, um ein besseres Gesamtergebnis zu erzielen, muss zwischen dem Einsatz dieses Verfahrens und einem größeren Suchbaum abgewogen werden.

Kapitel 6

Anwendungen, Ergebnis und Bewertung

Dieses Kapitel behandelt die konkreten Szenarien, in denen die Verhaltensarchitektur bisher eingesetzt wurde. Neben der technischen Umsetzung werden zudem die erzielten Resultate beschrieben und bewertet. Des Weiteren werden Zahlen zum Ressourcenverbrauch des Ansatzes aufgeführt und ein Ausblick auf mögliche Erweiterungen der Verhaltensarchitektur gegeben. Das Kapitel schließt mit der Zusammenfassung des erzielten Gesamtergebnisses.

6.1 Einsatz in der Sony-Liga

Die primäre Testumgebung der Verhaltenssteuerung ist das Fußballspielen in der Sony-Liga. Im Folgenden werden die Integration in die GT2003-Architektur des GermanTeam, die Modellierung der Umwelt sowie die implementierten Verhalten beschrieben.

6.1.1 Die GT2003-Architektur

Da sich das GermanTeam aus Arbeitsgruppen verschiedener Universitäten zusammensetzt, die jedoch bei regionalen RoboCup-Wettbewerben gegeneinander antreten, ist eine Architektur entwickelt worden, die ein zum Fußballspielen benötigtes Gesamtsystem in zahlreiche Module mit fest definierten Schnittstellen unterteilt, wie in Abbildung 6.1 zu sehen ist.

Mittels einer Reihe von Modulen zur Verarbeitung der Sensordaten und zur Lokalisierung wird ein Weltmodell erzeugt, welches unter anderem die Eigenposition

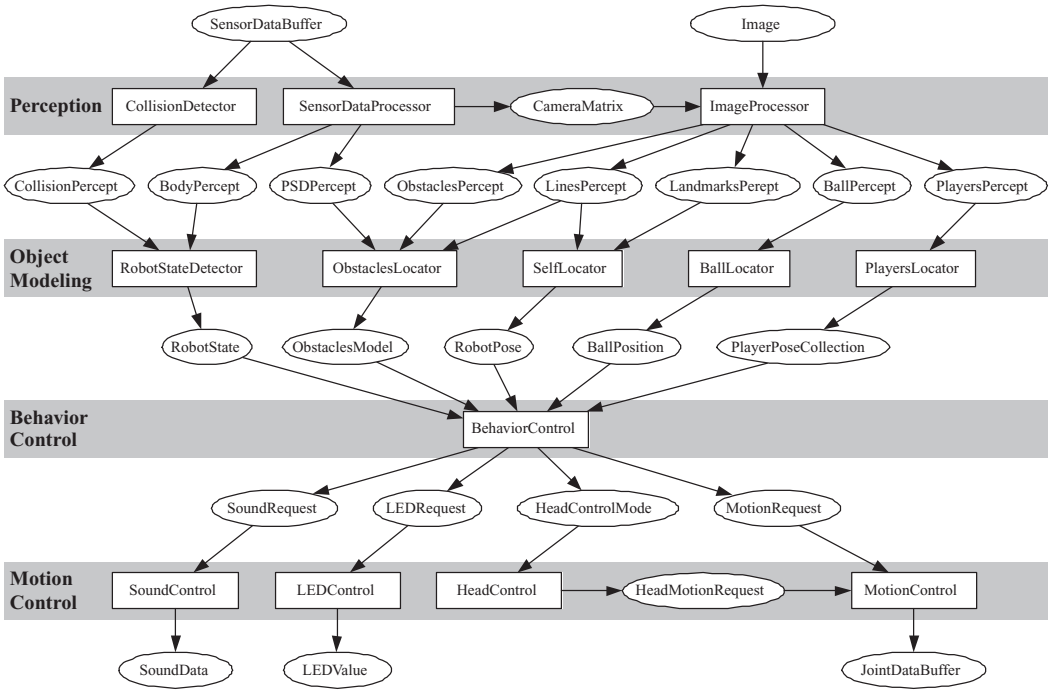


Abbildung 6.1: *Die Module und Schnittstellen der GT2003-Architektur (entnommen aus [Röfer et al., 2003]).*

des Roboters, die Position des Ball sowie die Positionen anderer Roboter auf dem Spielfeld enthält. Die Ansteuerung der Aktuatorik geschieht ebenfalls über mehrere Module, von denen *MotionControl* die größte Bedeutung zukommt, da es sämtliche Bewegungen des Roboters steuert. Die Verhaltenssteuerung (*BehaviorControl*) hat somit die Funktion einer Schnittstelle zwischen Sensorik und Aktuatorik, da sie das Weltmodell auf Aktionen abbildet. Eine direkte Kopplung zwischen Ein- und Ausgabeschicht ist zwar theoretisch ebenfalls möglich, wird aber im Allgemeinen nicht verwendet.

Die Module sind zunächst abstrakt definiert, so dass jeweils die Implementierung einer oder mehrerer konkreter Instanzen nötig ist. Ein Gesamtsystem kann letztendlich aus einer Menge von alternativen Modulen theoretisch beliebig zusammengesetzt werden. Die entwickelte Verhaltenssteuerung wurde im Rahmen einer Instanz von *BehaviorControl* eingesetzt, die von den *Bremen Byters*, einer RoboCup-Mannschaft der Universität Bremen, verwendet wurde.

Ausführliche Informationen zur GT2003-Architektur finden sich in [Röfer, 2003] sowie in der aktuellen Teambeschreibung [Röfer et al., 2003].

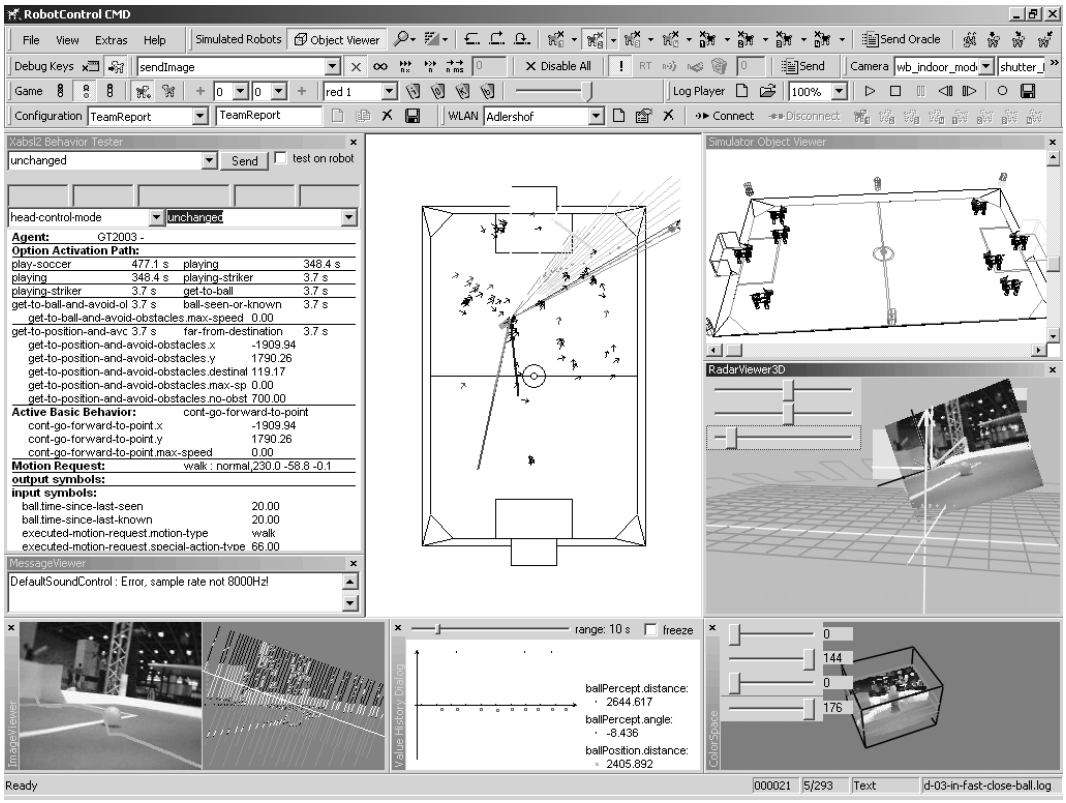


Abbildung 6.2: *Die RobotControl-Umgebung*

6.1.2 Die RobotControl-Umgebung

Zusammen mit der GT2003-Architektur ist das Programm *RobotControl* entwickelt worden. Dabei handelt es sich um eine Umgebung, die es ermöglicht, sämtliche entwickelten Module auf einem PC auszuführen und somit zu testen und auf Fehler zu untersuchen. Ebenso sind Funktionen zur Steuerung eines Roboters und Möglichkeiten des Datenaustauschs integriert.

Darauf aufbauend ist eine Vielzahl graphischer Dialoge und Visualisierungsmöglichkeiten entstanden, die zum Teil in Abbildung 6.2 zu sehen sind. Einige Abbildungen in dieser Diplomarbeit, wie beispielsweise 4.12, sind in RobotControl erzeugt worden.

Durch den integrierten Simulator *SimRobot* [Röfer, 2002] war es möglich, die Verhaltensarchitektur sowie die entwickelten Verhalten zunächst ohne einen Roboter zu testen und Modellierungsparameter anzupassen. Da es in der Simulation zudem möglich ist, ein vollständiges und korrektes Weltmodell zu verwenden, konnte an komplexeren Verhaltenskomponenten, wie beispielsweise Bewegungen relativ zu

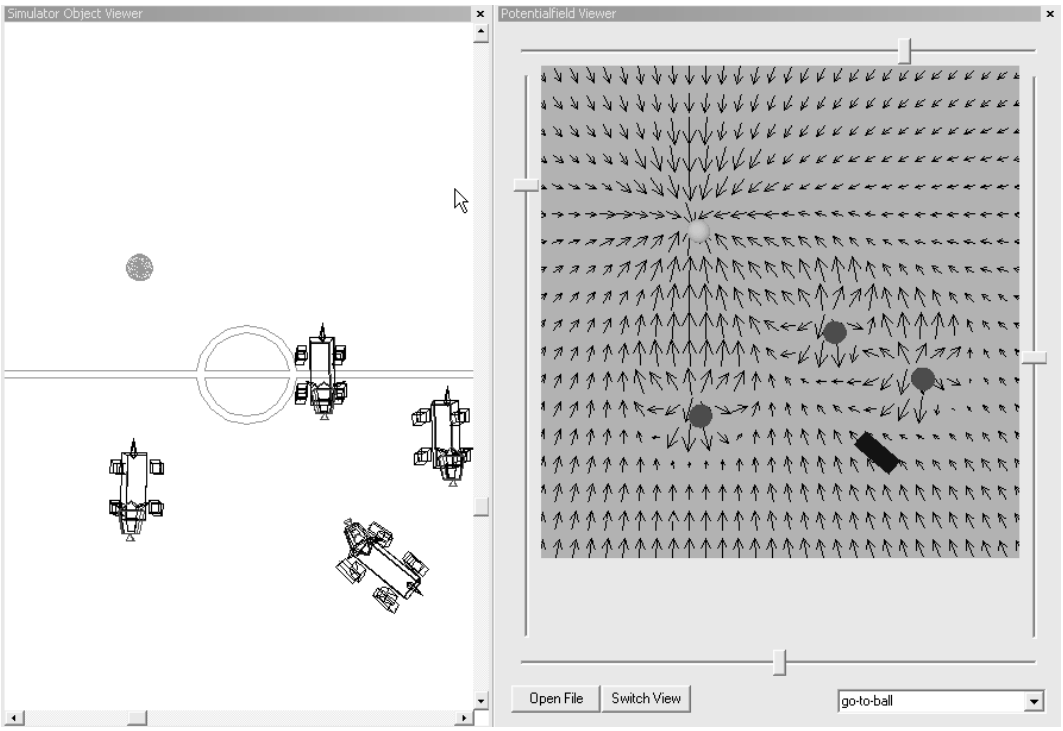


Abbildung 6.3: *Visualisierung eines Potentialfelds, welches den unten rechts in der Simulationsszene platzierten Roboter zu einem Ball führt.*

anderen Objekten, gearbeitet werden, ohne Rücksicht auf ein möglicherweise unvollständiges oder fehlerhaftes Bild der Umgebung nehmen zu müssen. In Abbildung 6.3 ist ein Ausschnitt einer Simulationsszene zusammen mit einer Visualisierung eines Potentialfelds, die basierend auf den in Abschnitt A.6 beschriebenen Funktionen entwickelt worden ist, zu sehen.

6.1.3 Einbettung der Verhaltensarchitektur

Die Verhaltensarchitektur wird nicht als eine Instanz von *BehaviorControl* verwendet, sondern innerhalb dieses Moduls von weiteren Verfahren genutzt. Zu diesem Zweck ist die Implementierung vollständig in einer Klasse *GTPotentialfields* gekapselt worden, die zusätzlich Funktionen zur Konvertierung des GT2003-Weltmodells enthält. Dadurch ist es auf einfache Weise möglich, eine Vielzahl von Instanzen der Verhaltenssteuerung zu erzeugen und zu nutzen.

Zwischen den einzelnen Arbeitsgruppen des GermanTeam wurde vereinbart, die *XABSL-Architektur* von [Lötzsch et al., 2004] als gemeinsame Basis für die Verhaltensentwicklung zu verwenden, um eine bessere Integration der jeweils imple-

mentierten Verhalten und Erweiterungen vor den RoboCup-Weltmeisterschaften zu ermöglichen.

Mittels der in XML spezifizierten Beschreibungssprache XABSL ist es möglich, ein Gesamtverhalten basierend auf einer Menge hierarchisch angeordneter Einzelverhalten, sogenannten *Optionen*, zu modellieren. Zur Bestimmung der nächsten Aktion wird die Hierarchie, die einem azyklischen gerichteten Graphen entspricht, von der Wurzeloption bis zum Erreichen eines ausführbaren Basisverhaltens (*BasicBehavior*) durchlaufen. Innerhalb einer Option wird ein Verhalten über einen Zustandsautomaten beschrieben, von dessen aktuellem Zustand die Transitionen zu einer weiteren Option oder zu einem Basisverhalten abhängt. Letztere stellen die Schnittstelle zur Bewegungssteuerung des Roboters dar und werden daher nicht in XABSL spezifiziert, sondern direkt in C++ implementiert.

Die Integration der Potentialfeldarchitektur in ein mit XABSL beschriebenes Verhalten war zunächst ausschließlich auf die Basisverhalten beschränkt. Durch die Modellierung umfangreicher Potentialfeldverhalten wurden Optionen lediglich zur Beschreibung der Spielzustände verwendet.

Da dieses Vorgehen zu einigen, später in Abschnitt 6.1.5 beschriebenen, Problemen führte, wurden Instanzen der Verhaltenssteuerung zusätzlich in Optionen integriert, so dass die berechneten Ergebnisse für Transitionen zu entsprechenden Ausführungszuständen verwendet werden konnten.

Abbildung 6.4 zeigt ein XABSL-Verhalten am Beispiel eines Torhüters.

6.1.4 Modellierung der Umgebung

Da sich die von den Lokalisierungsverfahren berechneten Positionen auf ein absolutes Koordinatensystem beziehen, können sie direkt in Objektzustände konvertiert werden. Dies ermöglicht zudem die Zuweisung statischer Positionen an Objekte, deren Anordnung bereits bekannt ist, wie beispielsweise die Banden und Tore.

Ein Großteil der Verhalten verwendet eine gemeinsame Modellierung der Umgebung. Im Folgenden werden die wichtigsten Objekte beschrieben; einige spezielle Modellierungen werden in Abschnitt 6.1.5 zusammen mit den jeweiligen Verhalten erläutert.

Modellierung statischer Objekte

Die Struktur der Umgebung eines RoboCup-Spiels ist aufgrund des festen Regelwerks stets gleich und kann mittels folgender Objekte modelliert werden:

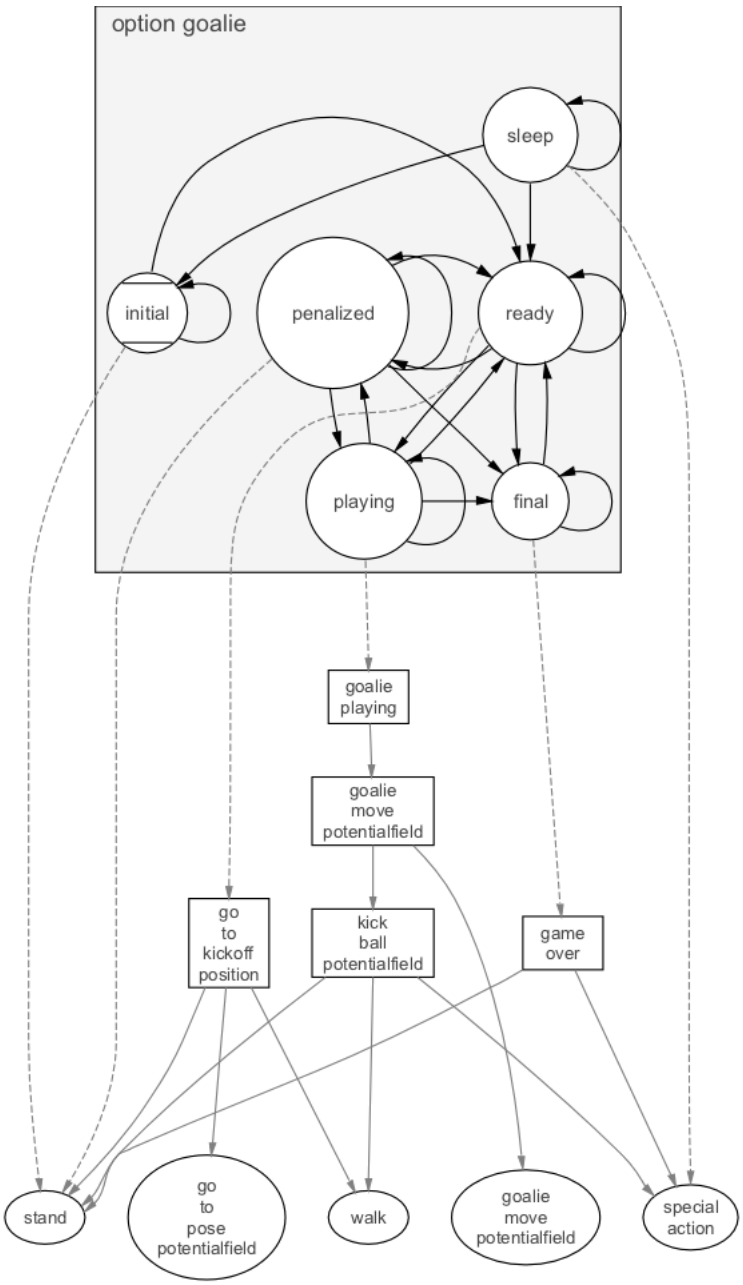


Abbildung 6.4: *Das XABSL-Verhalten eines Torhüters. Innerhalb der obersten, in dieser Abbildung vergrößert dargestellten, Option werden einzelne Spielzustände unterschieden. Dies führt zu Transitionen zu weiteren Optionen (von Rechtecken umgeben) oder Basisverhalten (Ellipsen am unteren Rand).*

Banden Das Spielfeld ist von einer Bande umgeben, die durch einzelne Strecken beschrieben werden kann, denen jeweils ein repulsives Feld kurzer Reichweite zugeordnet ist, um Zusammenstöße zu vermeiden. Zur Auswahl von Aktionen existiert eine separate Menge von Bandenobjekten, die kein Feld erzeugen und lediglich zur Kollisionserkennung bei Schüssen verwendet werden. Zu diesem Zweck wurden zusätzlich die Wände des gegnerischen Tores modelliert.

Eigener Strafraum Der eigene Strafraum darf nicht von Feldspielern betreten werden und wird daher vollständig von einem starken repulsiven Feld abgedeckt. Dies hat zudem den Effekt, dass Aktionen, die den Ball in die Nähe des eigenen Tors bewegen, sehr schlecht bewertet werden.

Gegnerische Spielhälfte Bei der Bewertung von Aktionen wird ein linear in Richtung der gegnerischen Torlinie abfallendes, attraktives Feld mit einer geringen Steigung verwendet. Die Reichweite erstreckt sich über das gesamte Feld, so dass Aktionen in Richtung des gegnerischen Tores tendenziell besser bewertet werden.

Gegnerisches Tor Zusätzlich ist die Grundfläche des Tors mit einem stark attraktiven Feld versehen, so dass eine Aktion, die den Ball dorthin befördern kann, sehr wahrscheinlich ausgewählt wird.

Modellierung dynamischer Objekte

Die Menge möglicher beweglicher Objekte auf einem Spielfeld ist ebenfalls im Vorfeld bekannt. Da deren Positionen jedoch nicht zu jeder Zeit bestimmbar sind, werden die entsprechenden Objektzustände vom Konverter, abhängig von der vergangenen Zeit seit der letzten Lokalisierung, aktiviert beziehungsweise deaktiviert. Folgende Objekte sind modelliert worden:

Ball Dem Ball ist keine geometrische Figur zugeordnet. Mittels eines zentrierten attraktiven Feldes, dessen Reichweite das gesamte Feld abdeckt, kann er als Zielposition für ein Bewegungsverhalten verwendet werden. Zusätzlich ist der näheren Umgebung des Balls ein weiteres, sehr stark attraktives, Feld zugeordnet worden, dass mögliche repulsive Felder von gegnerischen Robotern in Ballnähe überlagert. Dadurch können Ausweichbewegungen vermieden werden, die einen anschließenden Ballverlust zur Folge hätten. Zur Aktionsauswahl wird lediglich ein Ballobjekt ohne ein Feld verwendet.

Mitspieler Da sich der Roboter, zum einen aus taktischen Gründen und zum anderen zur Vermeidung von Kollisionen, möglichst nicht in der Nähe seiner

Mitspieler aufhalten sollte, wird jeder Roboter der eigenen Mannschaft von einem relativ weit reichenden, repulsiven Feld umgeben. Die Form der Roboter wird über einen Kreis angenähert. Bei der Bewertung von Aktionen werden keine Felder verwendet sondern lediglich eine Kollisionserkennung durchgeführt.

Gegenspieler Die Spieler der gegnerischen Mannschaft sind ähnlich modelliert. Es wird jedoch ein Feld kürzerer Reichweite verwendet, da zwar Kollisionen vermieden werden sollen, eine weitläufige Vermeidung anderer Roboter aus taktischen Gründen aber nicht sinnvoll ist. Aktionen, die den Ball in die Nähe eines Gegenspielers bewegen, sind negativ zu bewerten, daher wird die gleiche Modellierung auch für Aktionsfelder verwendet.

Hindernisse Eine spezielle Form von Objekten sind Hindernisse. Basierend auf Messungen eines Entfernungssensors sowie einigen Perzepten der Bildverarbeitung, wird ein Modell erzeugt, welches im Umkreis des Roboters zu jedem Winkel die Entfernung zum einem möglichen Hindernis der näheren Umgebung enthält. Dieses wird vom Konverter in fünf, in gleichmäßigen Winkelabständen in einem Intervall von $[-\frac{\pi}{2}, \frac{\pi}{2}]$ relativ zur Roboterausrichtung liegende, Objektzustände umgewandelt, auf denen repulsive Objekte, ähnlich den Gegenspielern, basieren.

6.1.5 Implementierte Verhalten

Sowohl für den Einsatz bei den *RoboCup German Open* als auch zum Test einzelner Aspekte der Architektur ist eine Reihe von Verhalten für die Bremen Byters entwickelt worden. Die wichtigsten werden im Folgenden beschrieben:

Zum Ball bewegen und schießen

Dem Konzept möglichst komplexer Basisverhalten, wie in Abschnitt 6.1.3 beschrieben, folgend, ist für die German Open ein Verhalten entwickelt worden, dass einen Roboter zu einem Ball führt und anschließend einen geeigneten Schuss auswählt. Hierzu sind insgesamt acht verschiedene Felder verwendet worden.

Zwei Bewegungsfelder, deren Freiheitsgrade sich gemäß Abschnitt 4.4.2 überlagern, berechnen einen kollisionsfreien Weg unter permanenter Ausrichtung zum Ball. Der Bewegung ist ein konstanter Aktivierungswert zugeordnet, so dass sie standardmäßig ausgewählt wird.

Der Roboter ist in der Lage, eine Vielzahl verschiedener Schüsse auszuführen, von denen sechs geeignete ausgewählt und jeweils in einem eigenen Aktionsfeld als eine feste Translation eines Objekts (siehe Abschnitt 5.2.1) modelliert worden sind. Befindet sich der Roboter in der Nähe des Balls, überlagern die ausführbaren Aktionsfelder, die eine Zustandsverbesserung der Ballposition berechnet haben, die Bewegungsverhalten und der bestmögliche Schuss wird ausgeführt.

Da die errechnete Ballposition jedoch stets um einige Zentimeter schwankt und somit nur schwer vorherzusagen ist, wann ein Schuss zuverlässig ausgeführt werden kann, musste die Ausführung einer Aktion, die sich zuvor auf die Aktivierung der entsprechenden Bewegung im MotionControl-Modul beschränkte, komplexer gestaltet werden. Dies führte zu einer Trennung der Bewegungskomponente von der Aktionsauswahl, die daraufhin, wie im folgenden Abschnitt beschrieben, realisiert wurde.

Auswahl des besten Schusses

Ein Verfahren zur relativ zuverlässigen Durchführung eines Schusses in der Sony-Liga ist die Abarbeitung einer kurzen Bewegungssequenz nach der Auswahl der entsprechenden Aktion. Dabei wird der zulässige Ausführungsbereich der Transformation relativ großzügig beschrieben und vor der Aktivierung des Schusses stets eine zeitlich begrenzte Vorwärtsbewegung durchgeführt, die in den meisten Fällen zu einer optimalen Schussposition führt. Dieses Vorgehen lässt sich mittels eines Zustandsautomaten realisieren. Da eine entsprechende Umsetzung innerhalb eines Basisverhaltens, verglichen mit einer Beschreibung in XABSL, relativ aufwändig gewesen wäre, ist die Aktionsauswahl in eine Option integriert worden, die in Abbildung 6.4 als *kick ball potentialfield* bezeichnet ist

Die Verhaltenssteuerung fungiert dabei als ein sogenanntes *Symbol*. Die berechneten Ergebnisse werden mittels eines speziellen Interpreters in Symbolwerte überführt, anhand derer ein Zustandsautomat in die entsprechenden Ausführungszustände wechseln kann. Ist keine Aktion ausführbar, wird eine Transition zu einem auf Bewegungsfeldern basierenden Basisverhalten ausgelöst.

Zusammen mit diesem Umbau wurde die Aktionsauswahl um ein weiteres Feld ergänzt, welches eine feste Aktionssequenz nach Abschnitt 5.3.1 evaluiert. Dabei handelt es sich um eine Rotation von Roboter und Ball in Richtung des Gradienten mit einem anschließenden geraden Schuss.

Einnehmen einer Pose

Zum autonomen Einnehmen einer Startposition ist es nötig, die Roboterpose in eine fest definierte Zielpose überführen zu können. Dazu wurde ein spezielles zentriertes Zielobjekt eingeführt, welches, ähnlich dem Ball, attraktiv über die gesamte Fläche des Spielfelds wirkt. Durch die Verwendung von drei Bewegungsfelder sowie eines Aktionsfelds, welches eine Messung durchführt, konnte eine Navigation unter Vermeidung von Hindernissen mit einer abschließenden Drehung in die vorgegebene Zielrotation modelliert werden.

Bewegung des Torhüters

Das vollständige Bewegungsverhalten eines Torhüters ist durch vier Potentialfelder beschrieben worden. Diese ermöglichen eine Positionierung im Tor, die Ausrichtung zum Ball sowie die Bewegung zu einem nahen Ball, der innerhalb dieses Verhaltens durch ein räumlich begrenztes Feld beschrieben wird.

Ursprünglich waren, ebenso wie im ersten beschriebenen Verhalten, die Schüsse ebenfalls integriert, so dass mittels der Potentialfeldarchitektur das vollständige Verhalten eines Torhüters beschrieben werden konnte.

Bewegungen der Feldspieler

Angelehnt an das Verhalten des Torhüters, ist für Feldspieler ebenfalls ein komplexes Verhalten zur vollständigen Beschreibung aller Bewegungen entwickelt, jedoch noch nicht in einem Spiel getestet worden. Durch vier Aktionsfelder, die die jeweilige Wirkung der Felder aller Spieler der eigenen Mannschaft auf die Ballposition messen, kann ein Roboter bestimmen, ob er selbst zum Ball gehen soll, der Torwart den Ball spielen wird und eine Ausweichbewegung, basierend auf einem repulsiven Feld in Form eines Kreisausschnitts, welches dem Ball zugeordnet ist, durchgeführt werden muss oder eine Bewegung relativ zu einem Mitspieler ausgeführt wird. Die dazu benötigten Bewegungsfelder sind, wie in Abschnitt 2.2.3 beschrieben, an die Aktionsfelder gebunden.

Verwendung einer Wegplanung

Es konnten in Tests diverse Bewegungsverhalten durch eine Wegplanung erweitert werden. Diese ist allerdings bisher noch nicht in einem Spiel eingesetzt worden.

6.1.6 Bewertung des Ergebnisses

Die Verhaltensarchitektur wurde, unter Verwendung eines Teils der zuvor beschriebenen Verhalten, erstmals bei den German Open 2003 eingesetzt. In diesem Turnier haben die Bremen Byters ein Spiel knapp gewonnen, zwei Spiele nach vorheriger Führung sowie eines deutlich verloren. In späteren Testspielen, nach der verbesserten Integration der Potentialfelder in die XABSL-Architektur und unter Verwendung eines schnelleren Laufens, konnten gegen den Gewinner des Turniers ein Sieg sowie ein Unentschieden erzielt werden.

Derartige Spielergebnisse sagen allerdings im Allgemeinen sehr wenig über die Qualität einer Verhaltensarchitektur aus. Die Spielstärke einer Mannschaft resultiert vielmehr aus dem Inhalt des modellierten Verhaltens, dem Zusammenspiel mit den anderen Modulen sowie aus deren Qualität und Zuverlässigkeit.

Dennoch konnten durch den Einsatz der Potentialfeldarchitektur in einer Reihe von Spielen einige Erkenntnisse gewonnen werden. Abgesehen von der Tatsache, dass ein grundlegender Beweis für die Funktionsfähigkeit der Implementierung erbracht wurde, war es möglich, sämtliche Bewegungen sowie die Auswahl aller Schüsse durch einzelne, auf Potentialfeldern oder Potentialfunktionen basierende, Verhalten oder deren Kombination zu beschreiben und somit, zum Teil unter Verwendung der XABSL-Architektur, ein Gesamtverhalten zum Fußball spielen zu erzeugen.

Während die Konvertierung des Weltmodells des Roboters problemlos vorgenommen werden konnte, führte die Auswahl von Aktionen, wie bereits beschrieben, zu einigen Problemen. Da die Verhaltenssteuerung, wie in Abschnitt 5.1.1 definiert, stets nur eine auszuführende Aktion benennt, ist deren spätere Durchführung mit anderen Mitteln zu beschreiben.

Die externen Dateien zur Verhaltensmodellierung erwiesen sich als sehr gut handhabbar. Neben einer kompakten und übersichtlichen Beschreibung eines Verhaltens, war die Möglichkeit kurzfristiger Änderungen von großem Vorteil, da einzelne Umstellungen sogar in einer Halbzeitpause eines Spiels durchgeführt werden konnten.

Die Beschreibung eines Verhaltens durch Potentialfelder stellt einen Gegensatz zu einem rein in XABSL modellierten Verhalten dar. Während die hier vorgestellte Architektur auf der Modellierung der Umgebung, einzelner Ziele sowie den möglichen Aktionen mit einer anschließenden selbständigen Bewertung mittels kontinuierlicher Berechnungsverfahren beruht, müssen in XABSL, das allerdings die Modellierung beliebig komplexer, hierarchisch angeordneter Verhalten ermöglicht, sämtliche Abläufe detailliert beschrieben und eine Vielzahl diskreter Entscheidungen getroffen werden. Mittlerweile sind in die Architektur ebenfalls Potentialfelder in Form sogenannter *ContinuousBasicBehaviors* integriert worden. Diese werden jedoch in C++ beschrieben und sind speziell auf die Domäne der Sony-Liga zugeschnitten

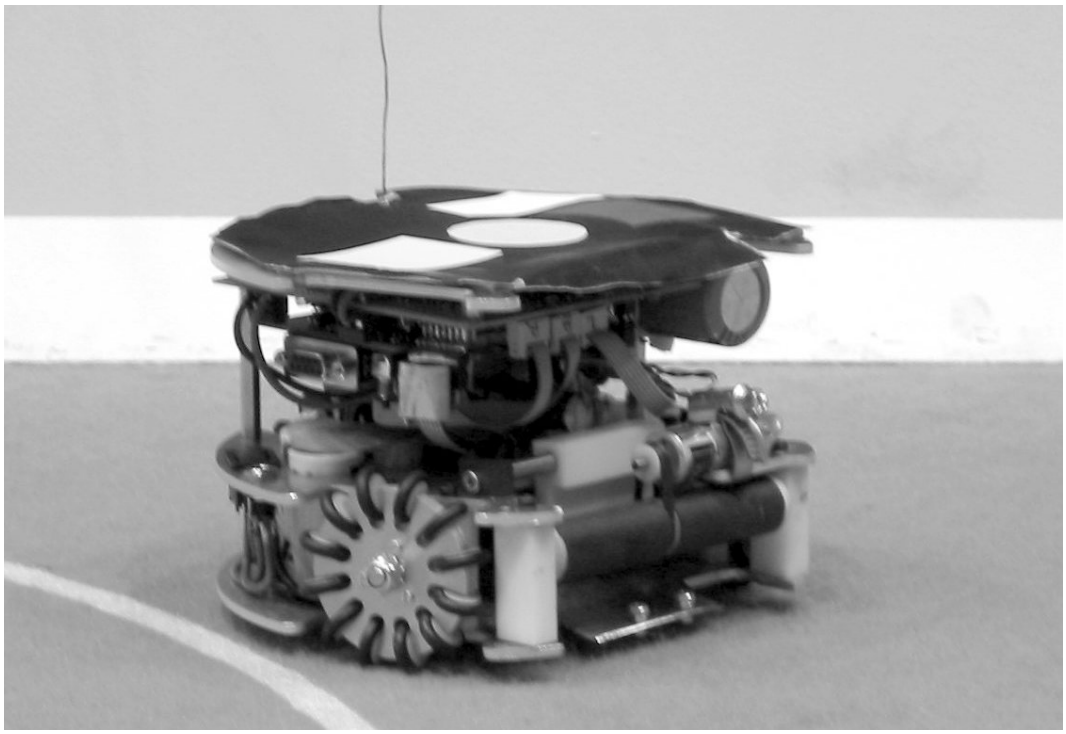


Abbildung 6.5: *Ein Small-Size-Roboter der Mannschaft B-Smart.*

und somit nicht direkt auf andere Umgebungen übertragbar. Der Funktionsumfang entspricht den in dieser Arbeit in Abschnitt 4.1 beschriebenen Verfahren zusammen mit einigen speziellen, Elemente des Feldes und Roboter repräsentierenden, Objekten. Die Möglichkeit einer Bewertung von Aktionen durch Potentialfunktionen ist bisher noch nicht implementiert worden.

6.2 Übertragung auf Small-Size-Roboter

Zum Test der Portierbarkeit des Ansatzes ist die Verhaltenssteuerung auf eine weitere Plattform übertragen worden: Die Roboter der RoboCup Small-Size-Mannschaft *B-Smart* der Universität Bremen.

6.2.1 Die Small-Size-Liga

Die *RoboCup Small Size League*, im Folgenden als Small-Size-Liga bezeichnet, ist eine weitere Fußball-Liga im Rahmen des RoboCup. Das Szenario ist vergleichbar mit dem der Sony-Liga, welches bereits in Abschnitt 1.4.2 beschrieben wurde.

Abgesehen von den Dimensionen der Umgebung und der Anzahl von fünf statt vier Spielern pro Mannschaft, sind die signifikantesten Unterschiede die Art der Roboter sowie die Möglichkeit externer Sensoren und Steuerungsmechanismen.

Jeder Mannschaft ist es erlaubt, eine beliebige Anzahl an Kameras über oder neben dem Spielfeld anzubringen, die mit einem oder mehreren externen Rechnern verbunden sind. Dadurch ist es möglich, ein sehr genaues Weltmodell aufzubauen und die Spielzüge sämtlicher Roboter von zentraler Stelle zu steuern. Im Allgemeinen werden per Funk lediglich Bewegungskommandos an die einzelnen Roboter verschickt.

Die Roboter werden von den Mannschaften selbst konstruiert. Die Spielregeln sehen dabei eine maximale Höhe (150 Millimeter) sowie einen maximalen Durchmesser der Grundfläche (180 Millimeter) vor. Bezüglich der Anzahl der Räder und Motoren, Art der Schussmechanismen sowie sonstiger Erweiterungen sind den Teilnehmern kaum Grenzen gesetzt. Die bei den Weltmeisterschaften 2003 eingesetzten Roboter verfügten größtenteils über omnidirektionale Antriebe, die Spitzengeschwindigkeiten von bis zu zwei Metern pro Sekunde ermöglichen. Abbildung 6.5 zeigt einen solchen Roboter.

Das genaue Weltmodell, die hohe Rechenkapazität und die schnellen Roboter führen zu einem, verglichen mit den anderen Roboter-Ligen, sehr dynamischen Spielgeschehen und komplexen Spielzügen. Ausführlichere Informationen zur Small-Size-Liga finden sich auf den offiziellen Internet-Seiten [Browning, B., 2003].

6.2.2 Integration der Verhaltenssteuerung

Das von B-Smart verwendete System sowie deren differential angetriebene Roboter werden ausführlich in [Hoppe et al., 2004] beschrieben. Kurz vor den Weltmeisterschaften 2003 wurde zusätzlich ein omnidirektional fahrender Roboter entwickelt. Abbildung 6.6 zeigt die vollständige Mannschaft.

Aufgrund der zentralen Steuerung sämtlicher Roboter existierte bereits vor der Integration der Verhaltensarchitektur eine, sich an dem Konzept der *Spielbücher* von [D'Andrea et al., 2001] orientierende, Umgebung, die einen Rahmen für die Verhalten der einzelnen Roboter darstellt. Anhand der Position des Balls im Weltmodell werden verschiedene Spielsituationen, wie beispielsweise Offensive oder Defensive, unterschieden und den Robotern dementsprechende Aufgaben zugeordnet. Spezielle Spielzustände, wie Freistöße oder Anstoß, werden ebenfalls behandelt.

Die Aufgaben, auch als *Skills* bezeichnet, beziehen sich im Allgemeinen auf das Anfahren bestimmter vorberechneter Positionen. Da keiner der Roboter über Mechanismen für eine Bewegung mit dem Ball verfügt und lediglich gerade Schüsse

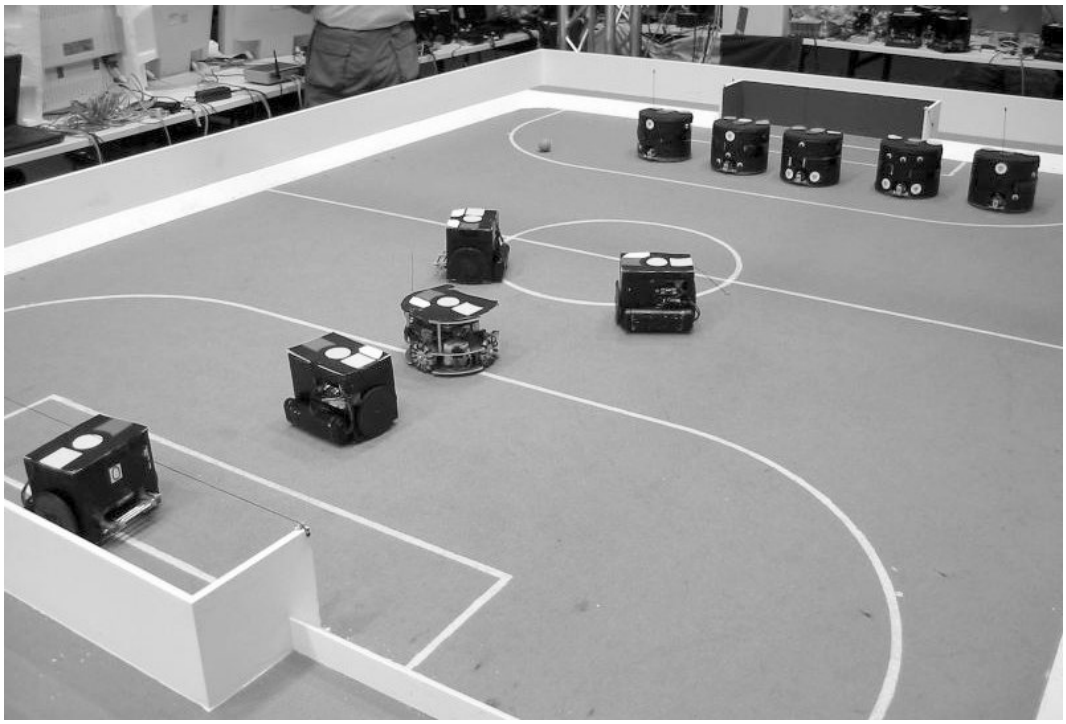


Abbildung 6.6: *Die Roboter von B-Smart vor dem Anstoss (aufgenommen bei den Weltmeisterschaften 2003 in Padua)*

möglich sind, lassen sich derartige Aktionen ebenfalls auf die Bestimmung einer günstigen Schussposition reduzieren.

Aufgrund dieser Begebenheiten wurde der Einsatz der Potentialfelder auf die Bestimmung von Bewegungsrichtungen reduziert. Analog zum, in Abschnitt 6.1.3 beschriebenen, Vorgehen in der Sony-Liga ist die Verhaltenssteuerung zusammen mit einem Konverter des Weltmodells in einer Klasse gekapselt worden. Durch die große Ähnlichkeit der beiden Umgebungen konnten die benötigten Bewegungsfelder zusammen mit den entsprechenden Objekten direkt aus dem Verhalten der Bremen Byters übernommen werden. Lediglich die Anzahl der Objektinstanzen sowie die Parameter der geometrischen Figuren und die Reichweiten der einzelnen Felder mussten angepasst werden. Die erzeugten Bewegungsparameter konnten direkt auf den Wertebereich des omnidirektional fahrenden Roboters skaliert werden. Für die differential angetriebenen Fahrzeuge war ein zusätzlicher Verarbeitungsschritt nötig, um den vorgegebenen Bewegungsrichtungen folgen zu können.

6.2.3 Ergebnis und Bewertung

Da die vier differential angetriebenen Roboter bei den Weltmeisterschaften in keinster Weise konkurrenzfähig waren, wurden sie lediglich als Torhüter und Abwehrspieler eingesetzt, mit der Aufgabe, herannahende Bälle und Gegenspieler zu blockieren.

Der verbleibende, omnidirektional fahrende Roboter hatte daraufhin die permanente Aufgabe, den Ball unabhängig von dessen Position zu erreichen und ihn in Richtung des gegnerischen Tors zu spielen. Obwohl zwischen der Fertigstellung des Roboters und seinem ersten Spiel nur 36 Stunden lagen, war es möglich, ein sehr schnelles, kollisionsfreies Bewegungsverhalten umzusetzen. Selbst in Spielen gegen starke Mannschaften konnte der Ball oftmals vor einem Gegenspieler erreicht werden, was allerdings aufgrund der mangelnden Möglichkeiten zur Ballführung selten zu einem größeren Vorteil führte.

Die Mannschaft konnte in vier Spielen zwei Tore erzielen und war insgesamt gesehen chancenlos. Im Bezug auf die Portierbarkeit der Verhaltensarchitektur konnte jedoch festgestellt werden, dass sich zum einen die Implementierung problemlos auf eine andere Plattform übertragen und dort einbinden ließ und zum anderen, aufgrund der abstrakten Schnittstellen, selbst modellierte Objekte und einzelne Verhalten direkt übertragbar waren.

Weitergehende ausführlichere Tests konnten leider nicht durchgeführt werden, da das verwendete Gesamtsystem nach den Weltmeisterschaften aufgrund eines vollständigen Umbaus nicht wieder in Betrieb genommen wurde.

6.3 Ressourcenverbrauch

Da sowohl Rechenzeit als auch Arbeitsspeicher auf den meisten Robotersystemen nur in stark begrenztem Umfang zur Verfügung stehen, ist die Betrachtung des Ressourcenverbrauchs einer Verhaltenssteuerung notwendig.

6.3.1 Ausführungszeiten einzelner Verhalten

Zur Bestimmung der benötigten Rechenzeit der Verhaltenssteuerung wurde eine Reihe von Versuchen durchgeführt. Als Testplattform diente der in Abschnitt 1.4.3 beschriebene Sony-Roboter. Auf einem Spielfeld wurden verschiedene Szenarien aufgebaut, in denen jeweils ein spezielles Verhalten ausgeführt wurde:

Die *Bewegung zu einem Ball* entspricht dem in Abschnitt 6.1.5 beschriebenen Verhalten. Auf dem Spielfeld wurden mehrere Roboter sowie ein Ball platziert und deren Positionen im Lauf der Zeit verändert. Die Beine des Experimentators dienten als zusätzliche bewegliche Hindernisse.

Der gleiche Versuch wurde später unter Verwendung einer permanenten *Wegplanung* wiederholt.

Zur Bestimmung der benötigten Zeit zur Bewertung *einer Aktion* wurde der Roboter gemeinsam mit einem vor ihm liegenden Ball und mehreren anderen, für ihn sichtbaren Robotern in der Nähe des gegnerischen Tors platziert. Die Bewegungssteuerung wurde vollständig deaktiviert, so dass der Roboter permanent den geraden Schuss eines Balls unter Beachtung verschiedener Hindernisse evaluieren musste.

Der gleiche Aufbau wurde auch zur Messung *mehrerer Aktionen*, entsprechend der in Abschnitt 6.1.5 beschriebenen Schussauswahl, verwendet.

Da ein Roboter während eines Fußballspiels die wenigste Zeit damit verbringt, den Ball tatsächlich zu spielen, wird die Durchführbarkeit der zur Modellierung der Schüsse verwendeten Transformationen permanent mit negativem Ergebnis getestet. Die für die Auswahl letztendlich *keiner Aktion* benötigte Zeit konnte bestimmt werden, indem der Ball, unter ansonsten gegenüber den vorherigen zwei Szenarien unveränderten Bedingungen, deutlich außerhalb des möglichen Aktionsradius platziert wurde.

Die einzelnen Versuche wurden mehrfach wiederholt. Bei jeder Durchführung wurden die Ausführungszeiten von 500 Aufrufen des *BehaviorControl*-Moduls gemessen. Die folgenden Zahlen beschreiben die jeweils durchschnittlich, minimal und maximal benötigte Zeit in Millisekunden. In den Messungen sind stets die Ausführung eines Teils des umgebenden XABSL-Verhaltens, die vollständige Konvertierung des Weltmodells sowie die Interpretation der Ergebnisse enthalten. Deren Gesamtzeit liegt jedoch, wie an der minimalen Ausführungszeit einiger Verhalten zu sehen ist, deutlich unter einer Millisekunde:

Verhalten	∅	Min.	Max.
Bewegung zu einem Ball	4 ms	1 ms	10ms
Wegplanung	22 ms	9 ms	51 ms
Eine Aktion	4 ms	1 ms	8 ms
Mehrere Aktionen	20 ms	18 ms	27 ms
Keine Aktion	2 ms	1 ms	7 ms

Nimmt man die Zeit zwischen zwei Sensormessungen – dies sind auf einem Sony-Roboter 40 Millisekunden – als Maßstab für die angestrebte maximale Ausführungszeit eines Gesamtsystems, so ist die Verhaltenssteuerung im Allgemeinen als

echtzeitfähig zu bezeichnen. Insbesondere die Bewegungssteuerung zeichnet sich durch eine sehr hohe Geschwindigkeit aus. Die Bewertung von Aktionen erscheint als relativ aufwändig, während eines Spiels tritt jedoch zumeist der – wiederum sehr schnell zu berechnende – Fall der Auswahl keiner Aktion ein.

6.3.2 Speicherbedarf

Der Speicherbedarf der Verhaltensarchitektur ist abhängig von der Anzahl und Komplexität der instantiierten Objekte sowie von der Anzahl und Art der verwendeten Verhalten. Da diese jedoch jeweils verschieden zusammengesetzt sein können und gemeinsame Speicherbereiche nutzen, lassen sich für einzelne Größen lediglich großzügige Obergrenzen, basierend auf den verwendeten Datentypen, schätzen.

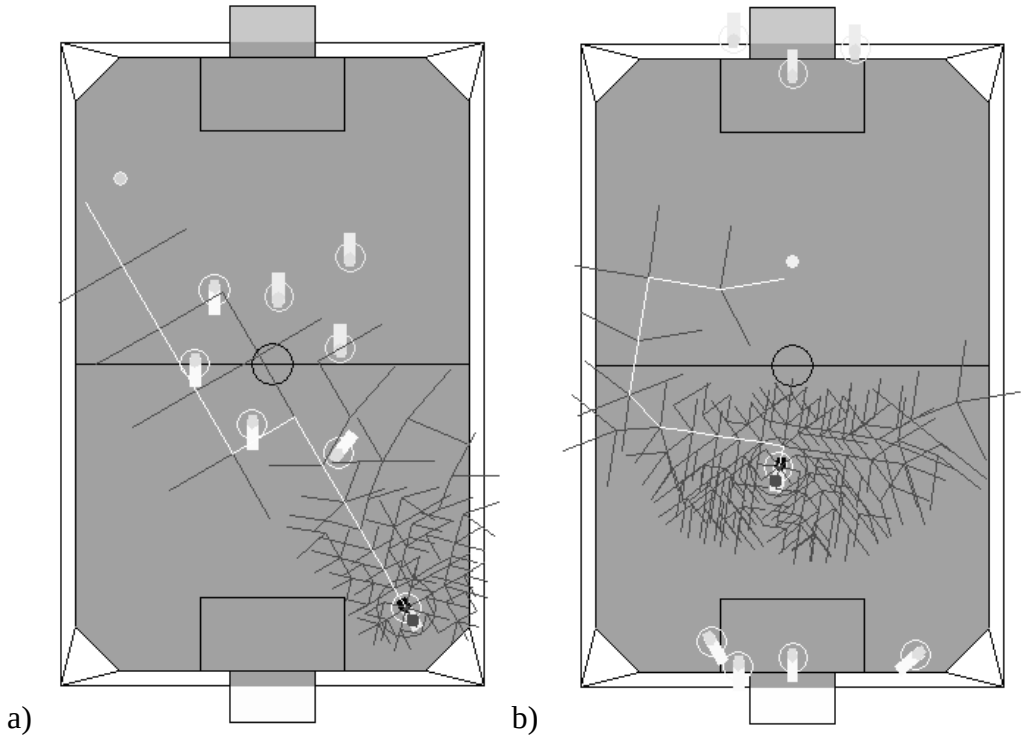
Die Instanz eines Objekts belegt, zusammen mit einer Funktion und einer geometrischen Repräsentation, im Normalfall[1] stets weniger als 400 Bytes.

Ein gewöhnliches Bewegungsverhalten kann, abgesehen von der theoretisch beliebig großen Anzahl an Referenzen auf Objekte, nicht größer als ein Kilobyte sein. Wird jedoch eine Wegplanung verwendet, so muss zusätzlicher Speicher für die Knoten des Suchbaums reserviert werden. Die Größe eines solchen Knotens beträgt, inklusive seiner Speicheradresse, 100 Bytes. Die Anzahl der benötigten Knoten wird im folgenden Abschnitt betrachtet.

Einem Feld zur Aktionsauswahl können beliebig viele Aktionen zusammen mit Transformationen zugeordnet sein, die gemeinsam circa 200 Bytes belegen. Ferner wird Speicher zur Repräsentation zukünftiger Zustände einzelner Objekte benötigt. Dies führt zu einem, verglichen mit der Bewegungssteuerung, größeren Speicherbedarf. Gewöhnliche, in annehmbarer Zeit berechenbare, Verhalten werden jedoch wahrscheinlich stets weniger als zehn Kilobytes belegen.

Der Speicherbedarf ist insgesamt als niedrig einzustufen. Selbst komplexere Gesamtverhalten, beispielsweise bestehend aus 100 Objekten und 20 Einzelverhalten, benötigen lediglich einen Bruchteil eines Megabytes des Arbeitsspeichers. Auf die Allozierung weiteren Speichers zur Laufzeit wird generell verzichtet, da dies zum einen eine sehr rechenaufwändige Operation ist und zum anderen zu unvorhersehbaren Fehlern führen kann.

[1] Theoretisch ist es möglich, beliebig große Polygone zu beschreiben und somit den angegebenen Wert zu überschreiten. In der Praxis ist die Verwendung konvexer Polygone mit deutlich über 20 Eckpunkten jedoch eher selten.



a) b)

Abbildung 6.7: *Zwei Versuche zur Bestimmung der maximalen Anzahl von Knoten in einem Suchbaum: a) Der Roboter muss durch eine Menge einzelner Hindernisse navigieren und b) einen Weg vorbei an einer virtuellen Mauer finden.*

6.3.3 Größe eines Suchbaums zur Wegplanung

Da die maximale Anzahl an Knoten in einem Suchbaum möglichst bei der Modellierung anzugeben ist, wurde zusätzlich die Größe erzeugter Suchbäume betrachtet. Die entsprechenden Versuche wurden im Simulator durchgeführt, da dieser ein vollständiges Weltmodell liefert und dadurch die Anzahl der möglichen Hindernisse stets maximal ist.

Im ersten Szenario wurden sieben Roboter im mittleren Bereich des Spielfelds platziert, wie in Abbildung 6.7a zu sehen ist. Mittels einer Wegplanung musste wiederholt von einer Ecke des Feldes zu einem Ball in der gegenüberliegenden Ecke navigiert werden. Dabei wurden sowohl die Positionen der Roboter als auch die des Balls permanent leicht verändert. Die verwendeten Parameter zur Diskretisierung des Raumes gemäß Abschnitt 4.3.2 waren: $r_{min} = 150mm$, $r_{max} = 500mm$, $b_{min} = 4$, $b_{max} = 8$, $r_{near} = 400mm$ und $r_{far} = 1500mm$. In einem zweiten, in Abbildung 6.7b zu sehenden, Versuch musste der Roboter ebenfalls wieder einen Weg zu einem Ball planen. Der verwendete Parametersatz blieb unverändert, die

anderen Roboter wurden jedoch durch ein einzelnes Hindernis ersetzt. Mittels eines sehr stark repulsiven Feldes, basierend auf der Form eines Rechtecks, wurde die gesamte Mittellinie, mit Ausnahme zweier schmaler Passagen an den Rändern, für den Roboter blockiert.

Es konnte in beiden Versuchen, jeweils nach einer Vielzahl von Durchführungen, sowohl die maximale Anzahl erzeugter als auch erweiterter Knoten bestimmt werden. Im ersten Szenario, dass einem Fußballspiel mit einer stark erhöhten Dichte an Robotern entspricht, ist stets ein Weg unter maximaler Verwendung von 187 Knoten, von denen 89 erweitert worden sind, erfolgreich bestimmt worden. Im zweiten Szenario, welches ein sehr großes lokales Minimum entlang der Kante des Hindernisses erzeugt, mussten jedoch bis zu 640 Knoten, von denen 414 erweitert worden sind, erzeugt werden, um einen Weg zum Ziel zu finden. In beiden Fällen nahm die Größe des Suchbaums nach nahezu vollständiger Überwindung des Hindernisses rapide ab.

Mittels dieser Daten ist es nun möglich, die ungefähre Anzahl an benötigten Knoten abzuschätzen und entsprechenden Speicher zu reservieren. Während das Suchverfahren in Umgebungen mit vielen kleinen Hindernissen sehr gute Ergebnisse erzielt, scheinen Positionen in der Nähe eines großen lokalen Minimums zu einer intensiveren Suche zu führen, so dass bei einem Einsatz des Verfahrens in einer derartigen Umgebung eine Reihe weiterer Experimente mit veränderten Parametersätzen durchgeführt werden sollte.

6.4 Weiterführende Arbeiten

Im Folgenden werden einige mögliche Ansatzpunkte für die Weiterentwicklung und Anwendung der Verhaltensarchitektur aufgezeigt.

6.4.1 Erweiterung auf drei Dimensionen

Wie bereits in Abschnitt 1.3.3 beschrieben, beschränkt sich die Architektur auf das Handeln im zweidimensionalen Raum. Für einen möglichen Einsatz in drei Dimensionen müssten zunächst die elementaren Klassen zur internen Repräsentation von Vektoren und Posen sowie die geometrischen Primitive erweitert werden. Ebenso müssten dreidimensionale Objekte eingeführt werden, wie beispielsweise Polyeder oder Zylinder. Einige solcher Objekte sind bereits in [Khatib, 1986] beschrieben.

Die zentralen Verfahren zur Bestimmung der Bewegungsrichtung und zur Bewertung von Aktionen arbeiten weitestgehend unabhängig von der Anzahl der verwendeten Dimensionen, da sie Vektoren und geometrische Objekte über abstrakte Zu-

griffsfunktionen nutzen. In diesem Bereich sind somit, abgesehen von einem neuen Algorithmus zur Bildung einer konvexen Hülle für Relativbewegungen, einer Erweiterung der Diskretisierung des Raumes für den A*-Algorithmus sowie der Möglichkeit der Bestimmung von drei Rotationswinkeln, wenige grundlegende Anpassungen zu erwarten. Der Aufbau der Gesamtarchitektur und die Verhaltensauswahl wären nicht von einer derartigen Veränderung betroffen.

6.4.2 Übertragung auf weitere Plattformen

Die bisherigen Eigenschaften und Möglichkeiten der Verhaltenssteuerung sind im Hinblick auf die allgemeine Verwendbarkeit weitestgehend abstrakt entworfen und mit einer umfangreichen Parametrisierung versehen worden. Sie sind jedoch auch ein Produkt der bisherigen Anforderungen, Beschränkungen und Erfahrungen mit den verwendeten Testplattformen. Durch die Portierung auf weitere Plattformen würden sehr wahrscheinlich weitere, bisher nicht betrachtete, Verfahren und Verhaltensaspekte entdeckt werden.

Ebenso konnten noch nicht sämtliche implementierten Verfahren, wie beispielsweise die Nutzung eines probabilistischen Weltmodells, in der Praxis angewendet werden, da diese noch nicht von den verwendeten Plattformen unterstützt werden.

Die bisher verwendeten Verhalten wurden vollständig vom Entwickler der Verhaltensarchitektur modelliert. Die Benutzung der Beschreibungssprache durch weitere, externe Personen kann dazu beitragen, das Gesamtsystem zu verbessern.

6.4.3 Modellierung hierarchischer Verhaltensstrukturen

Ein weit verbreitetes Verfahren zur Organisation einer großen Anzahl einzelner reaktiver Verhalten, die ein sehr komplexes Gesamtverhalten bilden, ist deren Anordnung in Hierarchien, wie beispielsweise in [Jäger and Christaller, 1997] oder [Behnke and Rojas, 2000] beschrieben.

Innerhalb der vorgestellten Architektur ist es bisher nicht möglich, beliebige Hierarchien von Verhalten zu modellieren. Mittels des in Abschnitt 2.2.3 beschriebenen Verfahrens der Verhaltenskombination lassen sich lediglich einfache Abhängigkeiten ausdrücken. Die Erstellung echter Hierarchien ist nur über den Weg der Instantiierung mehrerer Gesamtverhalten, die untereinander manuell verbunden werden, möglich.

Ein Veränderung der Architektur dahingehend, dass einzelne Verhalten oder Gruppen von Verhalten – zusätzlich zu den bisher verwendeten Verfahren – hierarchisch

angeordnet werden können, betrifft hauptsächlich die in Abschnitt 2.2 beschriebene Verhaltensauswahl. Die Modellierung der Objekte, Bewegungen und Aktionen bleibt vollständig unberührt.

6.5 Zusammenfassung

In dieser Arbeit wurde eine Verhaltenssteuerung für autonome mobile Roboter vorgestellt, die bisherige Ansätze zur Bewegungssteuerung und Aktionsauswahl durch Potentialfelder in einer gemeinsamen Architektur zusammenfasst. Durch die Interpretation von Feldern als konkurrierende Verhalten konnten komplexere Gesamtverhalten beschrieben werden, als es mit dem klassischen, rein auf Überlagerung basierenden Potentialfeld-Ansatz möglich gewesen wäre.

Einige der eingesetzten Verfahren wurden in ihrer Funktionalität erweitert, beispielsweise konnte das Prinzip der Aktionsbewertung von [Johannson and Saffiotti, 2002] auf Aktionssequenzen angewendet werden. Darüber hinaus wird die Architektur durch neu entwickelte Verfahren ergänzt, unter anderem ist ein, im Zusammenhang mit beliebigen Bewegungsverhalten verwendbarer, Suchalgorithmus zur Vermeidung lokaler Minima implementiert worden.

Das verwendete Modell zur Beschreibung der Umwelt sowie die darauf aufbauenden Verfahren haben einen sehr geringen Speicherbedarf. Ebenso benötigen einzelne Verhalten im Allgemeinen wenig Rechenzeit.

Die Verhaltensarchitektur ist auf zwei verschiedenen Plattformen eingesetzt worden. Mittels der abstrakten Schnittstellen war jeweils eine flexible Integration in ein bestehendes Gesamtsystem möglich.

Durch die Modellierung der Verhalten in externen XML-Dateien ist es möglich, relativ kompakte, übersichtliche und strukturierte Beschreibungen zu erstellen. Eventuelle Änderungen lassen sich, verglichen mit einer möglichen Implementierung in C++, sehr schnell durchführen. Einzelne Verhalten oder Objekte können oftmals problemlos, zum Teil auch zwischen verschiedenen Plattformen, aus anderen Beschreibungsdateien entnommen werden.

Es wurde bisher erfolgreich eine Reihe verschiedener Verhalten modelliert. Dabei konnten jedoch nicht sämtliche implementierten Verfahren in der Praxis getestet werden, da in manchen Fällen keine geeignete Testplattform zur Verfügung stand.

Letztendlich lässt sich feststellen, dass ein Großteil der Ziele dieser Diplomarbeit erreicht worden ist. Lediglich im Bezug auf die Verwendung eines probabilistischen Weltmodells stehen noch Tests auf einer geeigneten Plattform aus. Die entworfene Architektur ist, wie bereits in Abschnitt 6.4.2 erwähnt, nicht als vollständig und endgültig zu betrachten.

Anhang A

Referenz der Programmierschnittstelle

Dieses Kapitel behandelt die Einbettung der Implementierung der Verhaltensarchitektur in ein Gesamtsystem. Dazu werden die nötigen Vorbedingungen, die verwendeten Datentypen sowie die aufzurufenden Funktionen beschrieben.

A.1 Plattformen

Die gesamte Architektur ist weitestgehend plattformunabhängig in C++ programmiert worden. Bis auf die *Standard Template Library*, die im Allgemeinen Teil der Distribution eines jeden C++-Übersetzers ist, sind keine weiteren externen Bibliotheken verwendet worden.

Bisher konnte die Verhaltensarchitektur auf folgenden Plattformen übersetzt und ausgeführt werden:

Linux Die Small-Size-Roboter werden von einem zentralen Linux-Rechner gesteuert. Auf diesem wurde die Software mit dem GNU C++-Compiler 3.2 übersetzt.

Aperios Der gleiche Übersetzer ist für einen Sony-Roboter verwendet worden. Die Plattform basiert jedoch auf dem Betriebssystem Aperios zusammen mit der Entwicklungsumgebung *OPEN-R SDK* [Sony Corporation, 2003a].

Windows Zur Simulation innerhalb der RobotControl-Umgebung ist die Verhaltenssteuerung mit Microsoft Visual C++ 6.0 unter Windows übersetzt worden.

Die einzigen benötigten Anpassungen waren Anweisungen zur Behandlung von Dateien und zur Bestimmung der Systemzeit, ebenso mussten einige globale Konstanten für einen Teil der Plattformen hinzugefügt werden. Dies ist vollständig über Präprozessor-Anweisungen in der Datei `PfieldConfig.h` sowie im Parser realiert worden.

A.2 Instantiierung

Die Verhaltenssteuerung ist vollständig in einer Klasse gekapselt. Nach einer Instantiierung von `PotentialfieldComposition` kann mittels der Funktionen

```
void load(const std::string& filename)
```
und
```
void close()
```

eine Beschreibungsdatei mit einem Verhalten geöffnet, beziehungsweise wieder geschlossen werden. Das Öffnen einer solchen Datei führt zu einer automatischen Auswertung mit der anschließenden Konstruktion sämtlicher Elemente des Verhaltens.

A.3 Datentypen

Zur Interaktion mit der Klasse `PotentialfieldComposition` werden mehrere Datentypen benötigt, die Informationen über Objektzustände beziehungsweise eine Beschreibung des ausgewählten Verhaltens kapseln.

A.3.1 `PfVec`

Die Klasse `PfVec` beschreibt einen zweidimensionalen Vektor. Neben einer Menge typischer Vektoroperationen, die für interne Berechnungen verwendet werden, enthält sie die beiden Variablen `x` und `y`, die die Komponenten des Vektors repräsentieren.

A.3.2 `PfPose`

Die Pose des Roboters oder eines Objekts der Umgebung wird mittels der Klasse `PfPose` beschrieben. Sie enthält folgende Werte:

```
PfVec                pos
double               rotation
double               speed
double               probability
bool                 hasProbabilityDistribution
std::vector<PfPose>  probabilityDistribution
```

Neben Angaben zur Position, Ausrichtung und Geschwindigkeit ist zusätzlich die Beschreibung einer Wahrscheinlichkeitsverteilung möglich. Dazu muss der Variable `hasProbabilityDistribution` der Wert `true` zugeordnet werden und die einzelnen Hypothesen in die Liste `probabilityDistribution` eingetragen werden. Jeder dieser Hypothesen kann zusätzlich eine Wahrscheinlichkeit zugeordnet werden.

A.3.3 ObjectStateDescription

Die Klasse `ObjectStateDescription` enthält die Beschreibung eines Objektzustands. Sie setzt sich folgendermaßen zusammen:

```
std::string  objectName
int          objectId
PfPose       pose
bool         isActive
```

Mittels des Bezeichners `objectName` kann die Beschreibung einem gleichnamigen, bei der Modellierung angegebenen, Zustand zugeordnet werden. Es kann zusätzlich, zur einfacheren internen Verarbeitung, die Identifikationsnummer des Objektzustands verwendet werden. Diese kann durch eine in Abschnitt A.4 beschriebene Funktion bestimmt werden. Der Zustand selbst wird über eine Pose beschrieben. Abhängig vom Wert der Variable `isActive` werden sämtliche von diesem Zustand abhängigen Objekte aktiviert beziehungsweise deaktiviert.

A.3.4 PotentialfieldResult

Die Klasse `PotentialfieldResult` beschreibt das ausgewählte Verhalten zusammen mit eventuell berechneten Parametern:

```
double         value
std::string    action
std::string    subAction
PfPose         motion
bool           actionPossible
unsigned int   fieldNumber
unsigned long  timeStamp
```

Die Variable `action` enthält den Namen des ausgewählten Verhaltens; sollte mittels einer Suche nach der besten Aktionsfolge gemäß Abschnitt 5.3.2 eine Unteraktion ausgewählt worden sein, so ist deren Name in `subAction` vermerkt. Mittels der Pose `motion` werden die von einem Verhalten erzeugten Bewegungsparameter beschrieben. Konnte kein Verhalten ausgewählt werden, so hat `actionPossible` den Wert `false`, `action` den Wert `"none"`. Die weiteren Variablen `value` (Bewertung des Verhaltens), `fieldNumber` (interne Nummer des Verhaltens) sowie `timeStamp` (Zeitpunkt der Berechnung des Verhaltens) sind im Allgemeinen für die Weiterverarbeitung des Ergebnisses nicht relevant, werden aber für interne Berechnungen verwendet.

A.4 Aktualisierung von Objektzuständen

Die Informationen über den Zustand der Umwelt können über verschiedene Funktionen an die Verhaltenssteuerung weitergegeben werden:

```
void setObjectState(ObjectStateDescription& desc)

void setOwnPose(const PfPose& pose)

void setFieldActivation
     (const std::string& fieldname,bool activation)
```

Mittels `setObjectState` wird der Zustand eines Objekts verändert. Diese Funktion muss gegebenenfalls für jedes einzelne Objekt des Weltmodells aufgerufen werden, sofern es sich seit der letzten Ausführung der Verhaltenssteuerung verändert hat. Da der Roboter stets gesondert betrachtet wird, kann lediglich seine Pose durch `setOwnPose` verändert werden. Des Weiteren ist es möglich, mittels `setFieldActivation` ein gesamtes Verhalten zu aktivieren beziehungsweise zu deaktivieren.

Die Funktion `getIdFromObjectStateSymbol` dient der Bestimmung einer Identifikationsnummer eines Objektzustandes mittels dessen Namen:

```
unsigned int getIdFromObjectStateSymbol
             (const std::string& objectName)
```

A.5 Ausführung der Verhaltenssteuerung

Nachdem sämtliche Objektzustände aktualisiert worden sind, kann die Bestimmung des nächsten auszuführenden Verhaltens durchgeführt werden. Dies geschieht mittels der Funktion `execute`:

```
void execute(PotentialfieldResult& result)
```

Das Ergebnis wird in die entsprechenden Felder des per Referenz übergebenen Datentyps eingetragen.

A.6 Visualisierung

Zu Testzwecken oder zum Auffinden von Fehlern kann es nützlich sein, ein Potentialfeld oder den Verlauf der Werte der Potentialfunktion zu visualisieren. Zu diesem Zweck sind zwei Funktionen implementiert worden, die einen Bereich der modellierten Umgebung mit einem Raster in Zellen einteilen und zu diesen jeweils den Feldvektor beziehungsweise das Potential bestimmen. Diese Daten können dann für eine beliebige Visualisierung genutzt werden.

```
void getValueArray(const std::string& fieldname,
        double x1, double y1, double x2,
        double y2, int xSteps, int ySteps,
        double value[], double& max)

void getDirectionArray(const std::string& fieldname,
        double x1, double y1, double x2,
        double y2, int xSteps, int ySteps,
        PfVec directions[])
```

Dabei ist `fieldname` jeweils der Name des zu visualisierenden Verhaltens. Die Koordinatenpaare (`x1`,`y1`) und (`x2`,`y2`) spannen ein Rechteck auf, das den Grenzen des Rasters entspricht. Es erfolgt eine Aufteilung in `xSteps*ySteps` Zellen. Die berechneten Werte werden in die per Referenz übergebenen Felder eingetragen. Der dafür benötigte Speicher muss vor dem Aufruf der Funktion erzeugt worden sein. Die erzeugten Vektoren haben eine maximale Länge von Eins. Der Wertebereich der Potentiale ist $[0, \ldots,$ `max`$]$.

Durch die Funktion `getFieldNames` können die Namen aller modellierten Verhalten bestimmt werden:

```
std::vector<std::string> getFieldNames()
```

Anhang B

Referenz der XML-Konfigurationsdateien

Die Modellierung eines Verhaltens geschieht durch die Erstellung einer Beschreibungsdatei in einem speziellen auf XML basierenden Format. Dieses Kapitel erläutert die verfügbaren Beschreibungselemente und deren Anwendung.

B.1 Verwendung der Konfigurationsdateien

Der Aufbau einer Konfigurationsdatei ist mittels einer DTD namens `pfc.dtd` spezifiziert worden, die ein Teil der Implementierung ist. Jedes modellierte Verhalten muss gültig entsprechend den Regeln dieser Beschreibung sein.

Zur späteren Verwendung auf einem Roboter werden die XML-Dateien in ein spezielles Format überführt. Die dazu benötigten Transformationsanweisungen sind in XSLT spezifiziert worden und finden sich in der Datei `transform-pfc.xsl`.

Validierung und Transformation können beispielsweise mittels der in Abschnitt 2.4.3 erwähnten Programme von [Veillard, 2003] durchgeführt werden. Auf den bisherigen Plattformen konnten diese beide Schritte problemlos in den Übersetzungsprozess des Gesamtsystems eingebunden werden.

B.2 Aufbau einer Konfigurationsdatei

Der Aufbau einer Verhaltensbeschreibung orientiert sich an den in Abschnitt 2.3 beschriebenen Komponenten: Modellierung der Objekte, Benennung der Objekt-

zustände, Instantiierung der Objekte und Objektgruppen, Modellierung möglicher Relativbewegungen, Anlegen der Verhaltensteuerung.

```
<!ELEMENT potentialfields-configuration
        (object*, object-state-symbol*,
        (object-instance | instance-group)*,
        formation-object*, potentialfield-composition)>
<!ATTLIST potentialfields-configuration
        name CDATA #REQUIRED
        description CDATA #IMPLIED>
```

Die Verhaltenssteuerung (gemäß Abschnitt 2.2.2) setzt sich zusammen aus Aktions- und Bewegungsfeldern. Zusätzlich kann ein Auswahlmechanismus (Abschnitt 2.2.3) angegeben werden:

```
<!ELEMENT potentialfield-composition
        ((actionfield | motionfield)*)>
<!ATTLIST potentialfield-composition
        selection-procedure (best-of-n | successive-n-times)
                            "successive-n-times"
        n CDATA "1">
```

B.3 Modellierung von Objekten

Ein Objekt setzt sich, wie in Kapitel 3 beschrieben, aus einer Funktion, einer Feldform sowie einer geometrischen Figur zusammen. Zusätzlich ist ihm ein eindeutiger Name sowie der Typ des Feldes (attraktiv oder repulsiv) zuzuordnen. Mittels eines Schalters kann ein tangential verlaufendes Feld erzeugt werden (siehe Abschnitt 3.3.4).

```
<!ELEMENT object ((parabolic-function | linear-function |
                   no-function | asymptotic-function |
                   social-function),
                  (point-field | sector-field | shape-field),
                  (polygon | line | circle | no-geometry))>
<!ATTLIST object
        name ID #REQUIRED
        type (attractive | repulsive) #REQUIRED
        tangential-field (clockwise | counter-clockwise | none)
                         "none"
        description CDATA #IMPLIED>
```

Die möglichen Funktionen können entsprechend Abschnitt 3.2 beschrieben werden. Dabei entspricht `at-zero` dem Extremwert z, `range` der Reichweite r sowie `const-interval` der Entfernung ϵ:

```
<!ELEMENT no-function EMPTY>

<!ELEMENT linear-function EMPTY>
<!ATTLIST linear-function
        at-zero CDATA #REQUIRED
        range CDATA #REQUIRED>

<!ELEMENT parabolic-function EMPTY>
<!ATTLIST parabolic-function
        at-zero CDATA #REQUIRED
        range CDATA #REQUIRED>

<!ELEMENT asymptotic-function EMPTY>
<!ATTLIST asymptotic-function
        at-zero CDATA #REQUIRED
        range CDATA #REQUIRED
        const-interval CDATA #REQUIRED>

<!ELEMENT social-function EMPTY>
<!ATTLIST social-function
        repulsive-constant CDATA #REQUIRED
        repulsive-exponent CDATA #REQUIRED
        attractive-constant CDATA #REQUIRED
        attractive-exponent CDATA #REQUIRED
        const-interval CDATA #REQUIRED
        k CDATA "0">
```

Das von einem Objekt erzeugte Feld kann zentriert sein (`point-field`), die Figur des Objekts umgeben (`shape-field`) oder lediglich innerhalb eines Kreisausschnitts wirken (`sector-field`), wie in Abschnitt 3.3 beschrieben. Dabei ist zu beachten, dass der Öffnungswinkel eines Kreisausschnitts, ebenso wie sämtliche weiteren Winkel, in Grad angegeben wird.

```
<!ELEMENT point-field EMPTY>

<!ELEMENT shape-field EMPTY>

<!ELEMENT sector-field EMPTY>
<!ATTLIST sector-field
        opening-angle CDATA #REQUIRED
        cross-function (parabolic-function | linear-function |
                        asymptotic-function) #REQUIRED>
```

Die in Abschnitt 3.4 aufgeführten, möglichen geometrischen Figuren sind folgendermaßen definiert:

```
<!ELEMENT no-geometry EMPTY>

<!ELEMENT line (pt, pt)>
```

```
<!ATTLIST line
        intersectable (true | false) "true">

<!ELEMENT polygon (pt, pt, pt+)>
<!ATTLIST polygon
        intersectable (true | false) "true">

<!ELEMENT circle EMPTY>
<!ATTLIST circle
        radius CDATA #REQUIRED
        intersectable (true | false) "true">

<!ELEMENT pt EMPTY>
<!ATTLIST pt
        x CDATA #REQUIRED
        y CDATA #REQUIRED>
```

B.4 Objektzustände und Instanzen

Die verwendeten Objektzustände müssen bei der Modellierung angegeben und mit einem eindeutigen Bezeichner versehen werden. Einer Instanz eines Objekts wird entweder einer dieser Zustände oder eine feste Position zugeordnet (siehe Abschnitt 2.3.2). Eine Menge von Instanzen kann in einer Gruppe zusammengefasst werden.

```
<!ELEMENT object-state-symbol EMPTY>
<!ATTLIST object-state-symbol
        name ID #REQUIRED>

<!ELEMENT static-pose EMPTY>
<!ATTLIST static-pose
        x CDATA #REQUIRED
        y CDATA #REQUIRED
        rotation CDATA #REQUIRED>

<!ELEMENT dynamic-pose EMPTY>
<!ATTLIST dynamic-pose
        get-data-from IDREF #REQUIRED>

<!ELEMENT object-instance (static-pose | dynamic-pose)>
<!ATTLIST object-instance
        type IDREF #REQUIRED
        name ID #REQUIRED
        description CDATA #IMPLIED>

<!ELEMENT instance-group (include*)>
<!ATTLIST instance-group
```

```
        name ID #REQUIRED
        description CDATA #IMPLIED>
```

B.5 Relativbewegungen

Die in Abschnitt 4.2 beschriebenen Relativbewegungen werden mittels spezieller Objekte (`formation-object`) modelliert. Ein solches Objekt kann eine oder mehrere sich überlagernde Relativbewegungen enthalten:

```
<!ELEMENT formation-object (between | among |
                            relative-to | best-fit)*>
<!ATTLIST formation-object
        name ID #REQUIRED>

<!ELEMENT between (parabolic-function | linear-function |
                   asymptotic-function | social-function)>
<!ATTLIST between
        object-1 IDREF #REQUIRED
        object-2 IDREF #REQUIRED
        priority CDATA "0">

<!ELEMENT among (parabolic-function | linear-function |
                 asymptotic-function | social-function)>
<!ATTLIST among
        object-group IDREF #REQUIRED
        priority CDATA "0">

<!ELEMENT relative-to (parabolic-function | linear-function |
                       asymptotic-function | social-function)>
<!ATTLIST relative-to
        object CDATA #REQUIRED
        angle CDATA #REQUIRED
        coordinates (relative | absolute) "relative"
        priority CDATA "0">

<!ELEMENT best-fit (between | among | relative-to)*>
<!ATTLIST best-fit
        select (max-gradient | min-gradient |
                min-distance | max-priority) #REQUIRED>
```

B.6 Bewegungssteuerung

Ein Bewegungsverhalten wird durch ein entsprechendes Bewegungsfeld beschrieben (siehe Kapitel 4). Mittels einer Reihe von Attributen ist die Deaktivierung von

Freiheitsgraden (Abschnitt 4.4.2), die Glättung der erzeugten Bewegungen (Abschnitt 4.4.3) sowie die Blockierung beziehungsweise Beibehaltung des Verhaltens möglich (Abschnitt 2.2.3):

```
<!ELEMENT motionfield ((return-gradient | return-const),
                       combine-with*, avoid-local-minima?,
                       include-random-motion-generator?,
                       (include | include-group |
                       include-formation)*)>
<!ATTLIST motionfield
        name ID #REQUIRED
        description CDATA #IMPLIED
        disable-translation (true | false) "false"
        disable-rotation (true | false) "false"
        max-acceleration CDATA "-1"
        max-gradient-difference CDATA "-1"
        keep (field | result) "field"
        for-n (calls | milliseconds) "calls"
        n CDATA "0"
        block-after-selection-for-m (calls|milliseconds) "calls"
        m CDATA "0"
        keep-max-for-o (calls|milliseconds) "milliseconds"
        o CDATA "-1">
```

Folgende Aktivierungswerte (gemäß Abschnitt 4.1.4) sind möglich:

```
<!ELEMENT return-gradient EMPTY>

<!ELEMENT return-const EMPTY>
<!ATTLIST return-const
        value CDATA #REQUIRED>
```

Einem Verhalten können einzelne Objektinstanzen, Gruppen von Objekten sowie Relativbewegungen zugeordnet werden. Des Weiteren ist die Kombination mit einer beliebigen Anzahl anderer Verhalten (nach Abschnitt 2.2.3) möglich. Die Zuordnung dieser Elemente ist in gleicher Form für ein Aktionsfeld möglich und wird daher im entsprechenden Abschnitt nicht wieder aufgeführt.

```
<!ELEMENT include EMPTY>
<!ATTLIST include
        name IDREF #REQUIRED>

<!ELEMENT include-group EMPTY>
<!ATTLIST include-group
        name IDREF #REQUIRED>

<!ELEMENT include-formation EMPTY>
<!ATTLIST include-formation
        name IDREF #REQUIRED>
```

```
<!ELEMENT combine-with EMPTY>
<!ATTLIST combine-with
        name IDREF #REQUIRED>
```

Eine Wegplanung zur Vermeidung lokaler Minima (nach Abschnitt 4.3) kann durch die Einbindung eines speziellen Elements verwendet werden. Es kann zwischen der permanenten und einer Verwendung bei Bedarf (Abschnitt 4.3.4) gewählt werden, wobei letztere einen zusätzlichen Parameter (`max-gradient-for-planning`) benötigt, der die zulässige Mindestlänge des Gradienten für eine Bewegungssteuerung ohne Wegplanung beschreibt. Ein Stabilisierungselement kann in Form eines gewöhnlichen Objekts definiert werden.

```
<!ELEMENT avoid-local-minima (stabilization-element?)>
<!ATTLIST avoid-local-minima
        use (always | if-needed) #REQUIRED
        max-gradient-for-planning CDATA "-1.0"
        goal IDREF #REQUIRED
        distance-to-goal CDATA #REQUIRED
        min-expansion-radius CDATA #REQUIRED
        max-expansion-radius CDATA #REQUIRED
        min-branching-factor CDATA #REQUIRED
        max-branching-factor CDATA #REQUIRED
        end-of-near CDATA #REQUIRED
        end-of-far CDATA #REQUIRED
        standard-gradient-length CDATA "1"
        min-cache-size CDATA "1"
        max-number-of-search-nodes CDATA "-1">

<!ELEMENT stabilization-element (object)>
<!ATTLIST stabilization-element
        distance CDATA #REQUIRED>
```

Ferner kann ein Generator für Zufallsbewegungen (wie in Abschnitt 4.4.1 beschrieben) hinzugefügt werden:

```
<!ELEMENT include-random-motion-generator EMPTY>
<!ATTLIST include-random-motion-generator
        min-value CDATA #REQUIRED
        max-value CDATA #REQUIRED
        value-dx CDATA #REQUIRED
        direction-dx CDATA #REQUIRED
        change-after-n (calls | milliseconds) #REQUIRED
        n CDATA #REQUIRED>
```

B.7 Aktionsauswahl

Ein Verhalten zur Aktionsauswahl (siehe Kapitel 5) kann ähnlich wie ein Bewegungsverhalten modelliert werden. Die Einbeziehung der benötigten Zeit einer Aktion in die anschließende Bewertung ist abhängig vom Wert des zusätzlichen Attributs `consider-time`.

```
<!ELEMENT actionfield ((return-gradient | return-gain |
                        return-absolute | return-const),
                       (action | fixed-sequence |
                        find-best-sequence), combine-with*,
                       (include | include-group |
                        include-formation)*)>
<!ATTLIST actionfield
        name ID #REQUIRED
        description CDATA #IMPLIED
        consider-time (true | false) "false"
        keep (field | result) "field"
        for-n (calls | milliseconds) "calls"
        n CDATA "0"
        block-after-selection-for-m (calls|milliseconds) "calls"
        m CDATA "0"
        keep-max-for-o (calls|milliseconds) "milliseconds"
        o CDATA "-1">
```

Zusätzlich zu den bereits beschriebenen Verfahren zur Bestimmung eines Aktivierungswertes sind noch weitere, in Abschnitt 5.1.5 erläuterte, Berechnungsmodelle verfügbar:

```
<!ELEMENT return-absolute EMPTY>

<!ELEMENT return-gain EMPTY>
<!ATTLIST return-gain
        object IDREF #REQUIRED>
```

Einem Aktionsfeld kann eine einzelne Aktion, eine Aktionssequenz (Abschnitt 5.3) oder eine Menge von Aktionen zusammen mit einem Suchverfahren (Abschnitt 5.3.2) zugeordnet sein.

```
<!ELEMENT action ((((move-object, impact-area) | move-self),
                   (translation | translation-to-object |
                    translation-along-gradient | rotation |
                    rotation-to-object | rotation-to-gradient |
                    no-transformation | probability-distribution)) |
                  measure-self | measure-object)>
<!ATTLIST action
        name CDATA "No␣name"
        description CDATA "No␣description">
```

```
<!ELEMENT fixed-sequence (action+)>

<!ELEMENT find-best-sequence (action+)>
<!ATTLIST find-best-sequence
        depth CDATA #REQUIRED
        decreasing-values-only (true | false) "false">
```

Die in Abschnitt 5.1.5 beschriebenen Aktionen sind, zusammen mit dem Test zur Durchführbarkeit einer Transformation (nach Abschnitt 5.2.5) folgendermaßen definiert worden:

```
<!ELEMENT move-object EMPTY>
<!ATTLIST move-object
        name IDREF #REQUIRED
        join-action (true | false) "false">

<!ELEMENT impact-area (polygon)>

<!ELEMENT measure-object EMPTY>
<!ATTLIST measure-object
        name IDREF #REQUIRED>

<!ELEMENT move-self EMPTY>

<!ELEMENT measure-self EMPTY>
```

Für die einzelnen Transformationen (Abschnitt 5.2) gelten folgende Modellierungsvorschriften:

```
<!ELEMENT translation EMPTY>
<!ATTLIST translation
        x CDATA #REQUIRED
        y CDATA #REQUIRED
        time CDATA #REQUIRED>

<!ELEMENT translation-to-object EMPTY>
<!ATTLIST translation-to-object
        object IDREF #REQUIRED
        speed CDATA #REQUIRED>

<!ELEMENT translation-along-gradient EMPTY>
<!ATTLIST translation-along-gradient
        step-length CDATA #REQUIRED
        max-gradient-deviation CDATA #REQUIRED
        max-length CDATA #REQUIRED
        speed CDATA #REQUIRED>

<!ELEMENT rotation EMPTY>
```

```
<!ATTLIST rotation
        angle CDATA #REQUIRED
        time CDATA #REQUIRED>

<!ELEMENT rotation-to-object EMPTY>
<!ATTLIST rotation-to-object
        object IDREF #REQUIRED
        speed CDATA #REQUIRED>

<!ELEMENT rotation-to-gradient EMPTY>
<!ATTLIST rotation-to-gradient
        speed CDATA #REQUIRED>

<!ELEMENT no-transformation EMPTY>
<!ATTLIST no-transformation
        time CDATA "1">

<!ELEMENT probability-distribution (probability+)>
<!ELEMENT probability (translation | rotation)>
<!ATTLIST probability
        value CDATA #REQUIRED>
```

Literaturverzeichnis

[Arkin, 1989] Arkin, R. C. (1989). Motor Schema-Based Mobile Robot Navigation. *The International Journal of Robotics Research*, 8(4):92–112.

[Arkin, 1998] Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press, Cambridge, Massachusetts, USA.

[Balch and Arkin, 1993] Balch, T. and Arkin, R. C. (1993). Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 678–685, Atlanta, Georgia, USA.

[Balch and Arkin, 1999] Balch, T. and Arkin, R. C. (1999). Behavior-based Formation Control for Multi-robot Teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939.

[Ball and Wyeth, 2004] Ball, D. and Wyeth, G. (2004). Multi-Robot Control in Highly Dynamic, Competitive Environments. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Behnke, 2004] Behnke, S. (2004). Local Multiresolution Path Planning. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Behnke and Rojas, 2000] Behnke, S. and Rojas, R. (2000). A hierarchy of reactive behaviors handles complexity. In *Proceedings of Balancing Reactivity and Social Deliberation in Multi-Agent Systems, a Workshop at ECAI 2000, the 14th European Conference on Artificial Intelligence*, Berlin.

[Bronstein et al., 1999] Bronstein, I., Semendjajew, K., Musiol, G., and Mühlig, H. (1999). *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main.

[Brooks, 1986] Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

[Brooks, 1991] Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence Journal*, 47:139–159.

[Browning, B., 2003] Browning, B. (2003). Official RoboCup Small Size Leage. `http://www-2.cs.cmu.edu/~brettb/robocup/`.

[Bruce and Veloso, 2002] Bruce, J. and Veloso, M. (October 2002). Real-Time Randomized Path Planning for Robot Navigation. In *Proceedings of IROS-2002*.

[Damas et al., 2003] Damas, B., Custódio, L., and Lima, P. (2003). A Modified Potential Fields Method for Robot Navigation Applied to Dribbling in Robotic Soccer. In Kaminka, G. A., Lima, P., and Rojas, R., editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Artificial Intelligence*. Springer.

[D'Andrea et al., 2001] D'Andrea, R., Kalmar-Nagy, T., Ganguly, P., and Babish, M. (2001). The Cornell Robocup Team. In Stone, P., Balch, T., and Kraetschmar, G., editors, *RoboCup Symposium 2000, Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*, pages 41–51. Springer.

[Fox et al., 1999] Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proc. of the National Conference on Artificial Intelligence*.

[Hart et al., 1968] Hart, P., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107.

[Hoppe et al., 2004] Hoppe, A., Kurlbaum, J., Mohrmann, B., Poloczek, M., Reinecke, D., Röfer, T., and Visser, U. (2004). Bremen Small Multi-Agent Robot Team (B-Smart) Team Description for RoboCup 2003. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Johannson and Saffiotti, 2002] Johannson, S. J. and Saffiotti, A. (2002). Using the Electric Field Approach in the RoboCup Domain. In Birk, A., Coradeschi, S., and Tadokoro, S., editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*. Springer.

[Jäger and Christaller, 1997] Jäger, H. and Christaller, T. (1997). Dual Dynamics: Designing Behavior Systems for Autonomous Robots. In Fujimura, S. and Sugisaka, M., editors, *Proceedings of the International Symposium on Artificial Life and Robotics (AROB '97)*, pages 76–79, Beppu, Japan.

[Kambhampati and Davis, 1986] Kambhampati, S. and Davis, L. (1986). Multiresolution Path Planning for Mobile Robots. *IEEE Journal of Robotics and Automation*, RA-2(3):135–145.

[Khatib, 1986] Khatib, O. (1986). Real-time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5(1):90–98.

[Kiat, 2004] Kiat, N. B. (2004). LuckyStar 2003. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Kitano et al., 1995] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1995). RoboCup: The Robot World Cup Initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*.

[Koren and Borenstein, 1991] Koren, Y. and Borenstein, J. (1991). Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1404, Sacramento, California, USA.

[Latombe, 1991] Latombe, J. C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston, USA.

[Lin and Chuang, 2003] Lin, C.-C. and Chuang, J.-H. (2003). Potential-Based Path Planning for Robot Manipulators in 3-D Workspace. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan.

[Lötzsch et al., 2004] Lötzsch, M., Bach, J., Burkhard, H.-D., and Jüngel, M. (2004). Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Maes, 1989] Maes, P. (1989). How To Do the Right Thing. *Connection Science Journal*, 1(3):291–323.

[Meyer and Adolph, 2003] Meyer, J. and Adolph, R. (2003). Decision-making and Tactical Behavior with Potential Fields. In Kaminka, G. A., Lima, P., and Rojas, R., editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Artificial Intelligence*. Springer.

[Papula, 1999] Papula, L. (1999). *Mathematik für Ingenieure und Naturwissenschaftler Band 3*. Vieweg, Braunschweig / Wiesbaden.

[Reif and Wang, 1995] Reif, J. H. and Wang, H. (1995). Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots. In Goldberg, K., Halperin, D., Latombe, J.-C., and Wilson, R., editors, *The Algorithmic Foundations of Robotics*, pages 331 – 345. A. K. Peters, Boston, MA.

[RoboCup Federation, 2003] RoboCup Federation (2003). RoboCup-Homepage. `http://www.robocup.org`.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Inc.

[Röfer, 2002] Röfer, T. (2002). SimRobot - 3D-Robotiksimulator. `www.tzi.de/simrobot`.

[Röfer, 2003] Röfer, T. (2003). An Architecture for a National RoboCup Team. In Kaminka, G. A., Lima, P., and Rojas, R., editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Artificial Intelligence*. Springer.

[Röfer et al., 2003] Röfer, T., Brunn, R., Burkhard, H.-D., Dahm, I., Düffert, U., Engel, K., Göhring, D., Hoffmann, J., Jüngel, M., Kallnik, M., Kunz, M., Lötzsch, M., Osterhues, A., Petters, S., Risler, M., Schumann, C., Stelzer, M., von Stryk, O., Wachter, M., and Ziegler, J. (2003). GermanTeam RoboCup 2003. `www.robocup.de/germanteam`.

[Röfer et al., 2004] Röfer, T., Dahm, I., Düffert, U., Hoffmann, J., Jüngel, M., Kallnik, M., Lötzsch, M., Risler, M., Stelzer, M., and Ziegler, J. (2004). GermanTeam 2003. In Browning, B., Polani, D., Bonarini, A., and Yoshida, K., editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer. Im Erscheinen.

[Schmitt et al., 2002] Schmitt, T., Beetz, M., Hanek, R., and Buck, S. (2002). Watch their Moves: Applying Probabilistic Multiple Object Tracking to Autonomous Robot Soccer. In *The Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada.

[Schulz et al., 2001] Schulz, D., Burgard, W., Fox, D., and Cremers, A. (2001). Tracking Multiple Moving Objects with a Mobile Robot. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Sedgewick, 1992] Sedgewick, R. (1992). *Algorithmen*. Addison-Wesley, Bonn.

[Sony Corporation, 2003a] Sony Corporation (2003a). OPEN-R SDK. `http://openr.aibo.com`.

[Sony Corporation, 2003b] Sony Corporation (2003b). RoboCup Legged Robot League 2003. `http://www.openr.org/robocup/index.html`.

[Vail and Veloso, 2003] Vail, D. and Veloso, M. (2003). Multi-Robot Dynamic Role Assignment and Coordination Through Shared Potential Fields. In Schultz, A., Parker, L., and Schneider, F., editors, *Multi-Robot Systems*. Kluwer Academic Publishers.

[Veillard, 2003] Veillard, D. (2003). The XSLT C library for Gnome. `http://xmlsoft.org`.

[Weigel et al., 2002] Weigel, T., Gutmann, J.-S., Dietl, M., Kleiner, A., and Nebel, B. (2002). CS-Freiburg: Coordinating Robots for Successful Soccer Playing. *IEEE Transactions on Robotics and Automation*, 18(5):685–699.

[World Wide Web Consortium, 1999] World Wide Web Consortium (1999). XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`.

[World Wide Web Consortium, 2000] World Wide Web Consortium (2000). Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3.org/TR/REC-xml`.