# Roboter-Fußball: Selbstlokalisierung, Weltmodellierung, Pfadplanung und verhaltensbasierte Kontrolle

Diplomarbeit an der Universität Freiburg

Fakultät für Angewandte Wissenschaften

vorgelegt von:

**Thilo Weigel**

Betreuer: Prof. Dr. B. Nebel

1998

**Zusammenfassung**

Roboterfußball stellt einen herausfordernden Forschungsgegenstand dar, da Probleme auf den Gebieten Robotik, Künstliche Intelligenz, Multiagentensysteme und Bildverarbeitung gelöst werden müssen, um ein erfolgreiches Spiel zu gewährleisten. Da die hohe Relevanz hierbei behandelter Forschungsfragen der Wissenschaft auf einer Vielzahl von Gebieten neue Impulse gibt, wurde das *CS-Freiburg-RoboCup-Projekt* ins Leben gerufen, in dessen Rahmen eine Mannschaft für die Teilnahme an der Roboter-Fußballweltmeisterschaft – RoboCup'98 – entwickelt wurde. Diese Arbeit gibt eine Übersicht über die Gesamtarchitektur des Teams und beschreibt deren Schlüsselkomponenten ausführlich. Es wird gezeigt, wie aufgrund von Laserscannerdaten eine gleichermaßen zuverlässige und exakte globale Selbstlokalisierung auf dem Spielfeld durchgeführt wird. Aus den gleichen Daten werden Informationen über Objekte extrahiert und in eine Weltmodellrepräsentation überführt. Auf dieser Grundlage werden Pfadplanung und verhaltensbasierte Kontrollmechanismen eingesetzt, um bezüglich variierender Strategien kooperatives Mannschaftsspiel zu ermöglichen. Das vorliegende Gesamtsystem stellt eine überaus erfolgreiche Roboterfußballmannschaft dar, die bei offiziellen Spielen bisher ungeschlagen blieb und sowohl den Titel des Deutschen Meisters als auch den Weltmeistertitel errang.

**Danksagung**

Für die hervorragende Unterstützung meiner Arbeit möchte ich mich bei allen Beteiligten des *CS-Freiburg-RoboCup*-Projekts bedanken. Besonderer Dank gilt Steffen Gutmann: er stellte die *Scanstudio*-Bibliothek zur Verfügung, verzagte auch nicht nach nächtelanger gemeinsamer Softwareentwicklung und war als Betreuer und Projektleiter stets eine verläßliche Stütze. Desweiteren danke ich Augustinus Topor für die Entwicklung der Ballerkennung und dessen Geduld bei deren Kalibrierung. Dank gilt weiterhin Immanuel Herrmann für die Konstruktion der Schußvorrichtung, Wolfgang Hatzack für die Entwicklung der globalen Sensorintegration, Frank Rittinger für die Implementierung der graphischen Benutzerschnittstelle und Bruno Welsch für die technische Unterstützung. Ich bedanke mich bei Professor Nebel für die Unterstützung des Teams und die Betreuung meiner Arbeit.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Standardprobleme in der Wissenschaft dienen häufig als Plattform für wissenschaftlichen Austausch und den Vergleich verschiedener Ansätze und Methoden bezüglich des gleichen Forschungsgegenstands. Idealerweise stellen sie für Forscher eine herausfordernde Aufgabe dar, deren Bearbeitung Innovation vorantreibt und mit deren Lösung ein signifikanter technologischer Fortschritt einhergeht.

Das Schachspiel stellt ein prominentes Beispiel für ein solches Standardproblem innerhalb des Forschungsbereichs der *Künstlichen Intelligenz (KI)* dar. Durch die Attraktivität des Forschungsgegenstands, konzentrierte Forscherbemühungen und schnelle Fortschritte assoziierte die Öffentlichkeit bald das Spielniveau eines Schachcomputers mit dem Leistungsvermögen von allgemeinen KI-Anwendungen.

Ein noch sehr junges, aber dennoch schnell an Popularität gewinnendes Standardproblem im Bereich autonomer mobiler Roboter und Künstlicher Intelligenz hat seinen Ursprung in der Initiative einer Gruppe japanischer Forscher. Sie gründeten die *Robot World Cup Initiative (RoboCup)* und schlugen Roboterfußball als neue Herausforderung für eine Vielzahl von Forschungsbereichen in KI und Robotik vor [Kitano *et al.*, 1997]:

> The Robot World Cup Initiative (RoboCup) is an attempt to foster AI and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined.

Als Ziel der KI- und Robotikforschung wird oft die Entwicklung intelligenter, komplexe Verhalten zeigender Systeme gesehen, die gegebene Aufgaben durch Interaktion mit der sich dynamisch verändernden, realen Welt erfüllen [Asada *et al.*, 1997]. RoboCup wurde ins Leben gerufen, um bisherige Grundlagenforschung verschiedener Gebiete zu integrieren und die Forschung in Richtung dieses Ziels voranzutreiben.

Obwohl Roboterfußball ein Problem in einer sehr einfach strukturierten und limitierten Umgebung darstellt, entspricht der Problemtyp in seiner Komplexität doch Aufgaben der realen Welt, und es ist zu erwarten, daß im Roboterfußball gewinnbringend eingesetzte Techniken auch in anderen Bereichen erfolgreiche Verwendung finden können. Neben der Attraktivität für Publikum und Medien und der hohen Relevanz der involvierten Forschungsfragen motiviert die Komplexität und Charakteristik der Aufgabe die interdisziplinäre Zusammenarbeit bisher isoliert voneinander arbeitender Forschungsbereiche. Insbesondere müssen Fragen bezüglich folgender Forschungsgegenstände beantwortet werden [Kitano *et al.*, 1997]:

- *Agenten-Architektur:* Wie muß die Hardware eines Roboters beschaffen sein, welche Sensoren und Effektoren werden benötigt? Was vermögen reaktive Ansätze zu leisten? Können im Gegensatz dazu deliberative, auf explizite Weltmodellierung beruhende Ansätze den Echtzeitanforderungen und der Dynamik der Welt des Roboterfußballs gerecht werden? Wie können reaktive und deliberative Ansätze kombiniert werden?

- *Sensorfusion:* Welches sind wichtige Informationen über die Umwelt und welche Sensoren vermögen diese zu erfassen? Wie lassen sich Informationen verschiedener Sensoren bezüglich der gleichen Beobachtung fusionieren?

- *Lernen:* Lassen sich Lernverfahren zum Erlernen komplexer Verhaltensmuster und Strategien einsetzen? Welche Lernverfahren eignen sich hierzu am besten?

- *Multi-Agenten-Systeme:* Wie kann Kommunikation für Multi-Agenten-Kooperation genutzt werden? Welche Koordinationsmöglichkeiten existieren, um mehrere Agenten zur Erreichung eines gemeinsamen Ziels einzusetzen?

Sowohl die getrennte Bearbeitung jedes einzelnen Gebiets, als auch die Integration aller entwickelten Techniken in ein Gesamtsystem stellen eine Herausforderung für die Forschergemeinschaft dar, der nur in langfristigen Projekten begegnet werden kann. Um hierbei ein möglichst großes Spektrum an Schwerpunktlegungen zu erlauben, wurden verschiedene RoboCup-Ligen geschaffen, die durch ihre jeweilige Charakteristik die vorrangige Behandlung unterschiedlicher Forschungsgegenstände motivieren. Zum gegenwärtigen Zeitpunkt umfaßt RoboCup unter anderem folgende vier Klassen:

- Die *Simulationsklasse*, in der pro Mannschaft 11 virtuelle Spieler auf einem virtuellen Feld spielen. Jedem Spieler ist nur die ihn unmittelbar umgebende virtuelle Umgebung zugänglich. Spieler können beliebige Textnachrichten versenden, die von allen Spielern in der näheren Umgebung empfangen werden.

- Die *Klasse der Roboter kleiner Größe*, in der pro Mannschaft 5 Roboter mit einer Grundfläche von bis zu 180cm$^2$ auf einem Feld der Größe einer Tischtennisplatte spielen. Jedem Spieler sind Informationen einer das gesamte Spielfeld überblickenden Kamera zugänglich. Spieler können untereinander und mit einem Zentralrechner per Funk kommunizieren.
- Die *Klasse der Roboter mittlerer Größe*, in der pro Mannschaft 5 Roboter mit einem Durchmesser von maximal 50cm auf einem Feld von ca. 8.2m$\times$4.5m spielen. Jedem Spieler sind nur die Daten der eigenen Sensoren zugänglich, jedoch können über Funk beliebige Informationen ausgetauscht werden.
- Die *Expertenklasse*, in der Roboter spezielle Fähigkeiten demonstrieren.

Erstmals wurde 1997 eine Roboter-Fußballweltmeisterschaft – RoboCup'97 – als Teil der internationalen KI-Tagung *International Joint Conference on Artificial Intelligence* (IJCAI) in Nagoya, Japan, ausgetragen. Während 1997 in der Klasse der mittleren Roboter nur fünf Mannschaften teilnahmen, waren anläßlich des RoboCup'98 in Paris bereits 16 Mannschaften aus Japan, Australien, USA, Kolumbien, Portugal, Italien, Frankreich und Deutschland am Start. Zusammen mit dem großen Medieninteresse dokumentiert dies den rasanten Popularitätszuwachs und damit die Akzeptanz von RoboCup als neues Standardproblem für KI und Robotik.

Thema dieser Arbeit ist die Beschreibung des *CS-Freiburg-RoboCup-Teams*, das in einer Projektgruppe der Abteilung für Künstliche Intelligenz an der Fakultät für angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg entwickelt wurde. Das Team besteht aus fünf Robotern, das in der Klasse der Roboter mittlerer Größe bei RoboCup'98 in Paris antrat, wo es mit einem Unentschieden im ersten Spiel und fünf Siegen in Folge unangefochten den Weltmeistertitel errang.

## 1.1 Voraussetzungen, Zielsetzungen

Ausgehend von vorhandener Hardware für Forschung und Lehre und der in Praktika gewonnenen Grundlagen wurde in der Projektgruppe zunächst ein Konzept für eine flexible, robuste und leicht erweiterbare Gesamtarchitektur entworfen. Die hierzu nötige Hardware wurde zusammengestellt und die Softwarearchitektur in die Schlüsselkomponenten *Selbstlokalisierung*, *Weltmodellierung*, *Pfadplanung*, *Kommunikation*, *Sensorfusion* und *verhaltensbasierte Kontrolle* gegliedert.

Die Fertigstellung all dieser Module bei Beachtung gegenseitiger Abhängigkeiten und die anschließende Integration in ein funktionierendes Gesamtsystem wurde als Ziel definiert. Besonders durch den Einsatz exakter Selbstlokalisierungs- und Weltmodellierungsverfahren sollte das bei RoboCup'97 beobachtete Spielniveau deutlich angehoben werden.

## 1.2 Aufbau der Arbeit

Im folgenden Kapitel 2 wird zunächst anhand der Betrachtung von Schlüsselkomponenten ein Überblick über die Ansätze der Teilnehmer bei RoboCup'97 und RoboCup'98 sowie den Ansatz des *CS-Freiburg-RoboCup-Teams* gegeben. Dieser Ansatz wird anschließend in Kapitel 3 bezüglich Hardwarekomponenten und allgemeiner Architektur konkretisiert.

In den folgenden Kapiteln 4 – 8 werden Verfahren beschrieben, an deren Entwicklung der Autor maßgeblich beteiligt war. Kapitel 4 stellt das Verfahren zur Selbstlokalisierung vor, das eine absolute Positionsbestimmung auf dem Spielfeld erlaubt. In Kapitel 5 wird beschrieben, wie aufgrund der Sensoreindrücke eine interne symbolische Repräsentation – ein Weltmodell – der Umgebung erstellt wird. Das Weltmodell dient als Basis für den in Kapitel 6 besprochenen Pfadplanalgorithmus, der für die Navigation auf dem Spielfeld entwickelt wurde und kollisionsfreie Pfade zwischen der aktuellen Roboterposition und einem beliebigen Zielpunkt liefert. Nachdem in Kapitel 7 beschrieben wurde, welche grundlegenden Fertigkeiten für die Roboter implementiert wurden, wird in Kapitel 8 gezeigt, wie diese jeweils der aktuellen Spielsituation entsprechend, gemäß der beschriebenen Taktik für komplexe Spielzüge kombiniert werden. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 9 ab.

# Kapitel 2

# Ansätze zum Roboterfußball

Dieses Kapitel gibt einen Überblick über die Ansätze der Teams, die bei RoboCup'97 und RoboCup'98 teilnahmen. Zunächst wird auf die verschiedenen Forschungsschwerpunkte eingegangen, die sich aus den unterschiedlichen Voraussetzungen für die Spielklassen ergeben. Danach werden anhand der Betrachtung von Schlüsselkomponenten Unterschiede und Gemeinsamkeiten in den Designs der Teams der Klasse mittlerer Roboter vorgestellt. Zum Abschluß wird der eigene Ansatz bezüglich der besprochenen Schlüsselkomponenten eingeordnet.

## 2.1 Forschungsschwerpunkte

Ein allgemeines Problem in der Robotik betrifft die Modellierung von Unsicherheit und Ungenauigkeit von Sensorinformationen. Obwohl in der Simulatorklasse versucht wird, diese Unsicherheit zu simulieren, indem ein Spieler durch den Server nur unregelmäßig und mit nur ungenauen Information versorgt wird, ist das Problem für real existierende Roboter durch die Unabwägbarkeiten in der Interaktion mit der realen Welt bedeutend schwieriger.

Weitaus komplexer als die Beschaffung von Sensoreindrücken ist jedoch die Interpretation derselben. Die Problematik, anhand eines Kamerabildes Mit- und Gegenspieler zu erkennen sowie die Ball- und die eigene Position zu bestimmen, stellt beispielsweise eine enorme Herausforderung an die Bildverarbeitungssysteme der Teams der Klassen realer Roboter dar. Da den Spielern in der Simulatorklasse eindeutige Informationen über Objekte in der näheren Umgebung zur Verfügung stehen, entfällt für diese das Interpretationsproblem.

Naturgemäß nehmen Hardwarefragen einen großen Anteil des Entwicklungsaufwands der Klassen realer Roboter in Anspruch. Diese umfassen zum einen Probleme mechanischer und elektronischer Art und zum anderen Probleme bezüglich Roboterkontrolle auf Motorsteuerungsebene.

Lediglich die eingeschränkte Möglichkeit zur Kommunikation der Spieler untereinander stellt eine Erschwernis für die Simulatorklasse gegenüber den anderen Klassen dar. Da aber für die Robotik typische Probleme hinsichtlich Sensorik und Steuerung entfallen, besteht für die Entwickler virtueller Teams bedeutend mehr Spielraum, abstrahierende Themen der Künstlichen Intelligenz anhand des Roboterfußballs zu erproben. Diese umfassen unter anderem Subsumptions- und BDI-Architekturen [Burkhard *et al.*, 1997], Multiagentensysteme [Stone und Veloso, 1997], Lernverfahren [Stone und Veloso, 1997; Andou, 1997] und Genetisches Programmieren [Luke *et al.*, 1997].

Eine große Erleichterung auf dem Gebiet der Sensorik stellt für die Spieler in der Klasse der kleinen Roboter die Möglichkeit dar, auf Daten einer das gesamte Spielfeld überblickenden Kamera zuzugreifen. Hierdurch wird die Bestimmung absoluter Positionen von Objekten auf dem Spielfeld und somit das Problem der angemessenen Interpretation von Sensoreindrücken entscheidend vereinfacht und den Teams mehr Spielraum für Forschungsfragen bezüglich Robotersteuerung, Lernen und Pfadplanen gegeben [Nakamura, 1997; Oller *et al.*, 1997].

Das *CMUnited-Team* konnte bei RoboCup'98 eindrucksvoll demonstrieren, wie für die Simulatorliga entwickelte Multi-Agenten-Verhalten auch erfolgreich in der Klasse der kleinen Roboter eingesetzt werden können [Stone und Veloso, 1997; Veloso *et al.*, 1997]. Mit dem Erringen des Weltmeistertitels in beiden Klassen zeigten sie, wie auf der Basis robuster und zuverlässiger Interpretation von Sensorinformationen bereits in der Klasse der kleinen Roboter komplexe KI-Konzepte erfolgreiche Anwendung finden können.

Von diesem Ziel sind die Teams der Klasse der Roboter mittlerer Größe noch weit entfernt. Hier entspricht die Komplexität des Problems bezüglich Unsicherheit bei Wahrnehmung und Steuerung denen allgemeiner Probleme im Bereich der autonomen, mobilen Robotik. In dem noch sehr frühen Forschungsstadium lagen für diese Klasse daher die Schwerpunkte neben der Entwicklung und Zusammenstellung robuster Hardware auf der Entwicklung grundlegender Spielerverhalten und Techniken zur Erfassung der Umgebung.

Im folgenden wird für die Klasse der mittleren Roboter anhand charakteristischer Merkmale auf Unterschiede in deren Teamdesigns eingegangen.

## 2.2 Hardware

**Roboterplattform**

Bereits bei der Auswahl der Roboterplattform wurden sehr unterschiedliche Wege gegangen. Während einige Teams eigene Robotersysteme konstruierten, verwendete die Mehrheit im Handel erhältliche Roboter, wobei sich die Roboter *Pioneer 1* [Kraetzschmar *et al.*, 1998; Plagge *et al.*, 1998; Klupsch *et al.*, 1998; Nardi *et al.*, 1998; Werger *et al.*, 1997] und *Pioneer AT* [Plagge *et al.*, 1998; Werger *et al.*, 1997] der Firma *ActivMedia* großer Beliebtheit erfreuten. Desweiteren wurden *B14*-Roboter der Firma *RWI* [Werger *et al.*, 1997] und *SuperScout*-Roboter der Firma *Nomad* verwendet [Cantor *et al.*, 1998]. Bei einigen Teams diente als Basis für Eigenentwicklungen das Modellautofahrgestell *Black Beast* der Firma *Nikkoh* [Suzuki *et al.*, 1998; Shen *et al.*, 1998; Nakamura *et al.*, 1998]. Andere entwickelten von Grund auf eigene Systeme [Aparício *et al.*, 1998; Kuth *et al.*, 1998; Yokota *et al.*, 1998; Bui *et al.*, 1998]. Zwei Eigenkonstruktionen zeichneten sich durch besondere Kreativität bei der Fahrzeugsteuerung aus: ein omnidirektionaler Antrieb erlaubte es, den Roboter jederzeit in jede beliebige Richtung zu bewegen [Yokota *et al.*, 1998; Price *et al.*, 1997].

**Rechner**

Auch bei den verwendeten Rechnern zur Steuerung wurden verschiedene Wege gegangen. Einige Teams stellten eigene Mini-PCs zusammen, die auf dem Roboter integriert wurden [Aparício *et al.*, 1998; Werger *et al.*, 1997; Kraetzschmar *et al.*, 1998; Shen *et al.*, 1998; Yokota *et al.*, 1998; Cantor *et al.*, 1998; Plagge *et al.*, 1998; Bui *et al.*, 1998; Klupsch *et al.*, 1998; Nardi *et al.*, 1998], andere verwendeten Notebooks der Firma *Toshiba* [Nakamura *et al.*, 1998; Kuth *et al.*, 1998]. Nur wenige Teams verließen sich auf eine Funkverbindung zwischen Spielern und einem Remote-Computer zu Steuerung der Spieler [Suzuki *et al.*, 1998; Plagge *et al.*, 1998].

**Kommunikation**

Funk wurde von vielen Teams für die Kommunikation der Spieler untereinander und zum Beginnen und Beenden des Spiels genutzt. Häufig kamen hierzu Radio-Ethernets zum Einsatz [Aparício *et al.*, 1998; Kuth *et al.*, 1998; Yokota *et al.*, 1998; Plagge *et al.*, 1998; Klupsch *et al.*, 1998; Nardi *et al.*, 1998].

**Sensoren**

Den zentralen Sensor stellte bei allen gegnerischen Teams eine *Farb-CCD-Kamera* dar, deren Daten als Grundlage ihrer Objekterkennungs- und Selbstlokalisierungsmethoden dienten, wobei die Selbstlokalisierung in den meisten Fällen durch die *Odometrie* unterstützt wurde. Zur Hinderniserkennung im Nahbereich wurden *Ultraschallsensoren* [Werger *et al.*, 1997; Kraetzschmar *et al.*, 1998; Cantor *et al.*, 1998; Plagge *et al.*, 1998; Klupsch *et al.*, 1998; Nardi *et al.*, 1998] verwendet. Für besondere Zwecke statteten einige Teams ihre Roboter mit *Touch-Sensoren* [Aparício *et al.*, 1998; Nakamura *et al.*, 1998; Kuth *et al.*, 1998], *Sound-Sensoren* [Aparício *et al.*, 1998] oder *Infrarot-Abstandsensoren* [Aparício *et al.*, 1998] aus.

**Effektoren**

Als Effektoren zur Ballmanipulation brachten mehrere Teams an ihren Robotern seitlich vorstehende *Führungsfedern* an [Kraetzschmar *et al.*, 1998; Plagge *et al.*, 1998] und entwickelten spezielle *Schußmechanismen* [Kraetzschmar *et al.*, 1998; Yokota *et al.*, 1998; Plagge *et al.*, 1998].

## 2.3 Software

**Agentenarchitektur**

Neben Unterschieden in der Hardwarearchitektur unterschieden sich die Ansätze der einzelnen Teams auf Softwareseite durch unterschiedliche Gewichtung von *reaktiven* und *deliberativen* Komponenten. Einige Teams verzichteten auf jegliche interne, symbolische Repräsentation der Umgebung und basierten die Handlungskontrolle gemäß dem Brookschen Leitspruch „The world is its own best model“ [Brooks, 1991] stets auf die aktuell verfügbaren Sensoreindrücke. Das Ullanta-Team stellt für diese reaktiven Ansätze den wohl typischsten Vertreter dar [Werger *et al.*, 1997]. Während die meisten Teams reaktive und deliberative Komponenten kombinierten, gründeten nur wenige Architekturen die Aktionsauswahl auf expliziten Weltmodellen. Das *Dreamteam* – einer der letztjährigen Weltmeister – vertraute beispielsweise auf einen solchen Ansatz [Shen *et al.*, 1998]. Bei vielen Teams blieb die Frage offen, wie Hindernisse beziehungsweise Kollisionen mit anderen Spielern vermieden werden sollten. Kein Team schien explizit kollisionsfreie Pfade zu planen, und nur wenige Teams schienen eine Hindernisvermeidung auf reaktiver Basis implementiert zu haben.

**Sensorfusion**

Manche Teams verwendeten Sensorfusion, um genauere Informationen mit höherer Verläßlichkeit zu erhalten. Hierbei werden Messungen verschiedener Sensoren bezüglich der gleichen Beobachtungen derart fusioniert, daß der geschätzte Meßfehler kleiner ist als jeder der eigentlichen Messungen. Häufig wird dieses Verfahren eingesetzt, um die eigene Positionsschätzung der Odometrie durch aus der Bildverarbeitung gewonnenen Daten zu verbessern.

**Lernverfahren**

Für die *Osaka Trackies*, die sich bei RoboCup'97 mit dem *Dreamteam* den Weltmeistertitel teilten, bildeten Lernverfahren einen Schwerpunkt in ihrer Entwicklungsarbeit. Sie verwendeten Reinforcement-Lernen, um spezielle Spielerverhalten zu erlernen und zu optimieren [Suzuki *et al.*, 1997; Suzuki *et al.*, 1998].

**Multi-Agenten-Systeme**

Auch zu der Thematik Multi-Agenten-Systeme wurde bisher in der Klasse der mittleren Roboter nur von sehr wenigen Teams ein Beitrag geleistet. Hervorzuheben ist das *Uttori-Team*, das explizite *Kommunikation* der Spieler untereinander vorsieht, um ein einer globalen Taktik entsprechendes kooperatives Spiel zu erreichen [Yokota *et al.*, 1998].

## 2.4 Der Ansatz des CS-Freiburg

Das *CS-Freiburg-RoboCup-Team* besteht aus fünf Pioneer 1 Robotern, die jeweils um ein Notebook mit Radio-Ethernet-Verbindung, einen Laserscanner, eine Kamera sowie einen Schußmechanismus erweitert wurden.

Grundlage des Teams sind zuverlässige und präzise Selbstlokalisierungs- und Objekterkennungstechniken, die auf den Daten des Laserscanners basieren. Zur Selbstlokalisierung werden Odometrie- und Laserscannerdaten, zur Ballerkennung Kamera- und Ultraschalldaten fusioniert.

Alle Informationen über die Umwelt werden von den Spielern in einem lokalen Weltmodell gespeichert, das als Basis für die Navigation auf dem Spielfeld dient. Dabei werden explizit kollisionsfreie Pfade geplant und unter konstantem Neuplanen verfolgt. Die durch das Weltmodell gegebene Beschreibung der Spielsituation dient auch als Basis für die Auswahl der aktuell auszuführenden Aktion eines Spielers. Der Ansatz des *CS-Freiburg* ist somit stark deliberativ und modellbasiert ausgerichtet.

Optional kann Kommunikation zum Veranlassen des Spielbeginns bzw. -endes sowie zum Austausch von Informationen innerhalb des Teams eingesetzt werden: Zum einen können die lokalen Weltmodelle zu einem Zentralrechner gesendet werden, der diese zu einem globalen Weltmodell fusioniert und wiederum allen Spielern zugänglich macht. Zum anderen können über den Zentralrechner Informationen versendet werden, die in der Lage sind, kooperatives Teamverhalten zu steuern.

# Kapitel 3

# Architektur

Gegenstand dieses Kapitels ist die Beschreibung der Architektur des *CS-Freiburg-RoboCup-Teams*. Da der Schwerpunkt der Projektgruppe auf der Entwicklung und der Erprobung von Softwarekonzepten liegt, wurden bei der Zusammenstellung der Hardware hauptsächlich handelsübliche Komponenten verwendet. Diese werden im folgenden Abschnitt vorgestellt. Im Anschluß wird dann zunächst die Architektur des Teams und zum Abschluß die Architektur der Spieler vorgestellt.

## 3.1 Hardwarekomponenten

Als Roboterplattform wird der von Kurt Konolige entwickelte und von *ActivMedia* vertriebene *Pioneer 1* Roboter verwendet. Für die lokale Informationsverarbeitung wurde jeder Roboter mit einem *Toshiba* Notebook *Libretto 70CT*, mit *120 MHz Intel Pentium MMX* Prozessor, *32 MB* Arbeitsspeicher und *1.6 GB* Festplatte, ausgestattet. Gesteuert werden die Roboter unter dem Betriebssystem *Linux* mittels des ebenfalls von Kurt Konolige entwickelten *Saphira-Systems* [Konolige *et al.*, 1997].

Über das *WaveLan* Radio-Ethernet der Firma *Lucent Technologies* können die Spieler mit einem Zentralrechner kommunizieren. Bei einer Trägerfrequenz von *2.4 GHz* wird eine robuste Funkverbindung mit einer Datenrate von maximal *2Mbit/s* erreicht. Als Zentralrechner kommt ein *Twinhead* Notebook mit einem *133 MHz Intel Pentium* Prozessor, *32 MB* Arbeitsspeicher und *4 GB* Festplatte zum Einsatz.

Für die Ballerkennung wird eine *CCD-Kamera* in Verbindung mit dem *Cognachrome Vision System* von *Newton Labs* verwendet. Das System ist in der Lage, bis zu 6 farblich unterscheidbare Objekte bei einer Framerate von *60 Hz* und einer Bildauflösung von *200×250 Pixeln* zu verfolgen. Für jedes Objekt werden Pixelkoordinaten des Schwerpunktes, Höhe, Breite sowie räumliche Ausdehnung des Objekts ausgegeben.

Alle Objekte – außer dem Ball – werden mit Hilfe eines *PLS200-Laserscanners* der Firma *Sick* erkannt. Der Scanner ist in der Lage, für einen Winkelbereich von 180° und für eine Entfernung von bis zu 30 Metern Informationen mit 0.5° Winkelauflösung und 5 cm Genauigkeit zu liefern. Die Informationen des Scanners sind generell sehr zuverlässig und exakt. Um die Datenrate zu reduzieren, wurde eine Winkelauflösung von nur 1° verwendet. Damit können über eine serielle Verbindung mit 38.400 Baud bis zu 8 Scans pro Sekunde übermittelt werden.

Lediglich für die Ballmanipulation wurden mit Teilen des *Märklin Metallbaukastens* eigene Komponenten entwickelt: Für eine effektivere Ballführung wurden seitliche Führungsfedern und für eine bessere Ballbeschleunigung ein auf Magnetspulen basierender Schußmechanismus für den Pioneer 1 entwickelt [Herrman, 1998].

Abbildung 3.1: Der Torwart und zwei Feldspieler

Bild 3.1 zeigt drei um die in diesem Abschnitt beschriebenen Komponenten erweiterte Pioneer 1 Roboter.

## 3.2 Teamarchitektur

Die Teamarchitektur sieht jeden Roboter als prinzipiell vollständig autonomen Spieler vor. Durch optionale Kommunikation mit dem Zentralrechner außerhalb des Spielfelds kann ein Spieler jedoch Informationen beziehen, die sein Spiel potentiell beeinflussen.

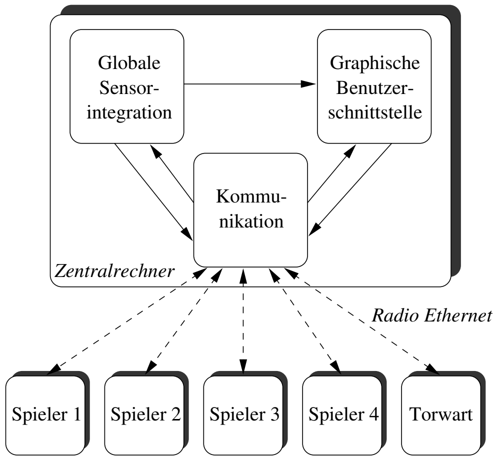



Abbildung 3.2: Die Teamarchitektur

Auf dem Zentralrechner arbeitet ein *Serverprogramm*, das Nachrichten der Spieler empfängt und verarbeitet, sowie Nachrichten an die Spieler versendet. Der Server besteht aus einer Einheit zur *globalen Sensorintegration*, einer *graphischen Benutzerschnittstelle*[1] und einem *Kommunikationsmodul*.

Abbildung 3.2 gibt einen Überblick über die in den folgenden Abschnitten vorgestellten Komponenten der Teamarchitektur.

### 3.2.1 Globale Sensorintegration

Jeder Spieler verwaltet die von ihm wahrgenommenen Objekte in einem lokalen Weltmodell bezüglich des Spielfeldkoordinatensystems. Diese lokale Weltsicht kann ein Spieler dem Server mitteilen, der diese mit den lokalen Weltbildern der anderen Spieler fusioniert. Dieser zu einem globalen Weltmodell führende Vorgang wird als *globale Sensorintegration* bezeichnet. Hierbei wird für die Positionen von Objekten verschiedener Weltmodelle, die gemäß gewissen Kriterien miteinander assoziierbar sind, ein Mittelwert berechnet. Für alle anderen Objekte werden die Positionen aus den lokalen Weltmodellen in das globale übernommen.

Das globale Weltmodell wird in regelmäßigen Abständen an die Spieler versendet, die damit ihre lokale Sicht der Welt erweitern können. Dies ist vor allem

[1] Im Englischen auch als *graphical user interface*, kurz *GUI* bezeichnet.

dann hilfreich, wenn einem Spieler die für ihn nicht erfaßbare Position des Balls mitgeteilt wird oder er mit Hilfe des globalen Weltmodells in der Lage ist, Mitspieler von Gegenspielern zu unterscheiden: Da jeder Spieler dem Server seine eigene Position und seine Identität mitteilt, erhalten auch alle Spieler diese Information über das globale Weltmodell. Weicht die Position eines Objekts des lokalen Weltmodells von der Position eines als Mitspieler bekannten Objekts des globalen Modells weniger als um einen bestimmten Betrag ab, so kann davon ausgegangen werden, daß es sich um das gleiche Objekt handelt, und das lokal wahrgenommene Objekt kann als Mitspieler identifiziert werden. Alle nicht identifizierten Objekte können dann folgerichtig als Gegenspieler betrachtet werden.

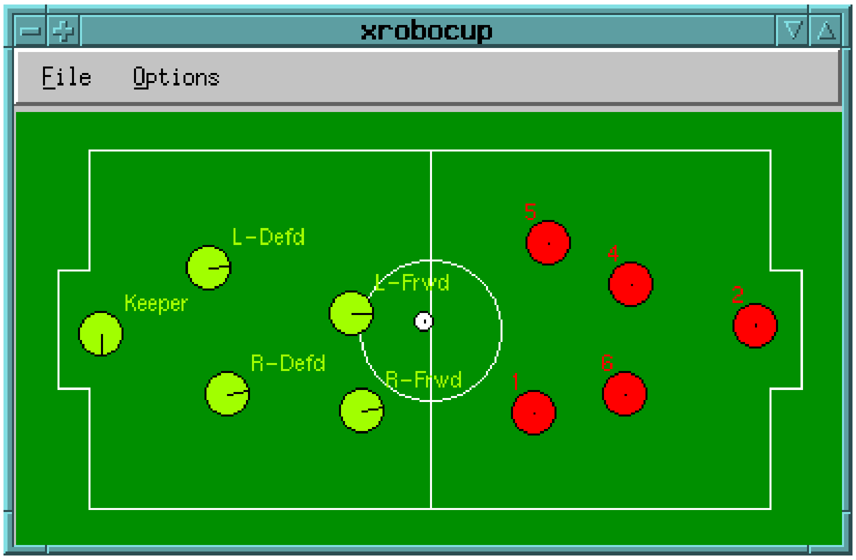



Abbildung 3.3: Das globale Weltmodell

Abbildung 3.3 zeigt das globale Weltmodell, das über die im folgenden Abschnitt vorgestellte *graphische Benutzerschnittstelle* angezeigt werden kann.

### 3.2.2 Graphische Benutzerschnittstelle

Hauptaufgabe der graphischen Benutzerschnittstelle ist die Anzeige des aktuellen Zustands jedes Spielers sowie das Versenden von Anweisungen an die Spieler.

Für jeden Spieler wird protokolliert, welche Aktion gerade ausgeführt wird, wie hoch die Batteriespannung ist und wieviele Nachrichten bisher verschickt wurden. Zur Fehlersuche kann das Versenden von Nachrichten unterbunden werden und die Bildschirmausgabe des Notebooks eines Spielers auf den Bildschirm des Zentralrechners umgeleitet werden.

Auf Knopfdruck kann über die Benutzerschnittstelle per *remote-login* der Start des *Spielerprogramms* veranlaßt werden, das bei Programmbeginn eigenständig

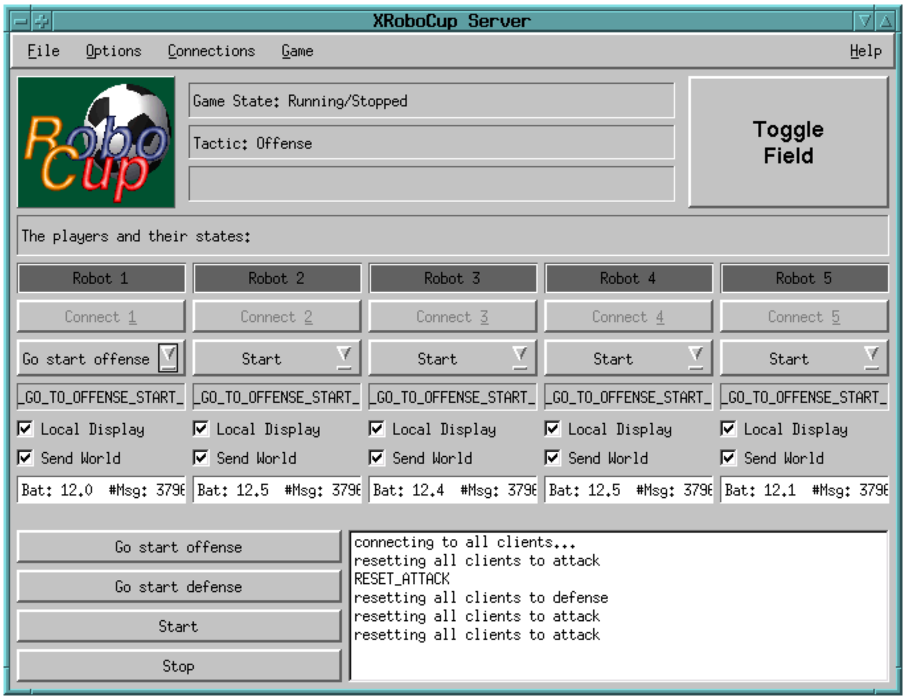



Abbildung 3.4: Die graphische Benutzerschnittstelle

Verbindung zu dem *Serverprogramm* aufnimmt. Ein Spieler kann nun angewiesen werden, zu einer von vier definierten Standardpositionen zu fahren, die Spielrichtung zu ändern, das Spiel zu beginnen, zu unterbrechen oder die Verbindung wieder aufzulösen. Bei Spielanpfiff können per Knopfdruck alle Spieler gleichzeitig angewiesen werden, das Spiel zu starten, wonach bei offiziellen Spielen die Benutzerschnittstelle erst wieder nach Spielabpfiff bedient wird.

Abbildung 3.4 zeigt ein Bild der graphischen Benutzerschnittstelle. Details zur Implementation und Funktionsweise können in [Rittinger, 1998] nachgelesen werden.

### 3.2.3 Kommunikationsmodul

Das Kommunikationsmodul stellt zum einen die Schnittstelle zwischen den Spielern und dem Serverprogramm dar, zum anderen bietet es den Spielern die Möglichkeit, Nachrichten direkt an alle anderen Spieler zu versenden. In Kapitel 8 wird beschrieben, wie dadurch Kooperation der Spieler untereinander ermöglicht wird.

Nachrichten werden jedoch nur dann verschickt, wenn der Kommunikationspartner auch bereit ist, eine Nachricht zu empfangen. Falls eine Nachricht nicht versendet werden kann, so wird sie ausgelassen, um später aktuellere Nachrichten versenden zu können[2].

[2] Im Englischen wird dieses Prinzip auch als *non-blocking-I/O* bezeichnet.

## 3.3 Spielerarchitektur

Ein Spieler handelt gemäß einer gegebenen Taktik aufgrund von Informationen seiner Sensoren und – falls Kommunikation eingesetzt wird – des Servers. Das *Wahrnehmungsmodul* (siehe Abbildung 3.5) integriert neue Informationen in das lokale Weltmodell oder leitet sie direkt mit dem Weltmodell an die *Verhaltensbasierte Kontrolle* weiter, wo die nächste auszuführende Aktion ausgewählt wird und entsprechende Motorsteuerungsbefehle veranlaßt werden. Erfordert eine Aktion Navigation auf dem Spielfeld, so berechnet das *Pfadplanmodul* auf Basis des aktuellen Weltmodells einen kollisionsfreien Pfad zum Zielpunkt.

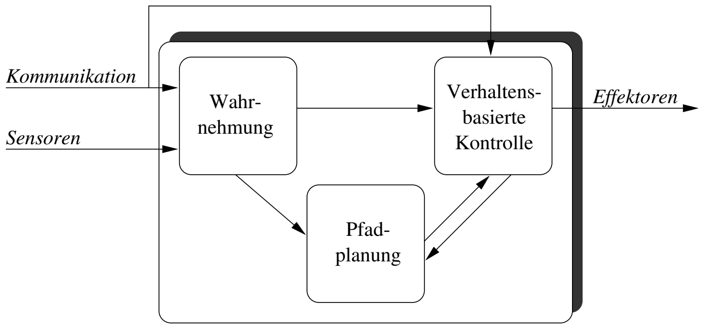



Abbildung 3.5: Die Spielerarchitektur

Die folgenden Abschnitte beschreiben die einzelnen Komponenten und verweisen auf die Kapitel, in denen eine ausführlichere Darstellung erfolgt.

### 3.3.1 Wahrnehmung

Die Position durch Sensoren erkannter Objekte ist zunächst relativ zur Roboterposition gegeben. Damit die Positionsinformationen in das auf Spielfeldkoordinaten basierende lokale Weltmodell integriert werden können, muß daher die Roboterposition auf dem Spielfeld bekannt sein. Aus diesem Grund wird in jedem Verarbeitungszyklus zunächst anhand von Laserscanner- und Odometriedaten die aktuelle Position des Spielers geschätzt. Das Verfahren hierzu ist in Kapitel 4 beschrieben.

Die Daten des Laserscanners werden weiterhin verwendet, um Mit- oder Gegenspieler zu erkennen. Da der Laserscanner so montiert ist, daß der Ball nicht von ihm erfaßt werden kann, muß ein Meßpunkt entweder durch Reflexion des Laserstrahls an einem anderen Roboter oder an der Spielfeldumrandung zustande kommen. In Kapitel 5.3 wird gezeigt, wie Objekte aus den Daten des Laserscanners extrahiert werden, indem Punkte, die der Spielfeldumrandung entsprechen, herausgefiltert und die verbleibenden Punktgruppen mit Objekten assoziiert werden.

Zur Ballerkennung werden anhand der Pixelkoordinaten eines erkannten Objekts der trainierten Ballfarbe dessen Koordinaten in der realen Welt berechnet.

Kapitel 5.4 beschreibt diese Koordinatentransformation und zeigt, wie durch den Vergleich des Ergebnisses mit den Daten der Ultraschallsensoren die Qualität der Schätzung der Ballposition weiter verbessert werden kann.

### 3.3.2 Pfadplanung

Um auf dem Spielfeld kollisionsfrei zu einem bestimmten Zielpunkt zu gelangen, muß ein Pfad gefunden werden, dem ein Spieler sicher folgen kann. Kapitel 6 beschreibt, wie ein solcher Pfad geplant wird, indem das aktuelle Weltmodell zunächst in eine Graphstruktur überführt wird, auf der dann mittels A*-Suche die kürzeste Verbindungsstrecke von der momentanen Roboterposition zu dem Zielpunkt gesucht wird.

### 3.3.3 Verhaltensbasierte Kontrolle

Wie sich ein Spieler verhalten soll, wird stets aufgrund des aktuellen Weltmodells entschieden. Während das Entscheidungsprinzip für alle Feldspieler identisch ist und unterschiedliche Verhalten lediglich durch abweichende Parameter erzielt werden, folgt der Torwart einer grundsätzlich verschiedenen Taktik.

Für das Verhalten des Torwarts ist lediglich die Position, Richtung und Geschwindigkeit des Balls relevant. Die Aktionsauswahl in Abhängigkeit dieser Größen ist Thema des Kapitels 8.1.

Da bei der Auswahl einer Aktion für einen Feldspieler weitere Spielsituationscharakteristika wie z.B. die eigene Position und die anderer Spieler von Bedeutung sind, wird hierfür ein komplexeres Verfahren eingesetzt. Kapitel 8.2 erläutert, welche Aktionen einem Feldspieler zur Verfügung stehen und wie aufgrund der aktuellen Beschreibung der Spielsituation mit Hilfe eines Entscheidungsbaums die aktuell auszuführende Aktion bestimmt wird.

# Kapitel 4

# Selbstlokalisierung

Für eine Vielzahl von Aufgaben in der Robotik ist das Wissen über Ort und Orientierung des Roboters von grundlegender Bedeutung. Erst durch die Kenntnis der Position[1] eines Spielers werden in der Welt des Roboterfußballs taktisches Spiel und Kooperation ermöglicht.

Bei der Selbstlokalisierung – der Bestimmung der eigenen Position – unterscheidet man zwischen absoluten und relativen Verfahren. Während bei relativer Selbstlokalisierung die ungefähre Position des Roboters als bekannt vorausgesetzt und lediglich eine Positionskorrektur relativ zu der aktuellen Position berechnet wird, vermögen absolute Verfahren die Position des Roboters ohne Vorwissen zu bestimmen.

Im folgenden Abschnitt wird zunächst ein Überblick über verschiedene Ansätze zur Selbstlokalisierung und hierzu einsetzbare Sensoren gegeben. Danach wird das Prinzip der Koppelnavigation dargestellt und gezeigt, wie mittels Kalman-Filterung die Positionsschätzungen von Koppelnavigation und Scanüberdeckung fusioniert werden können, um eine Positionsschätzung mit minimalem Fehler zu erhalten.

Im Anschluß wird nach einem kurzen Überblick über alternative Scanüberdeckungsmethoden das neu entwickelte Scanüberdeckungsverfahren vorgestellt und schließlich beschrieben, wie Koppelnavigation und Scanüberdeckung zu einem Selbstlokalisierungsverfahren kombiniert werden, das auf dem RoboCup-Feld eine globale Positionierung in Echtzeit erlaubt. Das Kapitel schließt mit einer Komplexitätsbetrachtung des entwickelten Scanüberdeckungsverfahrens und einer Besprechung der Ergebnisse.

[1] Der Begriff *Position* bezeichnet in dieser Arbeit immer *Ort und Orientierung*.

## 4.1 Ansätze zur Selbstlokalisierung

Es existieren eine Vielzahl von Verfahren zur Selbstlokalisierung, die unter anderem auf Daten folgender Sensoren basieren:

- *Odometrie*[2], bei der Sensoren die Drehbewegungen der Räder messen und dadurch auf den zurückgelegten Weg des Roboters geschlossen werden kann. Besteht die Odometrie aus zwei Rädern, so kann zusätzlich die Richtungsänderung aus der Differenz der beiden gemessenen Wegstrecken ermittelt werden. Problematisch bei der Odometrie ist, daß sich Meßfehler akkumulieren und daher die Positionsschätzung auf langen Strecken sehr ungenau wird.
- *Kreiselkompasse*, mit denen eine sehr genaue Bestimmung der Orientierung des Fahrzeugs möglich ist. Es kann allerdings bei großen Beschleunigungen zu Störungen kommen und die Genauigkeit nimmt mit der Zeit ab (Drift).
- *Ultraschallsensoren*, die aus der Zeit zwischen Aussenden und Empfangen eines Ultraschallsignals eine Entfernungsinformation ableiten. Da die Ultraschallsignale in einem großen Öffnungswinkel ausgesendet werden und die Signale an glatten Oberflächen abgelenkt werden können, liefern diese Sensoren leider nur unzuverlässige und ungenaue Information.
- *Laserscanner*, die nach dem gleichen Prinzip wie Ultraschallsensoren arbeiten, allerdings auf Laserstrahlen beruhen. Durch einen wesentlich kleineren Öffnungswinkel und weniger häufig auftretenden Ablenkungen des Signals bieten Laserscanner aber bedeutend zuverlässigere und exaktere Daten.
- *Kameras*, bei denen aus einem zweidimensionalen Pixelbild Umgebungsmerkmale (z.B. künstliche Landmarken) extrahiert werden, um auf die Roboterposition zu schließen. Die Problematik der Bildverarbeitung wird in der Regel durch Schwankungen in den Beleuchtungsverhältnissen und die Notwendigkeit der Verarbeitung großer Datenmengen erschwert.
- *Andere Sensoren*, die Funk- oder Lichtsignale empfangen können. Werden mindestens drei Signale von Signalquellen an bekannten Orten empfangen, so kann aufgrund des Einfallswinkels der Signale auf die globale Position des Roboters geschlossen werden[3].

Die aus der Literatur bekannten Ansätze zur Selbstlokalisierung lassen sich in folgende drei Klassen einteilen:

- *verhaltensbasierte Ansätze*, bei denen die Selbstlokalisierung ganz auf der Interaktion des Roboters mit der Umwelt basiert. Da bei diesen Ansätzen

[2] Auf Odometrie basierende Verfahren werden als *Koppelnavigation* bezeichnet.

[3] Dieses Prinzip kommt auch bei dem *globalen Positionierungssystem (GPS)* zum Einsatz.

stets nur die Daten eines kleinen Zeitfensters zur Verfügung stehen, ist jedoch die Fähigkeit, sich bezüglich einer Umgebungskarte zu lokalisieren, begrenzt.

- *landmarkenbasierte Ansätze*, die auf der Erkennung von Landmarken beruhen, um den Roboter geometrisch zu lokalisieren. Diese Verfahren erfordern aber entweder eine Modifikation der Umwelt durch Plazierung künstlicher Landmarken oder eine zuverlässige Erkennung von natürlichen Umgebungsmerkmalen.

- *dichte Sensordaten vergleichende Ansätze*, die darauf beruhen, alle verfügbaren Sensordaten mit einer Karte der Umgebung zu vergleichen. Wird eine eindeutige Überdeckung der Meßpunkte mit der Umgebungskarte gefunden, so kann daraus auf die Position des Roboters geschlossen werden.

Wie experimentelle Untersuchungen gezeigt haben, zeichnen sich auf Laserscannerdaten beruhende relative Selbstlokalisierungsverfahren in Verbindung mit Koppelnavigation durch sehr hohe Genauigkeit und Effizienz aus [Gutmann *et al.*, 1998a]. Die in [Burgard *et al.*, 1996] beschriebene Markov-Lokalisierungsmethode bietet zwar die Möglichkeit globaler Positionierung, erfordert aber bei geringerer Genauigkeit einen wesentlich höheren Berechnungsaufwand [Gutmann *et al.*, 1998a].

Aus diesem Grund beruht das selbst entwickelte Verfahren auf der Kombination von Laserscanner- und Odometriedaten. Wie gezeigt werden wird, eignet sich das Verfahren wegen der einfachen Struktur der RoboCup-Umgebung mit nur minimalen Annahmen über die Startposition des Roboters sogar zur globalen Selbstlokalisierung.

## 4.2 Koppelnavigation

Bei der Koppelnavigation[4] wird die Positionsänderung eines Fahrzeugs durch Messen des zurückgelegten Weges der Räder bestimmt. Zu diesem Zweck sind die Pioneer-Roboter an den vorderen Antriebsrädern mit Sensoren ausgestattet, die die Drehbewegung der Räder messen. Meßfehler entstehen insbesondere, wenn die Räder rutschen (Schlupf), der Roboter nicht exakt geradeaus fährt (Drift) oder sich nicht exakt um einen angegebenen Winkel dreht (Winkelfehler).

Mathematisch läßt sich die Position $l$ eines Fahrzeugs zum Zeitpunkt $t$ als Dreitupel angeben. $x$ und $y$ bezeichnen hier die Fahrzeugkoordinaten und $\alpha$ die Fahrzeugorientierung:

$$l(t) \quad = \quad (x(t), y(t), \alpha(t))^T$$

[4] Im Englischen auch als *dead-reckoning* bezeichnet.

Die Positionsänderung $u$ eines Roboters auf kurzen Distanzen kann durch die zurückgelegte Strecke $d$ und den Winkel $\theta$, um den sich der Roboter gedreht hat, angegeben werden:

$$u(t) \quad = \quad (d(t), \theta(t))^T$$

Insbesondere für die im nächsten Kapitel beschriebene Kalman-Filterung ist die Annahme der Meßfehler für Position und Bewegung als normalverteilt bedeutsam:

$$\begin{aligned} l(t) &\sim N(\mu_{l(t)}, \Sigma_{l(t)}) \\ u(t) &\sim N(\mu_{u(t)}, \Sigma_{u(t)}) \end{aligned}$$

Hier bezeichnen $\mu_l$ und $\mu_u$ die jeweiligen Erwartungswerte und $\Sigma_l$ sowie $\Sigma_u$ die zugehörigen Kovarianzmatrizen.

Die Fahrzeugposition $l$ zum Zeitpunkt $t+1$ wird nun aus der Fahrzeugposition $l$ zum Zeitpunkt $t$ und der Eingabe $u(t)$ nach folgender Formel berechnet:

$$l(t+1) \quad := \quad F(l(t), u(t)) = \begin{pmatrix} x(t) + d(t)\cos(\alpha(t)) \\ y(t) + d(t)\sin(\alpha(t)) \\ \alpha(t) + \theta(t) \end{pmatrix}$$

Die Normalverteilung für die neue Roboterposition zum Zeitpunkt $t+1$ berechnet sich damit wie folgt:

$$\begin{aligned} \mu_{l(t+1)} &:= E(F(l(t), u(t))) = \begin{pmatrix} x(t) + d(t)\ cos(\alpha(t)) \\ y(t) + d(t)\ sin(\alpha(t)) \\ \alpha(t) + \theta(t) \end{pmatrix} \\ \Sigma_{l(t+1)} &:= \nabla F_{l(t)} \Sigma_{l(t)} \nabla F_{l(t)}^T + \nabla F_{u(t)} \Sigma_{u(t)} \nabla F_{u(t)}^T \end{aligned}$$

Hier bezeichnet $E$ den Erwartungswert der Funktion $F$ und $\nabla F_{l(t)}$ sowie $\nabla F_{u(t)}$ deren Jakobi-Matrizen.

Für Details zu diesem Thema sei auf [Gutmann, 1996] verwiesen.

## 4.3 Kalman-Filterung

Durch Kalman-Filterung werden Meßwerte verschiedener Sensoren bezüglich der gleichen Beobachtung fusioniert, um die Genauigkeit der Messung zu erhöhen und den Meßfehler zu minimieren. Voraussetzung hierfür ist, daß die beiden Schätzungen mit einem normalverteilten Fehler vorliegen.

Bei dem in Abschnitt 4.5 beschriebenen Selbstlokalisierungsverfahren werden durch Koppelnavigation und Scanüberdeckung gewonnene Positionsschätzungen fusioniert. Diese seien durch $l$ und $s$ als normalverteilt gegeben:

$$\begin{aligned} l &\sim N(\mu_l, \Sigma_l) \\ s &\sim N(\mu_s, \Sigma_s) \end{aligned}$$

Die Kalman-Filterung beruht nun darauf, die Position als gewichtetes arithmetisches Mittel zu berechnen, wobei die jeweiligen Gewichte umgekehrt proportional zu der zugehörigen Varianz sind und die resultierende Positionsunsicherheit geringer als die der Ausgangsschätzungen ist:

$$\mu := (\Sigma_l^{-1} + \Sigma_s^{-1})^{-1} \cdot (\Sigma_l^{-1}\mu_l + \Sigma_s^{-1}\mu_s) \tag{4.1}$$
$$\Sigma := (\Sigma_l^{-1} + \Sigma_s^{-1})^{-1} \tag{4.2}$$

Eine ausführlichere Beschreibung des Verfahrens kann in [Maybeck, 1990; Smith *et al.*, 1990; Gutmann, 1996] nachgelesen werden.

## 4.4 Scanüberdeckung

Durch Aussenden eines Laserstrahls in verschiedene Richtungen entlang einer Achse lassen sich zweidimensionale Abstandsprofile gewinnen. Abbildung 4.1 zeigt einen zweidimensionalen *Scan* über einen Winkelbereich von 180° bei einer Winkelauflösung von 0.5°. Man erkennt deutlich die Randlinien des RoboCup-Spielfeldes und drei Punktwolken, die durch Reflexion des Laserstrahls an Objekten auf dem Feld zustande gekommen sind. Im folgenden bezeichnet der Begriff *Scan* immer eine Menge von Scanpunkten zusammen mit der zugehörigen Aufnahmeposition.

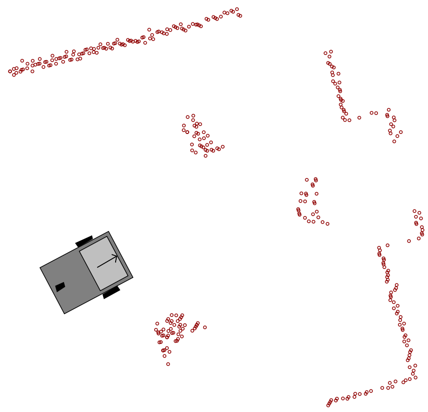


Abbildung 4.1: Ein Scan

Mit Scanüberdeckung bezeichnet man ganz allgemein den Versuch, die Punkte eines Scans mit einem Modell der Umgebung in Deckung zu bringen. Ziel ist es, den Scan so zu verschieben und zu verdrehen, daß eine maximale Überdeckung erreicht wird. Aus der Überdeckung kann dann auf die Scanaufnahmeposition bezüglich des Modells und damit auf die Roboterposition geschlossen werden.

### 4.4.1 Ansätze aus der Literatur

Aus der Literatur sind bereits ein Vielzahl von Scanüberdeckungsverfahren bekannt. Die wichtigsten unter ihnen werden an dieser Stelle kurz vorgestellt:

**Der Cox-Algorithmus:**
Cox [1991] stellt einen Algorithmus vor, der auf der Zuordnung von Scanpunkten zu Linien eines a-priori-Linienmodells beruht. Aus der Zuordnung läßt sich dann die Verschiebung und Verdrehung des Scans gegenüber dem Linienmodell bestimmen. Das Verfahren benötigt eine ungefähre Anfangsschätzung der Aufnahmeposition und setzt eine polygonale Struktur der Umgebung voraus.

**Der Kreuzkorrelationsalgorithmus:**
Weiss et al. [1994] beschreiben ein Verfahren, um ausgehend von einer ungefähren Schätzung der Aufnahmeposition zwei Scans unter Verwendung von Histogrammen zur Deckung zu bringen. Ein Histogramm ist eine Statistik der in einem Scan auftretenden Merkmale, wie Winkel zwischen aufeinanderfolgenden Scanpunkten oder Verteilung der x- bzw. y-Koordinaten. Durch Korrelation der Winkelhistogramme der beiden Scans wird zunächst die Verdrehung, danach durch Korrelation der x- und y-Histogramme die Verschiebung bestimmt.

**Der IDC-Algorithmus:**
Ein universell einsetzbarer Ansatz stammt von Lu und Milios [1994]. Der IDC-Algorithmus versucht, jeden Scanpunkt einem Punkt eines Referenzscans zuzuordnen. Das Verfahren setzt keinerlei Umgebungsmerkmale voraus, benötigt aber wie die vorigen Verfahren eine ungefähre Anfangschätzung der Aufnahmeposition.

**Der kombinierte Überdeckungsalgorithmus:**
Eine Synthese des Cox- und IDC-Algorithmus wurde von Gutmann und Schlegel [Gutmann, 1996; Gutmann und Schlegel, 1996] entwickelt. Eine Entscheidungslogik wählt hier, je nach Struktur der Umgebung, den für die Situation effektiveren Algorithmus aus.

**Das Verfahren von Guibas**
Guibas et al. [1997] stellen ein Verfahren vor, das auf der Überdeckung von Polygonen beruht. Hierbei wird eine polygonale Umgebungskarte in endlich viele *Sichtbarkeitspolygone* zerlegt, innerhalb derer sich die Menge der sichtbaren Kantenecken nicht ändert. Indem aus den Laserscannerdaten ein *Sichtbarkeitspolygon des Roboters* extrahiert und eine Überdeckung mit den Sichtbarkeitspolygonen des Umgebungspolygons ermittelt wird, kann auf die Position des Roboters geschlossen werden. Das Verfahren setzt allerdings unverrauschte Sensordaten, die Kenntnis der exakten Roboterorientierung und die Abwesenheit von Hindernissen voraus.

**Das Verfahren von Castellanos:**
Castellanos et al. [1996] beschreiben ein Verfahren, das nicht wie die bisher

> beschriebenen Verfahren auf einer Zuordnung von Scanpunkten zu Modellpunkten oder -linien beruht, sondern zunächst Linien aus einem Scan extrahiert und dann versucht, diese den Linien einer Umgebungskarte zuzuordnen. Durch Überprüfung, ob gewisse geometrische Bedingungen eingehalten werden, wird der Zuordnungsprozeß erheblich beschleunigt. Prinzipiell erlaubt das Verfahren eine Positionierung ohne Kenntnis der Startposition des Roboters.

Problematisch bei den ersten vier vorgestellten Verfahren ist die Tatsache, daß sie durch die Verarbeitung einzelner Scanpunkte einen hohen Berechnungsaufwand erfordern (siehe Abschnitt 4.7). Da jeweils nur ein eingeschränkter Suchraum betrachtet wird, erlauben diese Verfahren zudem nur eine relative Selbstlokalisierung. Weil der IDC-Algorithmus keinerlei Annahmen über die Struktur der Umgebung macht, ist er in vielen Fällen jedoch wegen seiner Allgemeinheit anderen Verfahren vorzuziehen. Um die Entscheidung für den optimalen Algorithmus zu automatisieren, wurde der kombinierte Überdeckungsalgorithmus entwickelt.

Da das Verfahren von Guibas et al. wegen der erwähnten Voraussetzungen als rein theoretischer Ansatz betrachtet werden muß und auch bei einer Anpassung des Verfahrens an die Praxis [Karch *et al.*, 1997] z.B. Fragen bezüglich der Problematik von Hindernissen ungeklärt bleiben, wurde dieser Ansatz nicht weiterverfolgt.

Für Umgebungen mit genügend polygonalen Strukturen scheint sich das Verfahren von Castellanos et al. anzubieten, da es effiziente Verarbeitung und globale Positionierung verspricht. Um dies zu bestätigen, müssen aber noch experimentelle Untersuchungen abgewartet werden.

Die einfache, polygonale und fest vorgegebene Struktur der RoboCup-Umgebung erlaubt den Einsatz eines Verfahrens, das nach dem gleichen Prinzip wie die Methode von Castellanos arbeitet. Der eigene Ansatz ermöglicht es, den Roboter in Echtzeit zuverlässig zu lokalisieren und erfordert nur minimale Annahmen über die Startposition des Roboters.

### 4.4.2 Der eigene Ansatz zur Scanüberdeckung

Grundlage für das in diesem Abschnitt beschriebene Scanüberdeckungsverfahren ist eine Menge $S$ von aus einem Scan extrahierter Linien und eine Menge $M$ von Linien eines Modells der Umgebung[5].

Um sicherzustellen, daß $S$ nur Linien enthält, die einer Modellinie aus $M$ entsprechen, wird bei der Extraktion auf die Einhaltung gewisser geometrischer Bedingungen geachtet. Für RoboCup bewirken diese, daß zu kurze Linien (die

[5]Wie aus einem Scan Linien extrahiert werden, kann in [Gutmann, 1996] nachgelesen werden.

auch zu einem Objekt gehören könnten) aus $S$ entfernt werden und für alle verbleibenden $s \in S$ überprüft wird, ob sie – wie die Linien des RoboCup-Spielfelds – paarweise parallel oder orthogonal zueinander stehen.

Bei dem Verfahren werden Abbildungen $h$ betrachtet, die jede Scanlinie aus $S$ auf eine der Modellinien in $M$ abbilden:

$$h: \quad S \to M$$

Sei $U$ die Menge aller $|M|^{|S|}$ möglichen Abbildungsvorschriften $h$. Aus $U$ werden nun zunächst all diejenigen $h \in U$ ausgewählt, mit deren Abbildungsvorschrift die Modellinien durch die Scanlinien so überdeckt werden, daß der Fehler in der Überdeckung einen Maximalbetrag MAX-ERROR nicht überschreitet. Der Fehler wird als die Summe der Abstandsquadrate zwischen Scanlinien und zugehörigen Modellinien ausgedrückt:

$$H \quad := \quad \left\{ h \in U \;\middle|\; \sum_{s \in S} d(s, h(s))^2 < \text{MAX-ERROR} \right\} \qquad (4.3)$$

Hier berechnet die Funktion $d$ die Distanz zwischen Scanlinie und Modellinie als die Summe der Distanzen zwischen den Endpunkten der Scanlinie und dem un-

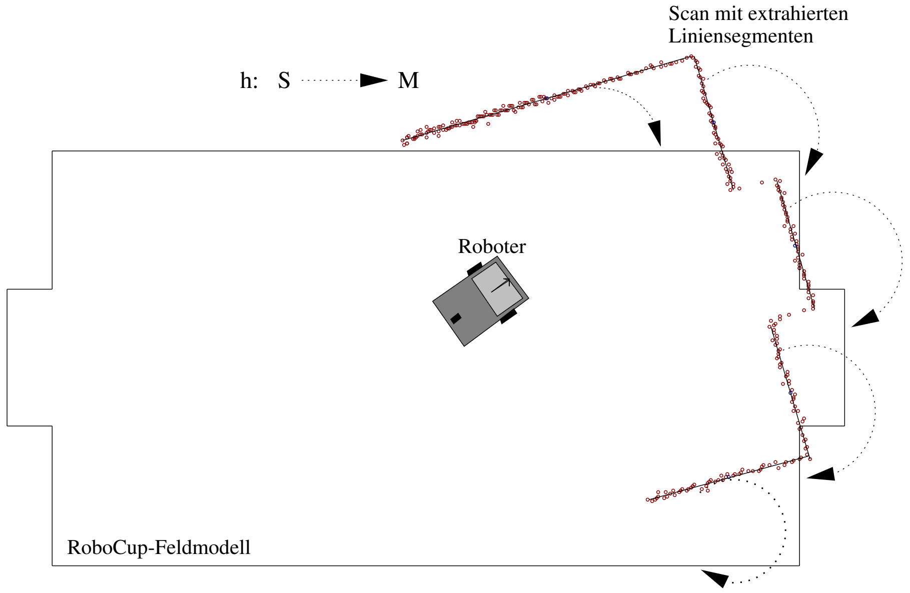



Abbildung 4.2: Eine Zuordnungshypothese $h \in H$

endlichen Geradenstück, auf dem die Modellinie liegt. Die Menge $H$ beinhaltet alle $h$, die dem beschriebenen Qualitätskriterium genügen.

Abbildung 4.2 zeigt ein $h \in H$, wobei die Pfeile jeweils die Zuordnung von Scanlinie zu Modellinie andeuten.

Zur Erzeugung der Hypothesenmenge $H$ wird ein Verfahren angewendet, das rekursiv alle $h$ aus $U$ erzeugt. Auf jeder Rekursionsstufe werden *Teilfunktionen*

$h'$ mit Definitionsbereich $S' \subseteq S$ erzeugt:

$$h' : \quad S' \to M$$

Ein $h'$ wird als Teilfunktion einer Funktion $h$ bezeichnet, falls gilt:

$$h(s) = \begin{cases} h'(s) & : \quad s \in S' \\ h(s) & : \quad sonst \end{cases}$$

Auf jeder Rekursionsstufe wird für jedes $h'$ die Fehlersumme berechnet. Überschreitet diese den Wert MAX-ERROR, so können alle $h$ als Hypothesen verworfen werden, die $h'$ als Teilfunktion haben[6].

Jedes $h \in H$ impliziert eine Verdrehung und Verschiebung des Scans, um Scanlinien und Modellinien zur Deckung zu bringen. Jedes $h \in H$ impliziert damit auch eine Hypothese für die tatsächliche Aufnahmeposition des Scans. Im nächsten Schritt wird daher eine Menge $P$ gebildet, die alle plausiblen Positionshypothesen von Funktionen $h \in H$ enthält:

$$P \quad := \quad \{pos(h) \mid h \in H \text{ \& h impliziert plausible Aufnahmeposition des Scans}\}$$

Idealerweise enthält die Menge $P$ nach dieser Auswahl nur noch ein Element – die gesuchte Schätzung für die Scanaufnahmeposition. Ist dies nicht der Fall, so liegt eine unauflösbare Mehrdeutigkeit vor.

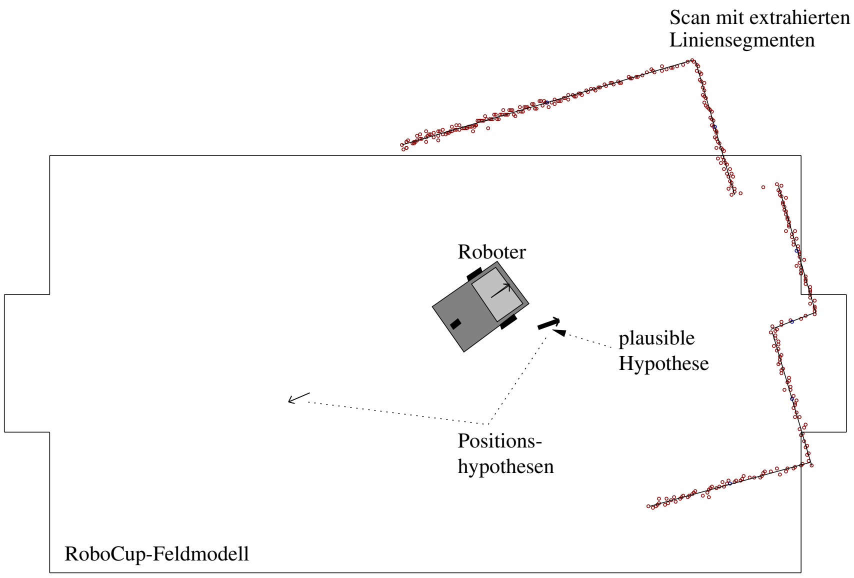



Abbildung 4.3: Zwei Positionshypothesen $p \in P$

Abbildung 4.3 zeigt zwei Positionshypothesen $p \in P$, wobei der fett gezeichnete Pfeil diejenige bezeichnet, die als plausible Hypothese betrachtet wird. Für RoboCup wird jeweils diejenige Position ausgewählt, deren Distanz und Winkel zu der letzten Positionsschätzung am geringsten ist.

[6] In [Castellanos *et al.*, 1996] wird dieses Prinzip als *identify while locating* bezeichnet.

Im folgenden wird nun das Verfahren zur Scanüberdeckung algorithmisch dargestellt. Der Algorithmus *ScanMatch* beschreibt das Überdeckungsverfahren:

**Algorithmus 4.1** *ScanMatch(S, M)*
**Eingabe:** Scanlinienmenge $S$, Modelllinienmenge $M$
**Ausgabe:** Position $p$
**Ablauf:**

$S := ScanLinesLengthFilter(S, \text{MIN-LENGTH})$
**if** $DomainConstraintsHold(S)$ **then**
  $H := MatchHypotheses(S, M, \{\})$
  **if** $|H| > 0$ **then**
    $P := PlausiblePositions(H, RobotPosition(S))$
    **if** $|P| = 1$ **then**
      $p := first(P)$
      **return** $p$
**return** $ERROR$

Die Funktion *ScanLinesLengthFilter(S, MIN-LENGTH)* entfernt aus $S$ all diejenigen Linien, die kürzer als MIN-LENGTH sind. *DomainConstraintsHold(S)* liefert $TRUE$, falls alle domainspezifischen Constraints[7] für $S$ eingehalten werden und *PlausiblePositions(H, L)* berechnet aus der Menge von Zuordungshypothesen $H$ deren implizierte Scanaufnahmeposition und gibt die Menge all derjenigen Positionen zurück, die bezüglich der aktuellen Roboterposition $L$ als plausibel erscheinen. Die Funktion *MatchHypotheses* berechnet rekursiv die Menge $H$:

**Algorithmus 4.2** *MatchHypotheses(S, M, Match)*
**Eingabe:** Scanlinienmenge $S$, Modelllinienmenge $M$, Zuordnungsmenge $Match$
**Ausgabe:** Menge von Zuordnungshypothesen $H$
**Ablauf:**

**if** $|Match| = |S|$ **then**
  $H := Match$
**else**
  $H := \{\}$
  $s := SelectScanline(S, Match)$
  **for all** $m \in M$ **do**
    **if** $VerifyMatch(S, M, Match \cup \{(s, m)\})$ **then**
      $H := H \cup MatchHypotheses(S, M, Match \cup \{(s, m)\})$
**return** $H$

Hier wählt *SelectScanline(S, Match)* die jeweils als nächstes zu betrachtende Scanlinie aus $S$ aus. *SelectScanline* wurde so implementiert, daß sie zu Beginn Linien auswählt die sich möglichst in einem großen Winkel schneiden. Wie sich in Abschnitt 4.6 zeigen wird, wirkt sich dieses Verhalten günstig auf die Gesamtlaufzeit des Verfahrens aus.

Die Funktion *VerifyMatch(S,M,Match)* berechnet die Fehlersumme, die sich aus

[7] Für RoboCup betreffen diese die Winkelkonsistenz der Linien.

der Überdeckung von $M$ durch $S$ gemäß der Abbildungsvorschrift $Match$ ergibt und liefert $TRUE$, falls die Fehlersumme kleiner als MAX-ERROR ist. Bei der Fehlersummenberechnung werden stets alle Scanlinien betrachtet, da schließlich die minimale Gesamtsumme darüber entscheidet, ob ein $h$ in $H$ aufgenommen wird oder nicht. In Abschnitt 4.6 wird sich allerdings zeigen, daß sich dieses Verhalten erkennbar auf die Laufzeit des Verfahrens auswirkt.

Obwohl das Verfahren speziell für die Verwendung bei RoboCup entwickelt wurde, ist es so allgemein gehalten, daß es auch in anderen Umgebungen anwendbar ist, die eine stark polygonale Struktur aufweisen.

## 4.5 Selbstlokalisierung

Nachdem nun bekannt ist, wie zum einen durch Koppelnavigation und zum anderen durch Scanüberdeckung die Position des Roboters geschätzt werden kann, wird in diesem Abschnitt gezeigt, wie das Selbstlokalisierungsmodul die beiden Verfahren kombiniert, um eine maximale Genauigkeit und einen minimalen Fehler der Schätzung zu erreichen. Abbildung 4.4 gibt das Prinzip des Verfahrens schematisch wieder.

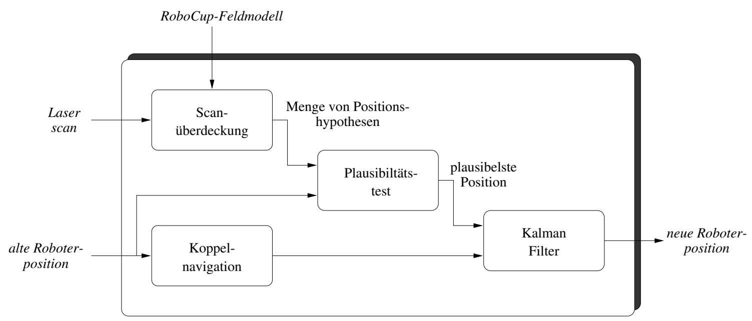



Abbildung 4.4: Das Selbstlokalisierungsmodul

Liegt zu einem Zeitpunkt, an dem die aktuelle Roboterposition geschätzt werden soll, kein aktueller Scan vor, oder konnte der aktuelle Scan nicht erfolgreich überdeckt werden, so wird für die Roboterposition die Schätzung durch die Koppelnavigation übernommen.

Liegt aber eine Positionsschätzung durch Scanüberdeckung vor, dann wird die aktuelle Roboterpositon mittels Kalman-Fusion der beiden Schätzungen von Scanüberdeckung und Koppelnavigation berechnet.

Um eine maximale Unsicherheit bezüglich der Startposition des Roboters zu modellieren, wird zu Beginn des Verfahrens die Roboterposition als der Spielfeldmittelpunkt mit Orientierung in Richtung gegnerisches Tor initialisiert und die Diagonalelemente der Kovarianzmatrix auf unendlich gesetzt. Da dadurch die Positionsschätzung der Scanüberdeckung bei der Kalman-Fusion ein sehr großes Gewicht bekommt (siehe Gleichung 4.1), ist damit zu Beginn des Verfahrens eine globale Selbstlokalisierung möglich – es muß lediglich durch ein grobes Positionieren des Roboters sichergestellt werden, daß zu Beginn die richtige Position als die plausibelste erkannt wird.

Der folgende Algorithmus gibt das beschriebene Verfahren wieder:

**Algorithmus 4.3** *Selflocalization(M)*
**Eingabe:** Modelliniemenge $M$
**Ablauf:**

$InitializeRobotPosition()$
**forever do**
  $S := CurrentScan()$
  $drPos := DeadReckon(Time(S))$
  $smPos := ScanMatch(S, M)$
  **if** $scanPos = ERROR$ **then**
    $newPos := drPos$
  **else**
    $newPos := KalmanFuse(drPos, smPos)$
  $currentPos := DeadReckon(CurrentTime())$
  $UpdateRobotPosition(drPos, currentPos, newPos)$

Hier liefert *DeadReckon(t)* die durch Koppelnavigation bestimmte Roboterposition zum Zeitpunkt $t$. Die Funktion *UpdateRobotPosition(oldPos, currentPos, newPos)* aktualisiert die Roboterposition, indem zu der neuen Roboterposition die Wegstrecke zwischen $oldPos$ und $currentPos$ hinzuaddiert wird, um zu berücksichtigen, daß sich der Roboter in der Regel während der Schätzung von $newPos$ weiterbewegt.

## 4.6 Komplexitätsbetrachtung

In diesem Abschnitt wird das Verhalten des Scanüberdeckungsalgorithmus einer genauen Untersuchung unterzogen, um dann Aussagen über dessen allgemeines Laufzeitverhalten zu machen. Zeitkritisch für das Verfahren ist die Erzeugung der Hypothesenmenge durch die Funktion *MatchHypotheses*, die aus diesem Grund zunächst besonders betrachtet wird.

Betrachtet man die Funktion *MatchHypotheses*, so liegt eine erste Komplexitätsschätzung von $O((|M||S|)^{|S|})$ nahe, da bei einer Rekursionstiefe von $|S|$ prinzipiell auf jeder Stufe $|M|$-Mal die Funktion *VerifyMatch*, die mit $O(|S|)$ abgeschätzt werden kann, aufgerufen wird. Die Funktion *SelectScanline* fällt nicht ins Gewicht, da sie so implementiert werden kann, daß sie einmalig die Scanlinien wie erforderlich sortiert (in maximal $O(|S| \log |S|)$ Schritten), um im folgenden jeweils das nächste Element der sortierten Liste zu liefern. Im weiteren wird aber gezeigt, daß sich die Komplexität von *MatchHypotheses* tatsächlich mit $O(|M|^3|S|^2)$ abschätzen läßt.

Jede Überdeckung von Scan- und Modellinien impliziert eine hypothetische Aufnahmeposition des Scans. Da diese durch ein Dreitupel $(x, y, \alpha)$ gegeben ist, kann das Erzeugen der Menge $H$ (siehe Gleichung 4.3) als Suche nach Dreitupeln $(x_i, y_i, \alpha_i)$ aufgefaßt werden, die als hypothetische Scanaufnahmeposition eine Überdeckung ausreichender Qualität implizieren.

Die Menge $H$ wird rekursiv durch die Funktion $MatchHypotheses$ erzeugt, indem in jedem Rekursionsschritt die Definitionsbereiche für Teilfunktionen $h'$ erweitert werden und überprüft wird, ob die resultierenden Funktionen $h''$ eine Überdeckung mit ausreichend kleiner Fehlersumme implizieren. Ist dies der Fall, so wird eine Funktion $h''$ wiederum erweitert. Eine Funktion $h''$ entsteht aus der Erweiterung einer Funktion $h'$ um die Zuordnung $s_i \to m_j$ $(s_i \in S\ ,\ m_j \in M)$ wie folgt:

$$h''(s) = \begin{cases} m_j & : \quad s = s_i \\ h'(s) & : \quad sonst \end{cases}$$

Bei der Berechnung der Fehlersummen wird die hypothetische Scanaufnahmeposition jeweils so gewählt, daß die Fehlersumme minimiert wird. Für die Wahl der hypothetischen Scanaufnahmepositionen bestehen zunächst drei Freiheitsgrade, da $x$, $y$ und $\alpha$ frei gewählt werden können. Im Verlauf des Verfahrens werden die Freiheitsgrade aber stetig eingeschränkt, da jede Zuordnung von einer Scanlinie zu einer Modellinie auch implizit die freie Wahl von $x$, $y$ und $\alpha$ eingrenzt.

Durch die Zuordnung der ersten Scanlinie zu einer Modellinie wird der Winkel $\alpha$ implizit festgelegt und damit die Zahl der Freiheitsgrade um einen verringert. Die Modellinie ist durch eine Geradengleichung der Form $m : a\hat{x} + b\hat{y} + c = 0$ beschreibbar. Da die ihr zugeordnete Scanlinie auf dieser Linie liegen muß, sind hierdurch x- und y-Koordinate aneinander gekoppelt und lediglich die Position der Scanlinie auf der Modellinie bleibt frei wählbar. Bereits nach der ersten Zuordnung bleibt somit lediglich ein Freiheitsgrad bei den weiteren Zuordnungen.

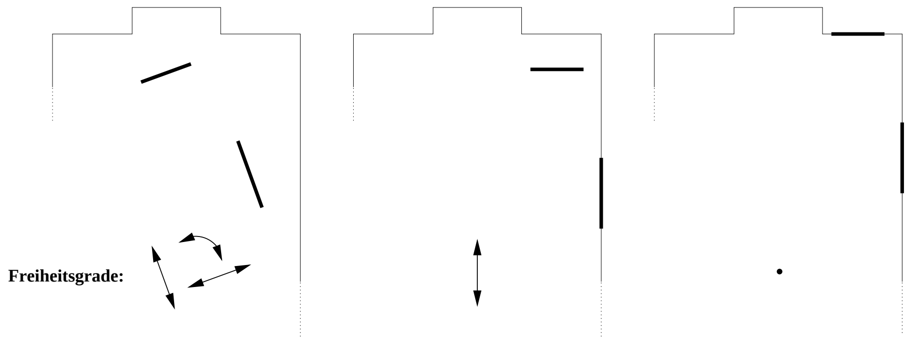



Abbildung 4.5: Einschränkung der Freiheitsgrade bei der Scanüberdeckung

Wird die zweite Scanlinie nun einer Modellinie zugeordnet, die weder parallel noch kolinear zur ersten ist, so bestimmt diese Überdeckung implizit die Lage der ersten Scanlinie auf der ihr korrespondierenden Modellinie. Damit ist auch der letzte Freiheitsgrad genommen und die hypothetische Aufnahmeposition des Scans eindeutig bestimmt. Abbildung 4.5 veranschaulicht diesen Sachverhalt.

Durch die Implementation der Funktion *SelectScanline* wird erreicht, daß zu Beginn zwei zueinander nicht parallele und nicht kolineare Linien zur Bearbeitung ausgewählt werden. Somit steht nach der zweiten Zuordnung bereits die hypothetische Aufnahmeposition fest, und es muß für die weiteren Zuordnungen lediglich anhand der Fehlersummen überprüft werden, ob sie die Hypothese stützen oder nicht. Es kann ohne Einschränkung der Allgemeinheit angenommen werden, daß ab der dritten Rekursionsstufe jeweils nur noch eine Zuordnung $s_i \rightarrow m_j$ gefunden werden kann, die die Hypothese der Scanaufnahmeposition bestätigt.

Mit diesen Überlegungen wird deutlich, daß die Funktion *MatchHypotheses* nur in den ersten beiden Rekursionsstufen tatsächlich $|M|$-Mal aufgerufen werden kann. Da bereits nach der zweiten Rekursionsstufe eine eindeutige Hypothese für die Scanaufnahmeposition besteht, existiert für die jeweils nächste Erweiterung nur maximal eine Zuordnungsalternative, die durch *VerifyMatch* akzeptiert wird. Ab der dritten Rekursionsstufe werden zwar stets $O(|S|)$ Schritte für die Überprüfung der Zuordnungen verwendet, aber es wird jeweils nur einmal erneut *MatchHypotheses* aufgerufen.

Die Zeitkomplexität von *MatchHypotheses* läßt sich daher mittels folgender Rekursionsgleichung bestimmen:

$$F(n) = \begin{cases} O(1) + |M| \cdot n \cdot F(n+1) & : \quad n \leq 2 \\ O(1) + |M| \cdot n + F(n+1) & : \quad n > 2 \end{cases}$$

Durch Auflösen ergibt sich:

$$\begin{aligned} F(|S|) &= O(1) + |M| \cdot (O(1) + |M| \cdot 2 \cdot (O(1) + |M| \cdot 3 + \\ &\qquad O(1) + |M| \cdot 4 + \cdots + O(1) + |M| \cdot (|S| - 1) + O(1)) \\ &= O(|M|^2 \cdot |M||S|^2) \\ &= O(|M|^3|S|^2) \end{aligned}$$

Eine nähere Betrachtung der Funktion *ScanMatch* zeigt, daß die Komplexität der Funktion *MatchHypotheses* die Gesamtkomplexität des Überdeckungsverfahrens bestimmt: die Zahl der Ausführungsschritte der Funktion *ScanLinesLengthFilter* wächst linear in der Anzahl der Scanpunkte, die Funktion *DomainConstraintsHold* benötigt $O(|S|)$ Schritte, und die Laufzeit von *PlausiblePositions* ist durch $O(|H|) = O(|M|^2)$ abschätzbar, da einer Scanlinie tatsächlich nur in den ersten beiden Rekursionsstufen eine beliebige Modellinie zugeordnet werden kann. Die Gesamtkomplexität des Verfahrens zur Scanüberdeckung lautet damit:

$$\boxed{O(|M|^3|S|^2)}$$

Eine Optimierung des Verfahrens auf $O(|M|^3|S|)$ ließe sich erreichen, indem man die Funktion *VerifyMatch* so implementierte, daß jeweils nur die gerade betrachtete Zuordnung von Scanlinie zu Modellinie überprüft wird. Dies würde allerdings bedeuten, daß die Funktion nur eine Näherung der minimalen Fehlersumme berechnen kann, so daß in einigen Fällen konsistente Überdeckungen falsch klassifiziert werden könnten. Da die Zahl der Scanlinien aber durch die Zahl der Scanpunkte begrenzt ist, fällt der Mehraufwand nicht zu sehr ins Gewicht und erscheint für ein zuverlässigeres Verfahren gerechtfertigt.

## 4.7 Ergebnisse

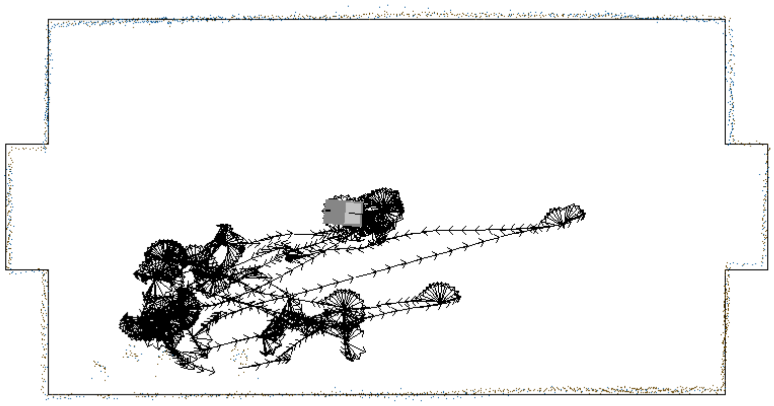
In der Praxis arbeitete das in diesem Kapitel vorgestellte Selbstlokalisierungsverfahren stets exakt und fehlerfrei. Mittels des neu entwickelten Scanüberdeckungsverfahrens wurde eine zuverlässige Schätzung der globalen Roboterposition gewonnen, die durch Kalman-Filterung mit den Odometriedaten noch weiter verbessert werden konnte.

In einer experimentellen Untersuchung wurde der Linienüberdeckungsalgorithmus mit dem Cox-Algorithmus verglichen. Hierzu wurden zunächst die während einer Fahrt des Roboters aufgenommenen Scans gespeichert und anschließend die Scanaufnahmepositionen durch die beiden Verfahren geschätzt. Abbildung 4.6 zeigt die Odometrieschätzung während der Fahrt sowie die mit den beiden Scanüberdeckungsverfahren ermittelten Positionsschätzungen.

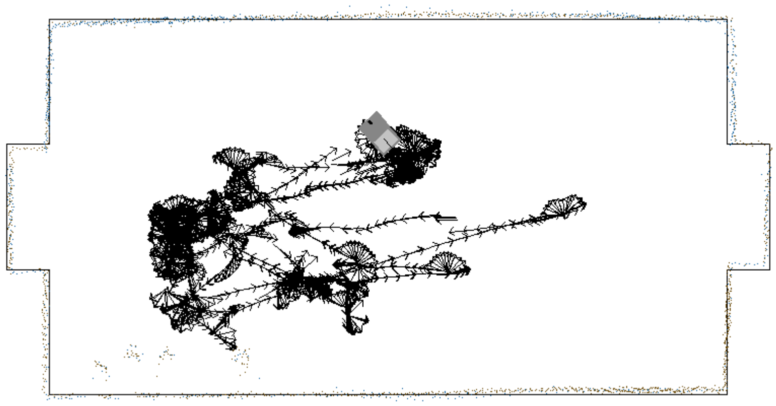
In der Abbildung unterscheiden sich die Positionsschätzungen der beiden Scanüberdeckungsverfahren nur geringfügig, weichen aber deutlich von der Odometrieschätzung ab. Dies läßt erwarten, daß eine vergleichbare Genauigkeit wie für den Cox-Algorithmus erreicht wird [Gutmann *et al.*, 1998a].

Für beide Scanüberdeckungsverfahren wurden auf den in Kapitel 3 vorgestellten Librettos die Zeit gemessen, die im Durchschnitt für die Überdeckung eines Scans benötigt wird. Hierbei war der Linienüberdeckungsalgorithmus mit 3 ms deutlich schneller als der Cox-Algorithmus, der 16 ms benötigte.

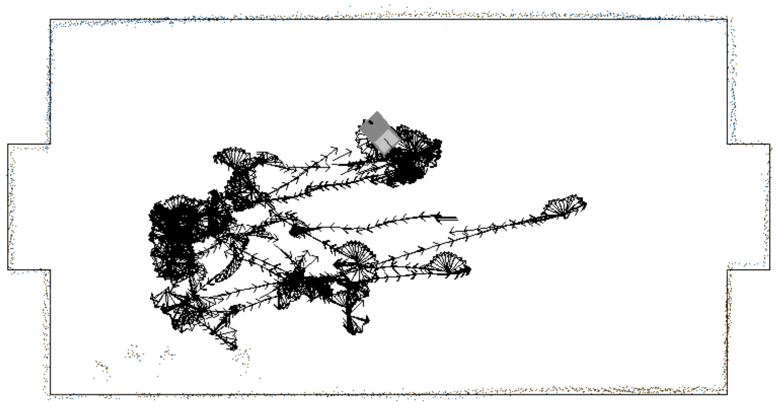
Aufgrund der Allgemeinheit des Verfahrens ist der Linienüberdeckungsalgorithmus nicht nur auf die Anwendung bei RoboCup beschränkt, sondern in jeder Umgebung mit ausreichend polygonalen Strukturen einsetzbar.



Odometrie



Cox-Algorithmus



Linienüberdeckungsalgorithmus

Abbildung 4.6: Die durch die verschiedenen Verfahren geschätzten Scanaufnahmepositionen während der Fahrt. Der Ursprung eines Pfeils gibt die Lage auf dem Spielfeld, die Richtung des Pfeils die Orientierung des Roboters wieder.

# Kapitel 5

# Weltmodellierung

In diesem Kapitel wird beschrieben, wie ein Spieler die ihm zugänglichen Informationen über seine Umwelt wahrnimmt, interpretiert und verwaltet. Verfügt ein Roboter über ein aktuelles und konsistentes *Weltmodell*, das einen Ausschnitt der Realität korrekt abbildet, so kann ihm dieses als solide Basis für die Bewältigung komplexer Aufgaben dienen.

Die Verwendung eines expliziten Weltmodells ist in der Fachwelt umstritten. Die Tatsache, daß mit nur einer Ausnahme [Suzuki *et al.*, 1997] alle bei RoboCup'97 und RoboCup'98 siegreichen Teams [Burkhard *et al.*, 1997; Veloso *et al.*, 1997; Shen *et al.*, 1997; Andre und Teller, 1998; Gutmann *et al.*, 1998b] ein explizites Weltmodell verwendeten, unterstützt jedoch deutlich die These, daß ein solches Weltmodell eine Voraussetzung für ein effektives und erfolgreiches Spiel darstellt.

Da davon ausgegangen werden kann, daß beim Roboterfußball erprobte Methoden auch in verwandten Problemgebieten der Robotik erfolgreich einsetzbar sind, besteht Anlaß für eine intensivere Auseinandersetzung mit der Thematik der Weltmodellierung.

Im folgenden wird zunächst ein Überblick über das Weltmodellierungsverfahren gegeben und die Struktur des bei RoboCup verwendeten Weltmodells vorgestellt. Danach wird beschrieben, wie aus den Laserscannerdaten Informationen über Spieler sowie aus den Kamera- bzw. Ultraschalldaten Informationen über den Ball extrahiert werden. Im Anschluß wird gezeigt, wie anhand dieser Informationen Spieler und Ball modelliert und in einem einheitlichen Weltmodell verwaltet werden. Das Kapitel schließt mit einer Besprechung wichtiger Ergebnisse.

## 5.1 Überblick

Ein Weltmodell repräsentiert auf einheitliche Weise die durch Sensoren erfaßten Informationen über die Umwelt. Da in der Regel nur unvollständige Informationen über die Umwelt zur Verfügung stehen und ein Weltmodell stets nur eine begrenzte Zahl von Aspekten der Umwelt widerspiegeln kann, muß den Sensordaten eine „Bedeutung“ gegeben werden, um sie mit einer Kategorie des Weltmodells assoziieren zu können. So wie bei dem Selbstlokalisierungsverfahren denjenigen Punkten, die auf Linien einer gewissen Mindestlänge liegen, die „Bedeutung“ „zu einer Linie des Spielfeldrandes gehörend“ gegeben wird, müssen alle Sensoreindrücke zunächst interpretiert werden, um in das Weltmodell integriert werden zu können.

Bei der Interpretation der Sensoreindrücke werden drei Kategorien von Objekten unterschieden: der Spieler selbst, andere Spieler und der Ball. Für jede Kategorie steht ein spezialisiertes Modul zur Verfügung, das aus den Sensordaten Informationen über Objekte der Kategorie extrahiert. In Kapitel 4 wurde bereits das Selbstlokalisierungsmodul besprochen, das Informationen über den Spieler selbst liefert. In den folgenden Abschnitten werden die Module zur Erkennung anderer Spieler und des Balls vorgestellt.

Alle Informationen der einzelnen Module werden dem Objektmodellierungsprozeß zugeführt, durch den die unterschiedlichen Daten in ein einheitliches Weltmodell integriert werden. Abbildung 5.1 gibt den Fluß der Informationen bei der Weltmodellierung schematisch wieder.

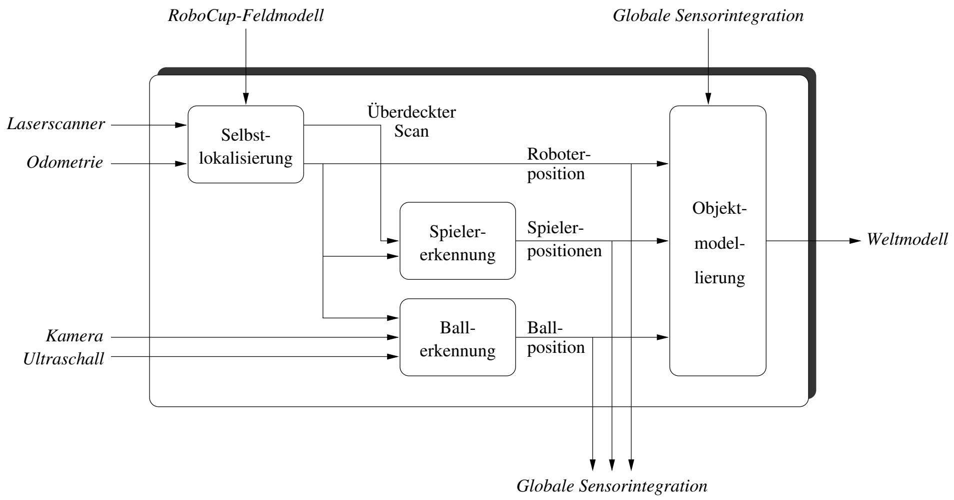



Abbildung 5.1: Das Weltmodellierungsmodul

Da verschiedene Sensoren zu unterschiedlichen Zeitpunkten Informationen über verschiedene Umweltaspekte liefern, muß bei der Verwaltung des Weltmodells der Aufrechterhaltung der zeitlichen und inhaltlichen Konsistenz große Aufmerksamkeit gewidmet werden. Hierauf wird in Abschnitt 5.5 eingegangen.

## 5.2 Struktur des Weltmodells

Ein lokales Weltmodell gibt die Sicht der Welt aus der Perspektive eines Spielers wieder. Relevant sind hierbei Informationen über den Spieler selbst, über den Ball und über andere Spieler auf dem Spielfeld. Zur Verwaltung des Weltmodells wurde daher folgende Datenstruktur gewählt:

**struct** $WorldModel$
  ObjectModel $ownModel$
  ObjectModel $ballModel$
  Integer $numPlayerModels$
  ObjectModel $playerModels$[MAX-PLAYER-MODELS]

Der Ball und alle Spieler werden durch das Weltmodell als Objekte modelliert. Während wegen der besonderen Stellung des Eigen- und Ballmodells für diese spezielle Variablen existieren, werden alle anderen Spieler in einem Array von Spielermodellen verwaltet, wobei *numPlayerModels* die Zahl der aktuell in der Liste vorhandenen Spieler angibt. Ein Objektmodell ist folgendermaßen strukturiert:

**struct** $ObjectModel$
  Position $currentPosition$
  Integer $numLastPositions$
  Position $lastPositions$[MAX-POSITIONS]
  Double $radius$
  Integer $identity$
  Integer $state$

Für jedes Objekt wird neben der aktuellen Position auch eine Liste der zuletzt wahrgenommenen Positionen verwaltet, wobei *numLastPositions* die Zahl der in der Liste vorhandenen Elemente angibt. Die Positionsangaben der Liste werden benötigt, um Richtung und Geschwindigkeit eines Objekts berechnen zu können. Ein Objekt ist durch Angabe der Position und des Radius geometrisch als Kreis repräsentiert, der den von dem Objekt eingenommenen Raum modelliert. Weiterhin wird die Identität („Mitspieler“, „Gegenspieler“ oder „Ball“) und der Zustand eines Objekts verwaltet. In Abhängigkeit davon, wann ein Objekt zum letzten Mal durch einen Sensor wahrgenommen wurde, wird dieses als „sichtbar“, „erinnert“ oder „ungültig“ angegeben. Die Position eines Objekts wird wie folgt beschrieben:

**struct** $Position$
  Double $x, y, \alpha$
  Double $velocity$
  Time $timeStamp$

Die Position eines Objekts ist durch die $x$- und $y$-Koordinaten (bezüglich des Spielfeldkoordinatensystems) und die Bewegungsrichtung $\alpha$ gegeben. Die Richtung, die Geschwindigkeit und der Zeitpunkt, an dem das Objekt durch einen Sensor erfaßt wurde, werden verwaltet, um die Position des Objekts zu einem anderen Zeitpunkt schätzen zu können. Hierauf wird in Abschnitt 5.5 eingegangen. Abbildung 5.2 gibt graphisch einen Zustand der Weltmodellstruktur

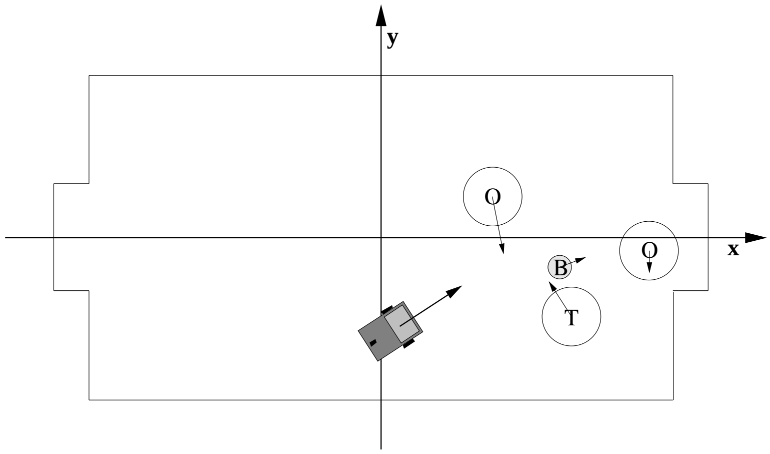



Abbildung 5.2: Ein Weltmodell mit Ball („B“), zwei Gegenspielern („O“) und einem Mitspieler („T“). Die Pfeile geben die Bewegungsrichtung, deren Länge die Geschwindigkeit der Objekte wieder.

wieder. Wie Sensoreindrücke zunächst interpretiert und dann in das Weltmodell integriert werden, zeigen die folgenden Abschnitte.

## 5.3 Spielererkennung

Grundlage für die Spielererkennung ist ein erfolgreich überdeckter Scan. Aus einem solchen Scan können leicht diejenigen Scanpunkte herausgefiltert werden, die einer Linie des Spielfelds entsprechen. Da der Laserscanner in einer Höhe montiert ist, in der der Ball nicht erfaßt werden kann, muß ein verbleibender Scanpunkt durch Reflexion an einem Spieler zustande gekommen sein.

In der Regel entsprechen mehrere Scanpunkte einem Objekt. Daher werden Punktwolken ermittelt, in denen zwei benachbarte Scanpunkte einen Maximalabstand nicht überschreiten. Jede dieser Punktmengen wird anschließend mit einem Spieler assoziiert, indem angenommen wird, daß der Schwerpunkt einer Punktwolke dem Mittelpunkt eines Spielers entspricht. Da der Scan als erfolgreich überdeckt vorausgesetzt wird und somit die eigene Position auf dem Spielfeld bekannt ist, sind damit die Positionen für die Spielermodelle leicht bestimmbar. Abbildung 5.3 veranschaulicht das Prinzip der Spielererkennung.

Trotz der hohen Genauigkeit und Zuverlässigkeit können auch bei einem Laserscanner Meßungenauigkeiten auftreten. Dadurch verursachte Abweichungen äußern sich in Scanpunkten, die vereinzelt ohne Scanpunkte in nächster Nachbarschaft auftreten. Um bei der Bestimmung der Punktwolken dadurch nicht irrtümlich zu viele Mengen zu erhalten, wird ein Scan zunächst mit einem Medianfilter „geglättet“.

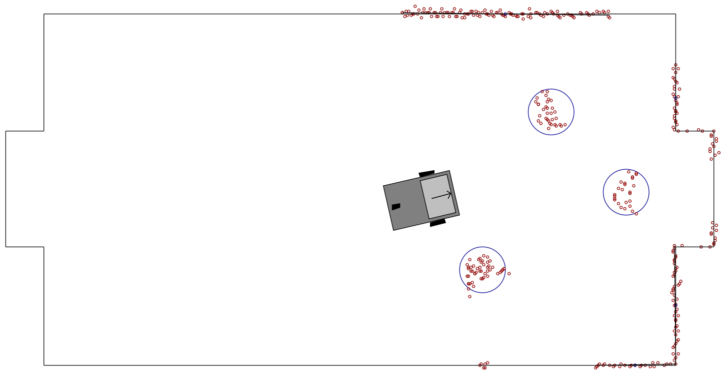


Abbildung 5.3: Ein überdeckter Scan mit drei extrahierten Spielern

Für einen in Polarkoordinaten gegebenen Scanpunkt ersetzt der Medianfilter den Distanzwert durch den Medianwert der Distanzen der benachbarten Scanpunkte. Der folgende Algorithmus zeigt die Arbeitsweise des Medianfilters:

**Algorithmus 5.1** *ScanMedianFilter(S)*
**Eingabe:** Scan $S$
**Ausgabe:** Scan $S'$
**Ablauf:**

$S' := \{\}$
**for all** $s \in S$ **do**
  $P := NeighbourPoints(s, S, \mathrm{NUM})$
  $p := Median(P)$
  $m := CreateScanpoint(Angle(s), Distance(p))$
  $S' := S' \cup m$
**return** $S'$

Die Funktion *NeighbourPoints(s, S, NUM)* erzeugt eine Punktmenge, bestehend aus dem Punkt $s$ und den jeweils $NUM$ linken und rechten Nachbarn in $S$. Aus einer solchen Punktmenge extrahiert die Funktion *Median* denjenigen Punkt $p$, dessen Distanz zur Scanaufnahmeposition dem Medianwert für die Punkte in $P$ entspricht. Im Anschluß wird der Ausgabemenge ein durch *CreateScanpoint* neu erzeugter Scanpunkt hinzugefügt, dessen Polarkoordinaten durch den Winkel von $s$ und die Distanz von $p$ (extrahiert durch die Funktionen *Angle* und *Distance*) zur Aufnahmeposition gegeben sind. Mit einer Komplexität von $O(|S|)$ erlaubt der Medianfilter eine schnelle Vorverarbeitung des Scans.

Nach erfolgreicher Glättung des Scans können die Punkte zu Punktwolken zusammengefaßt werden. Dies geschieht, indem solange Scanpunkte einer Hilfsmenge hinzugefügt werden, bis ein neuer Scanpunkt eine zu große Distanz zum letzten Scanpunkt aufweist. Für die Punkte der Hilfsmenge wird dann der Schwerpunkt ermittelt und dieser der Ausgabemenge hinzugefügt. Nachdem die Hilfsmenge mit dem zuletzt betrachteten Scanpunkt initialisiert wurde, kann fortgefahren werden.

Der folgende Algorithmus gibt die Vorgehensweise bei dem Verfahren wieder:

**Algorithmus 5.2** *ScanClusterFilter(S)*
**Eingabe:** Scan $S$
**Ausgabe:** Scan $S'$
**Ablauf:**

$S' := \{\}$
$ls := first(S)$
$P := \{ls\}$
**for all** $s \in S \setminus first(S)$ **do**
  **if** $DistancePointToPoint(s, ls) <$ MAX **then**
    $P := P \cup \{s\}$
  **else**
    $S' := S' \cup CenterOfGravity(P)$
    $P := \{s\}$
  $ls := s$
**return** $S'$

Hier berechnet die Funktion *DistancePointToPoint(p1, p2)* die Distanz zwischen zwei Punkten $p1$ und $p2$ und *CenterOfGravity(P)* den Schwerpunkt einer Punktmenge $P$. Die Komplexität des Algorithmus kann wieder mit $O(|S|)$ abgeschätzt werden.

Im weiteren Verlauf des Verfahrens werden die Schwerpunkte der Punktwolken als Mittelpunkte von Spielermodellen betrachtet und entsprechend in das Weltmodell eingefügt. Damit läßt sich das Verfahren zur Spielermodellierung algorithmisch wie folgt beschreiben:

**Algorithmus 5.3** *ScanExtractPlayers(W, S, L)*
**Eingabe:** Weltmodell $W$, Scan $S$, Modelllinien $L$
**Ausgabe:** Weltodell $W$
**Ablauf:**

$S := ScanLineDifferenceFilter(S, L, \text{MAX-LINE-DISTANCE})$
$S := ScanMedianFilter(S)$
$S' := ScanClusterFilter(S, \text{MAX-CLUSTER-RADIUS})$
**for all** $s \in S'$ **do**
  $o := CreateObjectModel(CoordX(s), CoordY(s),$
      $\text{PLAYER-RADIUS}, \text{PLAYER}, Time(S))$
  $W := WorldModelInsertObject(W, o)$
**return** $W$

Hierbei entfernt die Funktion *ScanLineDifferenceFilter* alle Punkte aus $S$, die nicht mehr als MAX-LINE-DISTANCE von einer Linie in $L$ abweichen. Die Funktion *CreateObjectModel* erzeugt durch Angabe der Koordinaten (extrahiert durch die Funktionen *CoordX* und *CoordY*), des Radius, der Identität und des Zeitstempels (extrahiert durch die Funktion *Time*) ein Objektmodell, das durch die Funktion *WordModelInsertObject* in das Weltmodell eingefügt wird.

Im Rahmen der Beschreibung der Objektmodellierung wird auf die Funktion *WordModelInsertObject* in Abschnitt 5.5 genauer eingegangen.

## 5.4 Ballerkennung

Zur Ballerkennung werden Kamera- und Ultraschalldaten herangezogen. Befindet sich ein Objekt der trainierten Ballfarbe im Bildbereich der Kamera (siehe Abbildung 5.4), so gibt das Bildverarbeitungssystem die Pixelkoordinaten des Objektschwerpunktes aus.

Abbildung 5.4: Ein Pixelbild als Grundlage der Ballerkennung

Von diesen Pixelkoordinaten muß nun auf die Koordinaten des Objekts relativ zum Roboter geschlossen werden. Diese Transformation von Pixelkoordinaten zu einer Position der realen Welt wird für eine Menge ausgewählter Positionen (siehe Abbildung 5.5) in der realen Welt explizit gemessen.

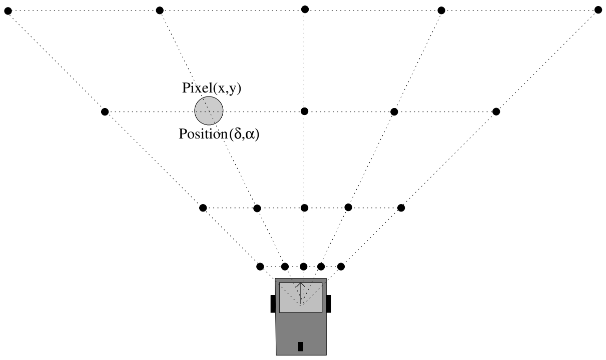



Abbildung 5.5: Meßpunkte für die Kalibrierung der Ballerkennung

Für alle anderen Pixelkoordinaten werden die korrespondierenden Weltpositionen durch Interpolation berechnet. Dies geschieht, indem zunächst diejenigen drei Meßpunkte bestimmt werden, durch deren korrespondierende Pixelkoordinaten ein Dreieck beschrieben ist, das den in Frage stehenden Pixelpunkt beinhaltet. Durch lineare Interpolation wird dann bestimmt, wie die Vektoren zu den drei Meßpunkten gewichtet werden müssen, um die korrespondierenden Weltpositionen zu erhalten.

Um die Genauigkeit des Verfahrens zu erhöhen, werden die Koordinaten im Anschluß in Polarkoordinaten transformiert und die Distanzinformation mit den Daten der Ultraschallsensoren verglichen. Zeigen die Ultraschallsensoren ein Objekt in ähnlicher Richtung und ungefähr gleichem Abstand an, so wird die genauere Entfernungsinformation des Ultraschalls verwendet.

Eine detaillierte Beschreibung des Verfahrens kann in [Topor, 1998] nachgelesen werden. Dort wird auch gezeigt, daß die Komplexität des Verfahrens mit $O(|M|)$ abschätzbar ist, wobei $M$ die Menge der Meßpunkte bezeichnet.

Kann anhand des Pixelbilds zum Zeitpunkt $t$ die Ballposition bestimmt werden, so wird diese mit einem Aufruf von

$$o := CreateObjectModel(x, y, \text{BALL-RADIUS}, \text{BALL}, t)$$
$$W := WorldModelInsertObject(W, o)$$

in das Weltmodell integriert, wobei $x$ und $y$ die Koordinaten des Balls bezeichnen, BALL-RADIUS den Radius des Balls und $BALL$ die Identität des Objekts mitteilt.

## 5.5 Objektmodellierung

Nachdem in den letzten beiden Abschnitten und in Kapitel 4 beschrieben wurde, wie Informationen über den Spieler, über andere Spieler und über den Ball erfaßt und interpretiert werden, wird in diesem Abschnitt gezeigt, wie die zu verschiedenen Zeitpunkten über unterschiedliche Objekte vorliegenden Informationen in einem einheitlichen, konsistenten Weltmodell verwaltet werden.

Bei der Ball- und Spielererkennung werden aus den Sensordaten für ein Objekt jeweils die Koordinaten und die Aufnahmezeit (Zeitstempel) extrahiert. Da Kamera und Ultraschall ausschließlich zur Ballerkenung und der Laserscanner ausschließlich zur Spielererkennung verwendet wird, implizieren die Sensordaten weiterhin die Identität und damit auch die Größe (Radius) der Objekte.

Aus den Sensordaten gehen aber weder Richtungs- noch Geschwindigkeitsinformationen hervor. Daher werden diese nach folgendem Prinzip ermittelt: Sind für ein Objekt die Aufenthaltsorte zu verschiedenen Zeitpunkten bekannt, so kann anhand der zurückgelegten Wegstrecke und der verstrichenen Zeit die Richtung und die Geschwindigkeit des Objekts geschätzt werden. Zu diesem Zweck werden für jedes Objekt des Weltmodells die zuletzt wahrgenommenen Positionen in einer Liste gespeichert.

Um für ein Objektmodell eine solche Liste verwalten zu können, muß für jedes in das Weltmodell einzufügende Objekt vorher überprüft werden, ob es ein neu erfaßtes Objekt darstellt, oder ob es lediglich aktuellere Informationen bezüglich eines bereits im Weltmodell vorhandenen Objekts liefert. Ist ein einzufügendes Objekt mit einem Objekt des Weltmodells assoziierbar, so kann für dieses die Liste der letzten Positionen aktualisiert und die aktuelle Geschwindigkeit und

Richtung des Objekts berechnet werden. Im anderen Fall muß das Objekt dem Weltmodell hinzugefügt und zunächst als statisch angenommen werden.

Während für das Eigen- und das Ballmodell stets eine eindeutige Zuordnung zwischen einzufügendem Objekt und Objekt des Weltmodells besteht, muß für aus einem Scan extrahierte Objekte zunächst nach einem korrespondierenden Objekt im Weltmodell gesucht werden.

Um eine Korrespondenz zwischen einem einzufügenden Objekt und einem Objekt des Weltmodells herstellen zu können, müssen die Daten bezüglich ein und desselben Zeitpunkts verglichen werden. Zu diesem Zweck werden für alle Objekte deren Positionen zum Aufnahmezeitpunkt des einzufügenden Objekts geschätzt. Die Funktion *PositionEstimate(o, t)* berechnet für ein Objekt $o$ dessen geschätzte Position zum Zeitpunkt $t$:

**Algorithmus 5.4** *PositionEstimate(o, t)*
**Eingabe:** Objekt $o$, Zeit $t$
**Ausgabe:** Position $p$
**Ablauf:**

$p := o.position$
$td := TimeDifference(t, p.timeStamp)$
$dx := cos(p.\alpha) \cdot p.velocity \cdot td$
$dy := sin(p.\alpha) \cdot p.velocity \cdot td$
$p.timeStamp := t$
$p.x := p.x + dx$
$p.y := p.y + dy$

**return** $p$

Hier berechnet die Funktion *TimeDifference* die Zeitdifferenz zwischen zwei Zeitpunkten.

Weichen die Positionen zweier Objekte bezüglich des gleichen Zeitpunkts nicht um mehr als einen Maximalwert voneinander ab, so wird angenommen, daß es sich bei den Objekten um ein und dasselbe handelt. In diesem Fall kann die Richtungs- und Geschwindigkeitsinformation neu berechnet werden. Hierbei wird sowohl bei der Richtungs- als auch bei der Geschwindigkeitsberechnung über die letzten Positionen gemittelt. Dies entspricht einer Tiefpaßfilterung der Daten, die zu stabileren Richtungs- und Geschwindigkeitsinformationen führt.

Da sich die Positionsinformationen der Ballerkennung als sehr unzuverlässig herausgestellt haben, wird eine neue Positionsinformation des Balls mit den Ballpositionen der letzten halben Sekunde gemittelt. Die Tiefpaßfilterung der Ballpositionen hat sich als ein sehr wirksames Instrument erwiesen, um starke, durch Meßungenauigkeiten[1] verursachte Positionssprünge des Balls auszugleichen.

[1] Meßungenauigkeiten entstehen unter anderem durch die asynchrone Anbindung der Sensoren an das System. Da zur Spielererkennung aber stets aktuell überdeckte Scans verwendet werden, ist hier die Referenzposition zum Zeitpunkt der Sensorinformation bekannt und damit das Verfahren bedeutend exakter.

Die Funktion *UpdateObjectModel* aktualisiert ein Objektmodell wie folgt:

**Algorithmus 5.5** *UpdateObjectModel(o, u)*
**Eingabe:** Objektmodell $o$, Objektmodell $u$
**Ausgabe:** Objektmodell $o$
**Ablauf:**

$o := UpdateHistory(o, u)$
$o := AverageAngle(o)$
$o := AverageVelocity(o)$
**if** $o.identität = BALL$ **then**
    $o := AveragePosition(o)$
**return** $o$

Hier aktualisiert die Funktion *UpdateHistory* das Array der letzten Positionen des Objekts, indem die neue Information aus $u$ an den Anfang gesetzt wird und alle Elemente des Arrays nach hinten verschoben werden. Die *Average*-Funktionen mitteln jeweils bezüglich der im zweiten Teil des Namens angegebene Information.

Hiermit sind alle wesentlichen Funktionen bekannt, die zur Integration eines Objektmodells in das Weltmodell notwendig sind. Die Funktion *WorldModelInsertObject* faßt das Verfahren zusammen:

**Algorithmus 5.6** *WorldModelInsertObject(W, object)*
**Eingabe:** Weltmodell $W$, Objektmodell $object$
**Ausgabe:** Weltmodell $W$
**Ablauf:**

$W' := \{\}$
**for all** $o \in W$ **do**
    $W' := W' \cup PositionEstimate(o, Time(object))$
$c := CorrespondingObjectModel(W', object)$
**if** $c \neq$ ERROR **then**
    $W := DeleteObjectModel(W, c)$
    $object := UpdateObjectModel(c, object)$
$W := AddObjectModel(W, object)$
**return** $W$

Hier liefert die Funktion *CorrespondingObjectModel(W, o)* ein zu $o$ korrespondierendes Objektmodell des Weltmodells $W$. Falls kein Korrespondenzpartner existiert, wird ERROR zurückgegeben. Die Funktion *Time(o)* ermittelt den Zeitstempel der aktuellsten Position des Objekts $o$. Durch die Funktionen *Delete-* bzw. *AddObjectModel* werden Objekte aus dem Weltmodell entfernt bzw. dem Weltmodell hinzugefügt.

Da ein korrespondierendes Objektmodell in Zeit $O|W|)$ gefunden werden kann und alle anderen aufgerufenen Funktionen lediglich einen Aufwand von $O(1)$ bedeuten, kann ein Objektmodell in Zeit $O(|W|)$ in das Weltmodell $W$ eingefügt werden.

## 5.6 Weltmodellierung

Bisher wurde besprochen, wie aus den aktuellen Sensordaten Objekte extrahiert und in das Weltmodell eingefügt werden. In diesem Abschnitt wird nun auf die allgemeine Verwaltung des Weltmodells und die Bedeutung globaler Weltmodellierung eingegangen.

Abbildung 5.6 zeigt die Bildschirmdarstellung des Weltmodells während des Spiels. Man erkennt den auf den eigenen Spieler zurollenden Ball sowie zwei weitere Spieler. Zu erkennen sind auch die aus den Scandaten extrahierten Linien, die noch nicht mit den Linien des Spielfeldmodells in Deckung gebracht wurden.

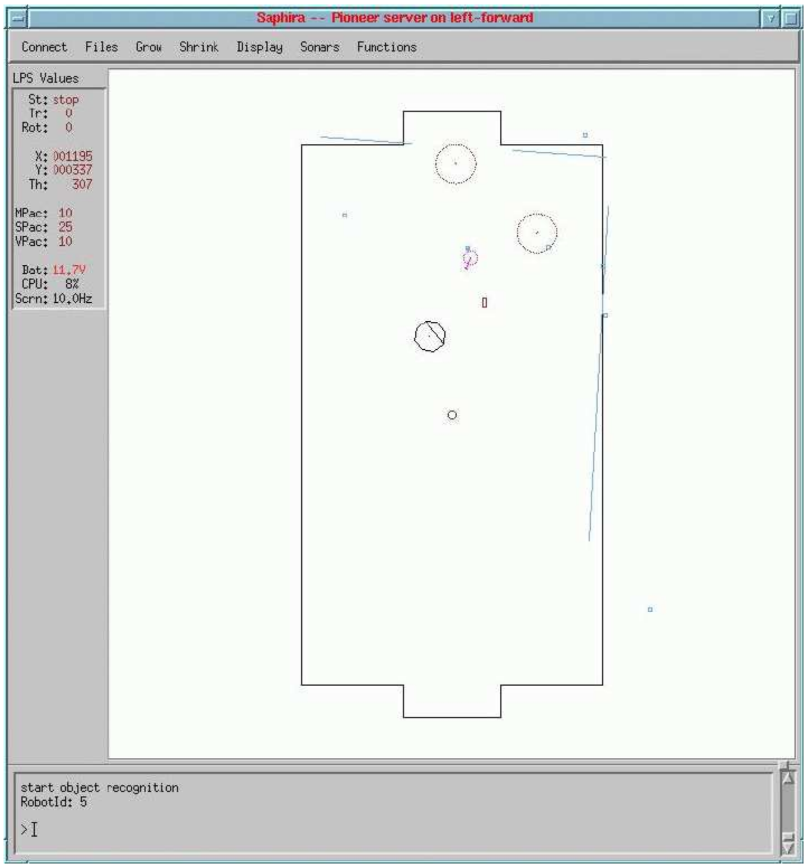



Abbildung 5.6: Die Visualisierung des Weltmodells

### 5.6.1 Weltmodellverwaltung

Liegt zum aktuellen Zeitpunkt für ein bereits im Weltmodell vorhandenen Objekt keine Sensorinformation vor, so wird dessen Zustand zunächst als „erinnert“ markiert und dessen aktuelle Position geschätzt. Erst nach einer maximalen Verweildauer im Weltmodell ohne Sensoraktualisierung wird das Objekt aus dem Weltmodell entfernt. Dies kompensiert zum einen kurze Sensorausfälle und ermöglicht zum anderen die Berücksichtigung des Objekts, auch wenn es sich für kurze Zeit außerhalb des Sensorblickfelds befindet. Dieses Vorgehen ist beispielsweise zum Umfahren von Objekten sehr hilfreich.

Die Aufgabe der Weltmodellverwaltung besteht darin, den Zustand der Objekte aufgrund des Zeitpunkts ihrer letzten Sensorinformation zu markieren. Ein Objekt gilt als „sichtbar", wenn es zum aktuellen Zeitpunkt von einem Sensor wahrgenommen wird. Ansonsten gilt ein Objekt als „erinnert", bis es entweder durch eine neue Sensorinformation aktualisiert wird, oder es eine Mindestverweildauer überschreitet und es als „ungültig" markiert wird. Ungültige Spielermodelle werden schließlich aus dem Weltmodell entfernt.

### 5.6.2 Globale Weltmodellierung

Jeder Spieler teilt dem Zentralrechner in regelmäßigen Abständen sein lokales Weltmodell mit, das dann in das globale Weltmodell integriert wird. Das globale Weltmodell wird wiederum an alle Spieler versendet, die Teile der globalen Information nutzen, um ihre lokale Sicht der Welt zu erweitern.

Informationen über Objekte des globalen Weltmodells werden nach dem gleichen Prinzip wie Sensorinformationen in das lokale Weltmodell eines Spielers eingefügt. Findet sich im lokalen Modell kein Korrespondenzpartner für ein globales Objekt, so wird es neu in das lokale Weltmodell aufgenommen, wobei es allerdings in jedem Fall als „erinnert" markiert wird, da es nicht für die eigenen Sensoren „sichtbar" ist. Findet sich aber ein Korrespondenzpartner, so wird die Position des Korrespondenzobjekts des lokalen Weltmodells nur dann mit der Information des globalen Modells aktualisiert, wenn die Objektinformation des lokalen Modells deutlich älter als die des globalen Modells ist.

Globale Informationen werden also nur verwendet, um nicht vorhandene oder unzuverlässige Informationen des lokalen Weltmodells zu korrigieren. Aktuellen Informationen des lokalen Weltmodells wird stets eine höhere Zuverlässigkeit als globalen Information zugesprochen.

Sowohl für neu eingefügte, als auch für korrespondierende Spielerobjekte des lokalen Weltmodells kann aber in jedem Fall die Identität genauer bestimmt werden: Da jeder Spieler beim Versenden seines Weltmodells seine Identität mitteilt, ist diese im globalen Weltmodell bekannt. Alle Spieler des lokalen Weltmodells, die mit einem Objekt der Identität eines Mitspielers assoziiert werden können, werden daher folgerichtig als Mit- alle anderen Objekte (außer dem Ball) als Gegenspieler gekennzeichnet.

## 5.7 Ergebnisse

Wie in den vorigen Abschnitten gezeigt[2], können Spieler in Zeit $O(|S|)$ aus einem überdeckten Scan extrahiert werden. Berechnet man zu Beginn eines Spiels für jeden Punkt des Pixelbilds[3] die zugehörige Position in der realen Welt und speichert diese in einer Tabelle ab, so reduziert sich die Ballerkennung zur Spielzeit auf ein Tabellenzugriff in $O(1)$ Schritten. Da ein Objekt in Zeit $O(|W|)$ in das Weltmodell eingefügt werden kann, ergibt sich eine Gesamtkomplexität für die Integration neuer Kamera- und Scannerdaten von

$$O(|S| \cdot |W|)$$

Da in der Praxis ca. 20 Meßpunkte für die Kalibrierung der Ballerkennung ausreichen, ein Scan aus maximal 180 Punkten besteht und $W$ nie mehr als 11 Objekte (10 Spieler und den Ball) umfaßt, ist zu erwarten, daß die Weltmodellierung in dem Gesamtsystem keinen zeitkritischen Faktor darstellt. Tatsächlich konnte dies in Messungen bestätigt werden, die auf den in Kapitel 3 vorgestellten Librettos durchgeführt wurden: In durchschnittlich 11 ms können zehn Spieler und der Ball aus den Sensordaten extrahiert und in das Weltmodell eingefügt werden.

Aufgrund der Tatsache, daß bei der Spielererkennung betrachtete Punktwolken durch Reflexionen des Laserstrahls an den Außenseiten der Roboter zustandekommen und damit die Schwerpunkte solcher Punktwolken stark von der Form der Roboter abhängen, ist die Annahme des Punktwolkenschwerpunkts als Mittelpunkt eines Spielermodells lediglich eine Approximation. Abgesehen von diesem systematischen Fehler liefert die Spielererkennung aber verläßliche und exakte Spielerpositionen.

Da der Spielererkennung stets ein überdeckter Scan zugrunde liegt, ist bei der Bestimmung der Spielerpositionen die genaue Referenzposition bekannt. Bei der Ballerkennung besteht hingegen keine Möglichkeit, nachträglich auf die exakte Position zum Zeitpunkt der Bildaufnahme zu schließen. Die Asynchronität der eintreffenden Daten stellt hier ein Problem dar, das vor allem bei schnellen Bewegungen und Drehung des Roboters deutlich wird und zu großen Ungenauigkeiten bei der Bestimmung der Ballposition führt.

Während die Auswirkungen dieses Problems für die Ballposition durch Tiefpaßfilterung der Positionsinformationen bedeutend verringert werden können, sind die Ballgeschwindigkeits- und Richtungsinformationen auch nach Filterung nur bedingt aussagekräftig.

Weiter erschwert wird die Ballmodellierung durch den Umstand, daß sich das gesamte Bildverarbeitungssystem als sehr anfällig gegenüber kleine Schwankungen in den Beleuchtungsverhältnissen oder der Kameraposition zeigt. Die

[2] Auch in diesem Abschnitt bezeichnet $S$ die Menge der Scanlinien, $M$ die Menge der Meßpunkte und $W$ das Weltmodell.

[3] Hierbei handelt es sich um die bereits durch das Bildverarbeitungssystem interpretierten Bilddaten.

Tatsache, daß Schwankungen des Bildverarbeitungssystems im Bereich nur weniger Pixel Positionssprünge in der Größenordnung mehrerer zehn Zentimeter für die Position der realen Welt bedeuten, erklärt weiterhin die hohe Sensibilität und Fehleranfälligkeit bei der Ballmodellierung.

Als sehr vorteilhaft erwies sich die Möglichkeit zur globalen Weltmodellierung, bei der sich vor allem die Tatsache, daß zu fast jedem Zeitpunkt die globale Ballposition bekannt war, spürbar auf die Gesamtperformanz des Teams auswirkte.

# Kapitel 6

# Pfadplanung

Damit ein mobiler Roboter vollständig autonom handeln kann, muß er unter anderem in der Lage sein, selbständig Bewegungen zu planen und auszuführen, um von seiner aktuellen Position zu einem gewünschten Zielort zu gelangen.

Ein explizites Planen von Bewegungssequenzen setzt allerdings ein internes Abbild relevanter Aspekte der Umwelt voraus, wie es beispielsweise in Form des in Kapitel 5 beschriebenen Weltmodells zur Verfügung steht. Auf Grundlage einer solchen Repräsentation der Umwelt können Pfade bestimmt werden, denen ein Roboter zu seinem Ziel folgen kann, ohne mit anderen Objekten unerwünscht zu kollidieren. Eine derartige Trajektorie kann damit als ein Plan aufgefaßt werden, der in jedem Moment Bewegungen des Roboters impliziert, um ihn sicher zu seinem Ziel zu leiten.

Da sich die Umwelt aber im allgemeinen auf unvorhersehbare Weise dynamisch verändert, kann ein Plan schnell an Gültigkeit verlieren. Bereits in der sehr limitierten Welt des Roboterfußballs erscheint es jedoch unmöglich, alle denkbaren zukünftigen Konstellationen der Umgebung in einem Plan zu berücksichtigen. Aus diesem Grund wird die Umwelt in der Regel für einen kurzen Zeitraum als statisch angenommen und ein Plan nur für diesen Zeitraum erstellt. Im nächsten Zeitabschnitt wird dann anhand einer aktuelleren Repräsentation der Umwelt ein neuer Plan bestimmt und damit auf die Veränderungen in der Umwelt reagiert.

In diesem Kapitel werden zunächst einige Grundlagen und Begriffe zur Pfadplanung erläutert. Danach werden verschiedene Ansätze aus der Literatur vorgestellt und der eigene Ansatz bezüglich derselben eingeordnet. Im Anschluß wird das speziell für RoboCup entwickelte Pfadplanungsverfahren beschrieben und auf Besonderheiten und Probleme, die sich beim Pfadverfolgen in der Praxis ergeben, eingegangen. Das Kapitel schließt mit einer Betrachtung der Zeitkomplexität der vorgestellten Algorithmen und einer Besprechung der Ergebnisse.

## 6.1 Grundlagen

Als dem Pfadplanungsprozeß zugrundeliegende Repräsentationsform ist in der Fachwelt das Konzept des *Konfigurationsraums*[1] weit verbreitet. Da dieses Konzept die Repräsentation des Roboters als ein Punkt des Raumes ermöglicht, reduziert sich das Problem, eine Bewegung für ein Objekt mit räumlicher Ausdehnung zu planen, auf die Bewegungsplanung für einen einzelnen Punkt.

Als die *Konfiguration* eines Roboters bezeichnet man die Spezifikation seiner Position $(x, y, \alpha)$. Der *Konfigurationsraum* $C$ enthält alle möglichen Konfigurationen des Roboters – es gilt daher $C \subseteq R^3$. Hindernisse $O_i$ werden repräsentiert, indem man all diejenigen Konfigurationen aus dem Konfigurationsraum ausschließt, bei denen der Roboter in der realen Welt mit einem der Hindernisse kollidieren würde. $C_{free}$ bezeichnet die Teilmenge von $C$ aller *freien Konfigurationen*, die mit keinem Hindernis in Konflikt stehen:

$$C_{free} = \{c \in C | R(c) \cap (\cup O_i) = \emptyset\}$$

Hier bezeichnet $R(c)$ den vom Roboter bei Konfiguration $c$ eingenommenen Raum in der realen Welt (bzw. im Weltmodell).

Nimmt man den Roboter als holonom[2] und dessen Form als kreisrund an, so sind für Konfigurationsübergänge und die Bestimmung der freien Konfigurationen lediglich die Roboterkoordinaten ausschlaggebend und $C_{free}$ kann als Teilmenge von $R^2$ beschrieben werden.

Für das in diesem Kapitel vorgestellte Pfadplanungsverfahren wird $C_{free}$ bestimmt, indem alle Hindernisse um den Radius des Roboters vergrößert und die Ausmaße des Spielfelds um den Roboterradius verringert werden[3]. Der Roboter kann damit als Punkt repräsentiert werden und jede Position im Weltmodell, die nicht mit einem vergrößerten Objekt zusammenfällt und innerhalb der Spielfeldbegrenzung liegt, entspricht einer Konfiguration aus $C_{free}$[4].

Ein mit keinem Hindernis in Konflikt stehender *Pfad* von einer Startkonfiguration $c_{start}$ zu einer Zielkonfiguration $c_{goal}$ kann nun als kontinuierliche Sequenz von Konfigurationen aus $C_{free}$ aufgefaßt werden:

$$\tau : [0, 1] \rightarrow C_{free}$$

Wobei gilt:

$$\tau(0) = c_{start} \quad \text{und} \quad \tau(1) = c_{goal}$$

Eine wünschenswerte Eigenschaft eines solchen Pfades ist sicherlich, daß er eine möglichst kurze Verbindungsstrecke zwischen Start und Ziel darstellt. Wie solche Pfade berechnet werden können, zeigen die folgenden Abschnitte.

[1] Im Englischen auch als *configuration space* bezeichnet.

[2] Sind alle Freiheitsgrade eines Fahrzeugs auch tatsächlich kontrollierbar, so bezeichnet man es als *holonom*.

[3] Diese Vorgehensweise wird auch als *obstacle growing* bezeichnet.

[4] Für die nicht holonomen Pioneer-Roboter wird $C_{free}$ dadurch approximiert.

## 6.2 Ansätze zum Pfadplanen

In der Literatur werden die verschiedenen Ansätze zum Pfadplanen bezüglich einer der Kategorien *Potentialfeldverfahren*, *Zellaufteilungsverfahren* oder *Wegekartenverfahren* klassifiziert [Latombe, 1991]. Im folgenden werden die Grundprinzipien der Ansätze der einzelnen Kategorien kurz erläutert:

**Potentialfeldverfahren**

Bei Potentialfeldverfahren wird der Roboter als ein Punkt betrachtet, der unter dem Einfluß eines künstlichen Potentialfelds steht, das durch die Zielkonfiguration und die Hindernisse impliziert wird. Hierbei erzeugt die Zielkonfiguration ein „anziehendes“ und die Hindernisse ein „abstoßendes“ Potential. An jedem Punkt kann der Gradient des Gesamtpotentials bestimmt werden, dessen Negation als künstliche, auf den Roboter wirkende Kraft, aufgefaßt wird. Für jede Konfiguration wird die Richtung dieser Kraft als die vielversprechendste Bewegungsrichtung betrachtet.

Da bei dieser Art von Verfahren die Gefahr lokaler Minima besteht, müssen allerdings weitere Heuristiken zur Anwendung kommen, um die Pfadsuche vollständig zu machen.

**Zellaufteilungsverfahren**

Bei Zellaufteilungsverfahren wird die Welt durch eine Unterteilung in *Zellen* diskretisiert, so daß ein Pfad zwischen zwei Konfigurationen in der gleichen Zelle leicht generiert werden kann.

Zur Pfadsuche wird ein ungerichteter Graph[5] konstruiert, in dem die Knoten den Zellen entsprechen und zwei Knoten genau dann verbunden sind, wenn die ihnen entsprechenden Zellen aneinander angrenzen und es die Kinematik des Roboters zuläßt, von der Konfiguration der einen Zelle zu der Konfiguration der anderen Zelle zu gelangen. Auf dem Graph kann dann mit einem geeigneten Suchverfahren der kürzeste Pfad von Start- zu Zielpunkt ermittelt werden.

Bei der Zellunterteilung werden exakte und approximierende Verfahren unterschieden: Während exakte Verfahren $C_{free}$ vollständig in Zellen unterteilen, liegt bei approximativen Verfahren nicht jede Konfiguration aus $C_{free}$ in einer Zelle. Aus diesem Grund sind nur exakte Verfahren vollständig. Zellunterteilungsverfahren können universell eingesetzt werden, da mit einer Unterteilung in Zellen keine Annahmen über die Umgebung gemacht werden. Da die Suchdauer und der benötigte Speicherplatz durch die Zellanzahl bestimmt werden, sind der Größe der Zellen und der Größe der Umgebung allerdings Grenzen gesetzt.

**Wegekartenverfahren**

Bei Wegekartenverfahren werden die Bewegungsmöglichkeiten im freien Konfigurationsraum $C_{free}$ auf ausgesuchte Wege begrenzt, die als ein Netzwerk von Verbindungsstrecken in einem ungerichteten Graphen re-

[5] Ein solcher Graph wird auch als *connectivity graph* bezeichnet.

präsentierbar sind. Das Pfadplanungsproblem reduziert sich damit wiederum auf ein Graphsuchproblem, bei dem diejenige Streckensequenz ermittelt werden muß, die Start- und Zielpunkt verbindet und dabei eine minimale Gesamtlänge aufweist.
Neben verschiedenen anderen Methoden, eine Wegekarte zu erstellen, ist das Prinzip des *Sichtbarkeitsgraphen* für zweidimensionale Konfigurationsräume mit polygonalen Hindernissen weit verbreitet. Ein Sichtbarkeitsgraph ist ein ungerichteter Graph, dessen Knoten der Start- und Zielpunkt sowie alle Ecken der Hindernisse sind. Die Kanten des Graphs sind all diejenigen Verbindungsgeraden zwischen zwei Knoten, die kein Hindernis schneiden.
Da Speicherplatzbedarf und Suchdauer für Wegekartenverfahren lediglich von der Anzahl der Hindernisse abhängen, erzielen diese Verfahren auch in großflächigen Umgebungen schnelle Suchzeiten. Die Tatsache, daß eine polygonale Beschreibung der Umwelt oft nicht einfach generierbar ist, schränkt deren universelle Anwendbarkeit aber deutlich ein.

Neuere Ansätze [Thrun und Bücken, 1996; Gutmann und Nebel, 1997] versuchen, die Vorteile von Zellaufteilung und Wegekarten zu kombinieren, indem sie global mittels Wegekarten und lokal mittels Zellaufteilung zu planen versuchen. Abbildung 6.1 zeigt ein Beispiel eines die Umwelt in regelmäßige Zellen

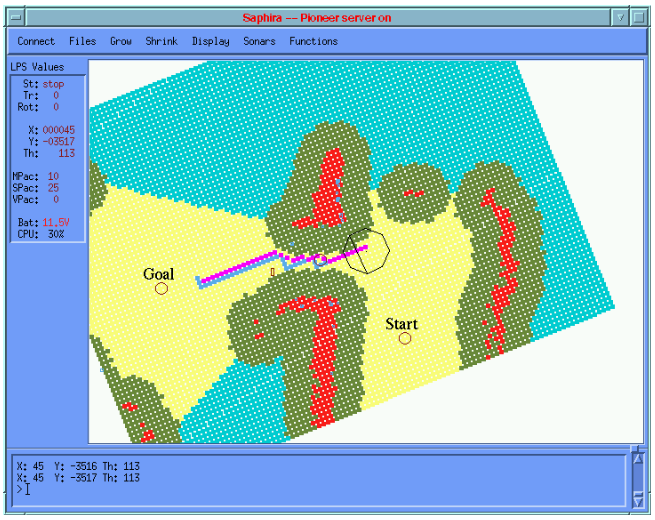



Abbildung 6.1: Ein Gitterplaner zur lokalen Wegeplanung

aufteilenden Gitterplaners, der zur Roboternavigation zwischen zwei Zwischenpunkten eines topologischen Plans verwendet wird [Gutmann und Nebel, 1997].

Wegen der einfachen geometrischen Beschreibung der Roboterfußballwelt sowie den Echtzeit- und Genauigkeitsanforderungen an das Planungsverfahren wurde für RoboCup ein Planalgorithmus implementiert, der auf dem Prinzip eines *erweiterten Sichtbarkeitsgraphen* basiert: Wege durch $C_{free}$ können in diesem Graph zusätzlich zu den Geradenstücken auch entlang von Kreisbögen an den durch Kreise approximierten Hindernissen verlaufen.

## 6.3 Erweiterter Sichtbarkeitsgraph

Ein *Graph* $G = (V, E)$ besteht aus einer Menge von Knoten $V$ und einer Menge von Kanten $E$, die jeweils zwei Knoten aus $V$ verbinden. Ist für jede Kante eine Richtung festgelegt, so spricht man von einem *gerichteten Graph*, ansonsten von einem *ungerichteten Graph*.

Das in diesem Kapitel beschriebene Pfadplanungsverfahren basiert auf einem ungerichteten *erweiterten Sichtbarkeitsgraphen* $S$, der auf Basis des Weltmodells $W$ erzeugt wird. $W$ enthalte $n$ Objekte. Dann bezeichnet $K_i$ $(i \in [1, n])$ eine Punktmenge, die alle Randpunkte desjenigen Kreises enthält, der das i-te Objekt des Weltmodells repräsentiert. Der Radius eines Kreises sei durch $r_i$ gegeben.

Der erweiterte Sichtbarkeitsgraph $S = (V, E)$ ist wie folgt definiert:

1. Der Startpunkt $p_{start}$ und der Zielpunkt $p_{goal}$ sind Knoten in $V$. Es wird ein $K_{n+1}$ mit einzigem Element $p_{start}$ und ein $K_{n+2}$ mit einzigem Element $p_{goal}$ erzeugt.

2. Seien $K_i$ und $K_j$ $(i, j \in [1, n+2]$ $(i \neq j))$ zwei Punktmengen. Falls zwei Punkte $p_1 \in K_i$ und $p_2 \in K_j$ existieren, so daß deren Verbindungsgerade vollständig in $C_{free}$ liegt und in $p_1$ bzw. $p_2$ eine Tangente an $K_i$ bzw. an $K_j$ ist, dann sind $p_1$ und $p_2$ Punkte in $V$ und deren Verbindungsgerade eine Kante in $E$. Im allgemeinen existieren 4 solche Geraden.

3. Existiert für zwei Punkte des gleichen Kreises $p_1, p_2 \in V$ kein $p \in V$, das zwischen $p_1$ und $p_2$ auf diesem Kreis liegt, so ist der $p_1$ und $p_2$ verbindende Kreisbogen eine Kante in $E$.

Im folgenden wird gezeigt, wie die oben beschriebenen Tangenten und Kreispunkte bestimmt werden. Hierzu werden die zwei Fälle $r_1 \neq r_2$ und $r_1 = r_2$ unterschieden.

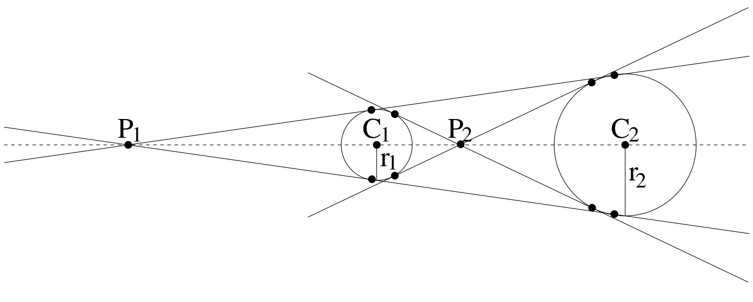



Abbildung 6.2: Die Tangentenbestimmung für zwei Kreise mit $r_1 \neq r_2$

Sei zunächst $r_1 \neq r_2$. Abbildung 6.2 veranschaulicht diesen Fall, in dem die Tangenten mit Hilfe des Strahlensatzes und einer allgemeinen Tangentengleichung ermittelt werden können. Da hierfür die Koordinaten der Tangentenschnittpunkte bekannt sein müssen, werden zunächst die Strecken $|P_1C_2|$ und $|P_2C_2|$ bestimmt:

Die Srecke $|P_1C_2|$ läßt sich wie folgt berechnen:

$$\begin{aligned}
\frac{|P_1C_2|}{|P_1C_1|} &= \frac{r_2}{r_1} \\
|P_1C_1| &= |P_1C_2| - |C_1C_2| \\
\Longrightarrow |P_1C_2| &= \frac{|C_1C_2|r_2}{r_2 - r_1}
\end{aligned}$$

Die Strecke $|P_2C_2|$ errechnet sich aus:

$$\begin{aligned}
\frac{|P_2C_2|}{|P_2C_1|} &= \frac{r_2}{r_1} \\
|P_2C_1| &= |C_1C_2| - |P_2C_2| \\
\Longrightarrow |P_2C_2| &= \frac{|C_1C_2|r_2}{r_2 + r_1}
\end{aligned}$$

Mit den berechneten Streckenlängen lassen sich nun die Koordinaten von $P_1$ und $P_2$ nach folgender Gleichung mit $i \in \{1, 2\}$ bestimmen:

$$\begin{aligned}
\tan\alpha &= \frac{C_{1_y} - C_{2_y}}{C_{1_x} - C_{2_x}} \\
P_{i_x} &= C_{2_x} - \cos\alpha |P_iC_2| \\
P_{i_y} &= C_{2_y} - \sin\alpha |P_iC_2|
\end{aligned}$$

Die Kreisgleichung für ein $K_i$ $(i \in \{1, 2\})$ lautet:

$$(x_i - C_{i_x})^2 + (y_i - C_{i_y})^2 = r_i^2$$

Eine Tangente durch $P_j$ berührt den Kreis $K_i$ $(i, j \in \{1, 2\})$ in $(x_i, y_i)$ wenn gilt [Sieber und Huber, 1980]:

$$(x_i - C_{i_x})(P_{j_x} - C_{i_x}) + (y_i - C_{i_y})(P_{j_y} - C_{i_y}) = r_i^2 \tag{6.1}$$

Auflösen der beiden letzten Gleichungen nach $x$ und $y$ führt zu einer quadratischen Gleichung, die maximal zwei Lösungen hat. Für jede Wahl von $i$ und $j$ erhält man somit maximal zwei Punkte, die die Berührpunkte der Tangente durch den Punkt $P_j$ an den Kreis $K_i$ sind.

Sei nun $r_1 = r_2$. Wie Abbildung 6.3 veranschaulicht, exisitiert in diesem Fall kein Schnittpunkt $P_1$, da die entsprechenden Tangenten parallel verlaufen. Während die Kreisberührpunkte der Tangenten durch $P_2$ wie im ersten Fall mittels Gleichung 6.1 bestimmt werden müssen, lassen sich die Berührpunkte

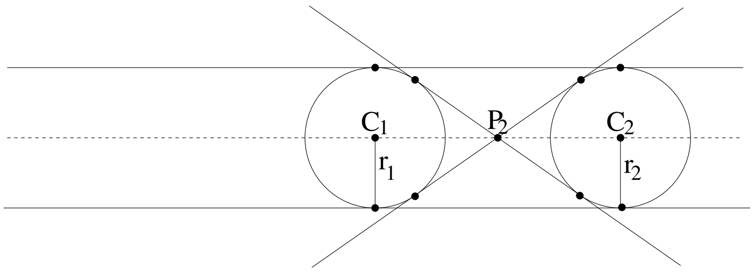



Abbildung 6.3: Die Tangentenbestimmung für zwei Kreise mit $r_1 = r_2$

der parallelen Tangenten direkt mit den folgenden Gleichungen für $i \in \{1, 2\}$ berechnen:

$$\begin{aligned}
\tan \alpha &= \frac{C_{1_y} - C_{2_y}}{C_{1_x} - C_{2_x}} \\
x_i &= C_{i_x} + r\cos(\alpha \pm \frac{\pi}{2}) \\
y_i &= C_{i_y} + r\sin(\alpha \pm \frac{\pi}{2})
\end{aligned}$$

Die mit den vorgestellten Gleichungen ermittelten Tangenten werden nun gemäß der eingangs beschriebenen Definition in den (erweiterten) Sichtbarkeitsgraphen eingefügt. Abbildung 6.4 zeigt ein Beispiel für einen solchen Graphen, auf dem

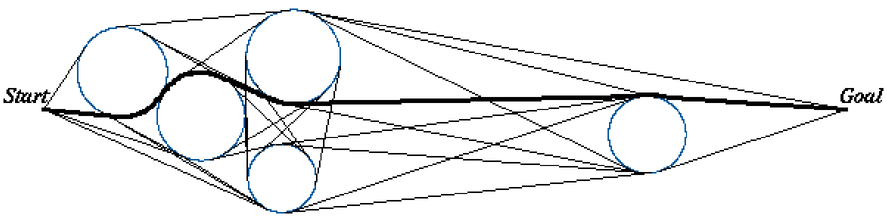



Abbildung 6.4: Der Sichtbarkeitsgraph mit kürzestem Pfad.

bereits der kürzeste Pfad zwischen Start- und Zielpunkt eingezeichnet ist. Wie dieser ermittelt wird, ist Thema des nächsten Abschnitts.

## 6.4 $A^*$-Suche

Sei $S = (V, E)$ der wie im vorigen Abschnitt erzeugte Sichtbarkeitsgraph. Das Problem der Pfadsuche kann nun als Suche nach einer Sequenz von Kanten in $E$ beschrieben werden, wobei sich jeweils zwei Kanten in einem Knoten aus $V$ treffen, eine Kante mit dem Start- sowie eine mit dem Zielpunkt verbunden und die Gesamtsumme der Kantenlängen minimal ist.

Für die Suche bietet sich die Verwendung des $A^*$*-Algorithmus* an, der in jedem Verarbeitungsschritt die Länge des kürzesten Pfades von Start- zu Zielknoten nach folgendem Prinzip schätzt: Bezeichnet $g(n)$ die Länge eines Pfades[6] von Startknoten zu Knoten $v_n$ und $h(n)$ die geschätzte Pfadlänge von Knoten $v_n$ zum Zielknoten, so beschreibt $f(n)$ die geschätzte Länge des kürzesten Pfades durch den Knoten $v_n$:

$$f(n) \quad = \quad g(n) + h(n)$$

Bei Pfadplanungsproblemen wird für die Funktion $h(n)$ in der Regel eine Heuristik gewählt, die die Länge der direkten Verbindungsstrecke von dem Knoten $v_n$ zum Zielknoten berechnet.

Ausgehend von dem Startknoten werden während der Suche schrittweise alle Knoten expandiert, die von dem jeweils aktuellen Knoten erreichbar sind. Die expandierten Knoten werden in eine Warteschlange sortiert nach aufsteigenden $f(n)$-Werten eingefügt und jeweils der erste Knoten in der Warteschlange ausgewählt, um weiter expandiert zu werden. Das Verfahren endet, sobald der Zielknoten erster Knoten in der Warteschlange ist.

Mit der oben beschriebenen Heuristik $h(n)$ garantiert der $A^*$-Algorithmus Vollständigkeit und Optimalität [Hart *et al.*, 1968] und zeichnet sich für den vorliegenden Problemtyp durch hohe Effizienz aus.

## 6.5 Der Pfadplanungsalgorithmus

Da bei Wegekartenverfahren die Erzeugung des Graphen am zeitkritischsten ist, wurde bei dem in diesem Abschnitt vorgestellten Verfahren ein iterativer Ansatz verwendet, der den Graph nur soweit wie nötig erzeugt.

Abbildung 6.5 zeigt den prinzipiellen Ablauf des Planvorgangs. Beginnend mit einem lediglich aus Start- und Zielknoten bestehenden Graph wird ein optimaler Pfad geplant und überprüft, ob der Pfad mit keinem Objekt des Weltmodells in Konflikt steht. Besteht ein Konflikt, so wird der Graph um das Objekt erweitert und nochmals geplant. Das Verfahren iteriert solange, bis entweder ein

[6] Diese ist durch die Summe der Längen der zugehörigen Kanten gegeben.

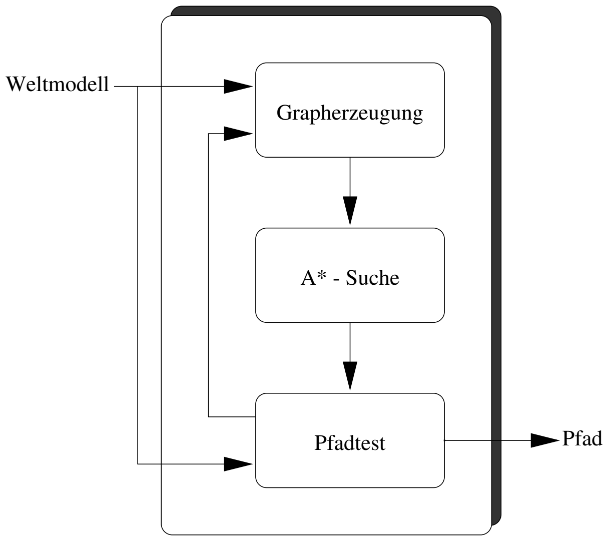



Abbildung 6.5: Das Pfadplanungsmodul

konfliktfreier Pfad gefunden, oder überhaupt kein Pfad zu dem gewünschten Zielpunkt gefunden werden konnte.

Im folgenden wird das Verfahren algorithmisch beschrieben:

**Algorithmus 6.1** *PlanPath(O, B, (x, y))*
**Eingabe:** Objektmodellmenge $O$, Randlinien $B$, Zielpunkt $(x, y)$
**Ausgabe:** Pfad $P$
**Ablauf:**

$G := (\{\}, \{\})$
$G := AddNodes(G, B, (RobotX(), RobotY(), 0))$
$o := (x, y, 0)$
**while** $o \neq NULL$ **do**
  $G := AddNodes(G, B, o)$
  $P := SearchPath(G)$
  **if** $P = NULL$ **then**
    **return** $NULL$
  $o := CollisionObject(P, O)$
**return** $P$

Die Funktion *AddNodes(G, B, c)* berechnet für alle bereits im Graphen $G$ vorhandenen Objektmodelle die Tangenten an das durch einen Kreis $(x, y, r)$ repräsentierte Objekt $o$ und fügt unter Beachtung der Randlinien $B$ entsprechende Kanten und Knoten in $G$ ein. *SearchPath(G)* startet eine A*-Suche nach dem kürzesten Pfad von Start- zu Zielknoten des Graphen $G$. Die Funktion *CollisionObject(P, O)* überprüft, ob ein Objektmodell der Menge $O$ mit dem Pfad $P$ in Konflikt steht und gibt gegebenenfalls das nächste solche Objekt zurück. Existiert kein solches Objekt, ist der Rückgabewert $NULL$.

## 6.6 Pfadabfahren in der Praxis

In der Realität kann selten einem idealen Pfad vollständig gefolgt werden. Dies liegt vor allem daran, daß die durch das Weltmodell gegebene Repräsentation der Welt stets dem aktuellen Zustand hinterherhinkt. Wegen der hohen Dynamik der Umgebung und der Tatsache, daß ein Roboter nicht mit beliebiger Präzision steuerbar ist, sind in der Praxis daher sowohl Verfeinerungen an dem Pfadplanungsalgorithmus, als auch an der Pfadabfahrstrategie notwendig.

Ein bedeutendes Problem tritt auf, wenn sich in kurzen Zeitabständen jeweils verschiedene Pfade als die kürzeste Alternative bieten. Das daraus resultierende oszillierende Verhalten wird in der Praxis vermieden, indem die Kosten eines Pfades, der zunächst ein Drehen des Roboters notwendig macht, abhängig vom Betrag des Winkels erhöht werden.

Obwohl theoretisch ausgeschlossen, entsteht in der Praxis häufig die Situation, daß sich der Roboter innerhalb eines einem Objekt entsprechenden Kreises befindet. In einem solchen Fall muß der Radius des Objektmodells so verkleinert werden, daß sich der Roboter außerhalb des Kreises befindet und ein Planen überhaupt möglich ist.

Bei häufigem Neuplanen kann hierbei ein weiteres Problem entstehen: Durch Ungenauigkeiten beim Abfahren eines Pfades kann sich der Roboter einem Hindernis immer mehr annähern, wobei der Radius ständig verkleinert werden muß, so daß der Roboter schließlich mit dem Objekt kollidiert. Für den Einsatz bei RoboCup wurde hierfür eine pragmatische Lösung gewählt, die den Roboter in solchen Fällen zunächst zurücksetzen läßt, damit dann aus sicherer Entfernung neu geplant werden kann.

Die Festlegung der Menge der freien Konfigurationen $C_{free}$ hat entscheidenden Einfluß darauf, ob überhaupt ein Pfad gefunden werden kann und wie groß beim Befahren des Pfades die Wahrscheinlichkeit für Kollisionen mit Objekten ist. Für den Betrag des *obstacle growings* muß daher ein angemessener Kompromiß gefunden werden, der gewährleistet, daß einerseits freie Pfade stets auch gefunden werden und Objekte dennoch in sicherem Abstand umfahren werden können.

Obwohl die sorgfältige Auswahl eines *obstacle growing*-Betrages zuverlässige Ergebnisse lieferte, ist es wünschenswert, den Ansatz so zu erweitern, daß zwischen den verschiedenen Anforderungen an einen Pfad jeweils situationsangemessen abgewogen werden kann. Eine naheliegende Erweiterung liegt in der Verwendung kleiner Beträge für das *obstacle growing*, um dann zusätzlich Methoden wie den *dynamic window approach* zur lokalen Hindernisvermeidung einzusetzen [Fox *et al.*, 1997].

## 6.7 Ergebnisse

Bei der Erzeugung des erweiterten Sichtbarkeitsgraphen werden für alle Objekte paarweise die Tangenten zwischen zwei Objektkreisen bestimmt. Da zwischen zwei Kreisen jeweils maximal vier Tangenten existieren, kann die Anzahl der Kanten des Graphs daher mit

$$\frac{4|W|(|W|-1)}{2} \quad = \quad O(|W|^2)$$

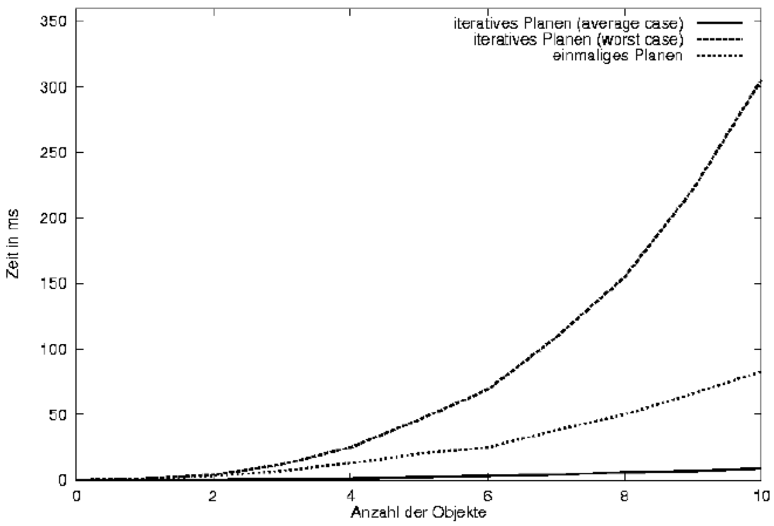
abgeschätzt werden, wobei $|W|$ die Anzahl der Objekte des Weltmodells bezeichnet. Wie Abbildung 6.6 zeigt, konnte diese Abschätzung in Messungen bestätigt werden, die auf den in Kapitel 3 vorgestellten Librettos durchgeführt wurden: Hierbei wurden in drei unterschiedlichen Messungen jeweils die Laufzeiten über tausend Durchläufe gemittelt. Die Meßdaten für einmaliges Planen geben Aufschluß darüber, wieviel Zeit der Planer im Durchschnitt benötigt, um in einem Schritt einen Graph mit $o$ Objekten zu erzeugen und in diesem den kürzesten Pfad von Start- zu Zielknoten zu finden. Diese Zeit $t$ kann tatsächlich durch eine quadratische Funktion $t = 0.8 \cdot o^2$ approximativ vorhergesagt werden.

Die tatsächlichen Planzeiten des iterativen Planers wurden in den beiden folgenden Messungen für den schlechtesten („worst case") und den durchschnittlichen Fall („average case") bestimmt: Um die Laufzeit im schlechtesten Fall zu ermitteln, wurde jeweils ein Weltmodell konstruiert, das bei iterativem Planen die Integration aller Objekte in den erweiterten Sichtbarkeitsgraphen erzwingt. Da in diesem Fall $o$-mal ein Graph mit bis zu $o$ Objekten erzeugt wird, spiegeln die Meßwerte ein deutlich schlechteres Laufzeitverhalten wieder.

In der Regel kann das Eintreten des schlechtesten Falls aber als sehr unwahrscheinlich angesehen werden, was die mit zufällig erzeugten Weltmodellen[7] durchgeführten Messung des durchschnittlichen Falls belegt: Die mittleren Laufzeiten des iterativen Planers für maximal $o$ Objekte liegen deutlich unter denen für einen einmaligen Planvorgang mit $o$ Objekten.

Die Tatsache, daß selbst für zehn Objekte im Durchschnitt nur 2-3 Objekte in den Graph eingefügt und lediglich 9 Millisekunden für den Planvorgang benötigt werden, unterstreicht die Effizienz des Verfahrens und begründet, warum der Ansatz als geeignet angesehen wird, in Echtzeit planend auf sich ändernde Umweltbedingungen zu reagieren.

[7]Hierbei wurden die Positionen und Größen der Objekte mit einer uniformen Wahrscheinlichkeitsverteilung zufällig gewählt.




| Objekte | Laufzeiten | | |
|---|---|---|---|
| | einmaliges Planen | iteratives Planen | |
| | | worst case | average case |
| 0 | 0ms | 0ms | 0ms |
| 1 | 1ms | 1ms | 0ms |
| 2 | 3ms | 4ms | 0ms |
| 3 | 7ms | 12ms | 1ms |
| 4 | 13ms | 25ms | 1ms |
| 5 | 20ms | 46ms | 2ms |
| 6 | 25ms | 69ms | 3ms |
| 7 | 38ms | 109ms | 4ms |
| 8 | 50ms | 155ms | 6ms |
| 9 | 66ms | 222ms | 7ms |
| 10 | 83ms | 306ms | 9ms |

Abbildung 6.6: Die Ergebnisse der Laufzeitmessungen

# Kapitel 7

# Grundfertigkeiten

In diesem Kapitel werden grundlegende Fertigkeiten vorgestellt, deren kombinierte oder sequentielle Ausführung komplexe Verhaltensmuster ergeben, mit denen auf die gerade aktuelle Spielsituation reagiert werden kann. Zunächst wird beschrieben, welche elementaren Steuerbefehle als Basis für die Entwicklung der Grundfertigkeiten zur Verfügung standen. Im Anschluß wird dann gezeigt, wie diese mit den in den vorigen Kapiteln vorgestellten Techniken zur Weltmodellierung und Pfadplanung zu grundlegenden Verhaltensbausteinen kombiniert werden.

## 7.1 Elementare Steuerung

Unter anderem stellt das *Saphira-System* [Konolige *et al.*, 1997] die Möglichkeit zur Verfügung, einen gewünschten Roboterzustand bezüglich Translationsgeschwindigkeit, Rotationsgeschwindigkeit und Orientierung zu spezifizieren. Das Betriebssystem sendet daraufhin die notwendigen Kommandos an den Roboter, um den jeweils aktuellen Zustand in den gewünschten zu überführen. Für die Implementierung der im folgenden beschriebenen Fertigkeiten wurde ausschließlich von dieser Form der Robotersteuerung Gebrauch gemacht – insbesondere wurden folgende Befehle verwendet:

- *sfSetVelocity* – setzt die gewünschte Translationsgeschwindigkeit
- *sfSetRVelocity* – setzt die gewünschte Rotationsgeschwindigkeit
- *sfSetDHeading* – setzt die gewünschte relative Orientierung

Bei der Berechnung der Geschwindigkeiten wurde in vielen Fällen von zwei weiteren durch das Saphira-System bereitgestellte Funktionen Gebrauch gemacht, die einen Wert im Intervall [0,1] berechnen[1]:

[1] Funktionen mit diesem Wertebereich werden auch als *Fuzzy-Funktionen bezeichnet.*

- *up_straight(float x, float min, float max)*
- *straight_down(float x, float min, float max)*

Abbildung 7.1 zeigt die beiden Funktionen.

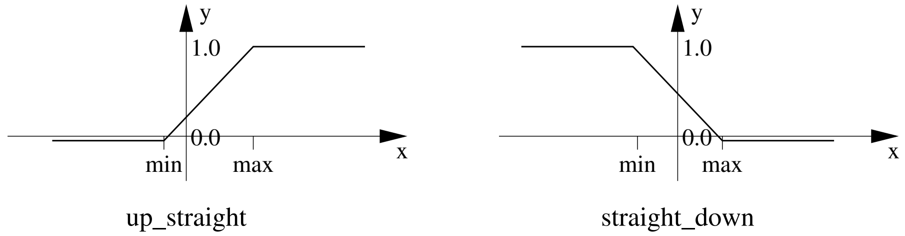



Abbildung 7.1: Die Funktionen up_straight und straight_down

## 7.2 Spiel ohne Ball

In diesem Abschnitt werden Fertigkeiten vorgestellt, die verwendet werden, um einen Roboter entsprechend der Spielsituation angemessen auf dem Spielfeld zu bewegen, ohne dabei direkt den Ball zu spielen. Notwendig sind hierzu Verhalten, um den Roboter eine gewünschte Position einnehmen zu lassen, um ihn entlang geplanten Pfaden über das Spielfeld zu bewegen und um ihn Ball oder Spielfeld observieren zu lassen.

### 7.2.1 Positionierung

In vielen Situationen ist es notwendig, den Roboter zu einer gewünschten Position zu bewegen. Dies leistet die Funktion *ApproachPosition*, die solange mittels der Funktion *SetApproachVelHead* geeignete Translations- und Orientierungszustände definiert, bis der gewünschte Zielpunkt erreicht ist.

**Algorithmus 7.1** *ApproachPosition(p)*
**Eingabe:** Position $p$
**Ablauf:**
  **while** $FrontClear()$ **and** $(DistanceRobotToPos(p) >$ MAX-D
    **or** $AngleRobotToPos(p) >$ MAX-A) **do**
    $SetApproachHeadVel(p)$

Hierbei überprüft die Funktion *FrontClear*, ob sich vor dem Roboter nicht ein Hindernis befindet. Das gewünschte Ziel gilt als erreicht, wenn sowohl die Distanz, als auch die Winkeldifferenz zwischen aktueller Roboterposition und Zielposition klein genug ist.

Die Funktion *SetApproachVelHead* bestimmt für eine gegebene Position $p$ die Distanz und Winkeldifferenz zur aktuellen Roboterposition und berechnet hieraus eine geeignete Translationsgeschwindigkeit, um die Position $p$ so schnell wie möglich zu erreichen – allerdings ohne über die Position hinauszuschießen:

**Algorithmus 7.2** *SetApproachVelHead(p)*
**Eingabe:** Position $p$
**Ablauf:**
$d := DistanceRobotToPoint(p)$
$\alpha := AngleRobotToPos(p)$
$tVel := up\_straight(d, \text{MIN-D}, \text{MAX-D}) \cdot \text{MAX-TVEL}$
$tVel := straight\_down(fabs(\alpha), \text{MIN-A}, \text{MAX-A}) \cdot tVel$
$sfSetVelocity(tVel)$
$sfSetDHeading(\alpha)$

Die Translationsgeschwindigkeit wird zum einen durch die Distanz des Roboters zum Ziel und zum anderen durch die Winkeldifferenz zwischen tatsächlicher und gewünschter Roboterorientierung bestimmt: je kleiner die Distanz und je größer die Winkeldifferenz ist, umso langsamer bewegt sich der Roboter. Die Parameter der Fuzzy-Funktionen konnten so justiert werden, daß sich ein Spieler z.B. bis auf nur wenige Zentimeter dem Ball nähern kann, ohne ihn zu berühren. Eine derart exakte Ballkontrolle hat sich als äußerst vorteilhaft für eine Reihe von Spielzügen erwiesen, die in Kapitel 8 vorgestellt werden.

### 7.2.2 Navigation

Damit ein Spieler auch beim Zurücklegen größerer Distanzen kollisionsfrei bei einer gegebenen Zielposition angelangt, werden explizit Pfade geplant, die ihn auf kürzest möglichem Wege, vorbei an Hindernissen, zu seinem Ziel führen. Die hohe Dynamik des Roboterfußballs macht es jedoch erforderlich, in jedem Verarbeitungsschritt durch Neuplanen auf die sich stetig ändernden Umweltbedingungen zu reagieren. Die Funktion *GoToPosition* plant in jedem Zyklus den kürzesten Pfad zum Ziel und veranlaßt geeignete Kommandos, um dem Pfad zu folgen.

**Algorithmus 7.3** *GoToPosition(p)*
**Eingabe:** Position $p$
**Ablauf:**
**while** $DistanceRobotToPos(p) > \text{MIN-D}$ **do**
$path := PlanPath(CurrentWorldModel(), BorderLines(), p)$
**if** $path \neq NULL$ **then**
$c := PathClosestPoint(path)$
$r := PathCurrentRadius(path, c)$
$c' := PathProjectPoint(path, c, r)$
$SetApproachVelHead(c')$

Nachdem mittels *PlanPath* ein Pfad gefunden wurde, liefert die Funktion *PathClosestPoint* den zum Roboter nächstgelegenen Punkt $c$ auf dem Pfad. Dieser Punkt befindet sich auf einem Pfadsegment mit charakteristischem Radius $r$, wobei für Geradensegmente ein vorgegebener Maximalwert verwendet wird. Durch die Funktion *PathProjectPoint* wird der Punkt $c$ nun um den Betrag $r$ in Richtung des Pfades projiziert, womit ein Referenzpunkt $c'$ bestimmt ist, der als Zielpunkt für die Funktion *SetApproachVelHead* dient.

Da die Distanz von $c$ zu $c'$ in Abhängigkeit des Radius des aktuellen Pfadsegments bestimmt wird und *SetApproachVelHead* die Geschwindigkeit in Abhängigkeit von der Entfernung zum Zielpunkt setzt, werden „enge“ Kreisbögen vorsichtiger und genauer abgefahren als gerade lange Strecken. Die Art und Weise, wie einem Pfad gefolgt wird, paßt sich also der Charakteristik des jeweils befahrenen Pfadsegments an.

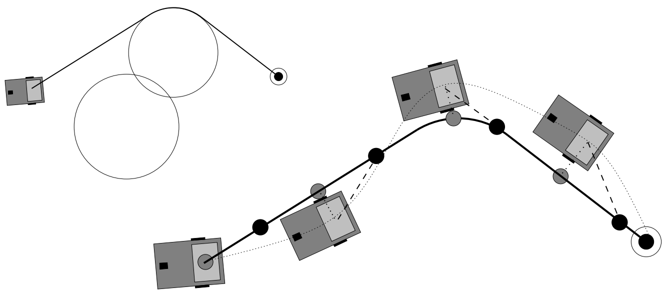


Abbildung 7.2: Das Pfadabfahren mittels *GoToPosition*

Wie ein Pfad in der Realität abgefahren wird, zeigt Abbildung 7.2: die grauen Punkte zeigen den jeweils zum Roboter nächsten Punkt auf dem Pfad, die schwarzen Punkte zeigen die entlang des Pfades projizierten Zielpunkte für die Funktion *SetApproachVelHead*. Abweichungen des Roboters vom Pfad werden so ausgeglichen und ein „glattes“ Befahren des Pfades gewährleistet.

### 7.2.3 Observieren

Während des Spiels treten immer wieder Situationen auf, in denen der Ball nicht sichtbar ist, oder sich in einem Bereich des Spielfelds befindet, den der Spieler aufgrund einer globalen Taktik nicht befahren sollte. In diesen Fällen können die im folgenden beschriebenen Fertigkeiten eingesetzt werden, um den Spieler angemessen auf dem Spielfeld zu bewegen.

#### Ballobservation

Verlangt eine globale Taktik von einem Spieler, den Ball nicht zu spielen, so versucht er, eine taktisch gute Position einzunehmen und den Ball zu observieren, um sofort reagieren zu können, wenn der Ball in einen bespielbaren Bereich gelangt. Die Funktion *ObserveBall* verwirklicht dies auf einfache Weise:

**Algorithmus 7.4** *ObserveBall*
**Ablauf:**
$GoToPosition$(OBSERVE-BALL-POS)
**forever do**
$\alpha := AngleRobotToPos(BallPosition())$
$sfSetDHeading(\alpha)$

### Ballsuche

Ist einem Spieler die Position des Balls nicht bekannt, so „sucht“ er nach ihm, indem er sich an der aktuellen Position um die eigene Achse dreht. Hierbei wird in diejenige Richtung gedreht, in der der Ball zuletzt wahrgenommen wurde. Ist der Ball auch nach einer kompletten Umdrehung nicht sichtbar, so wird die Suche erfolglos abgebrochen:

**Algorithmus 7.5** *SearchBall*
**Ablauf:**
$tDir := (AngleRobotToPos(BallPosition()) > 0)\ ?\ 1 : -1$
$\alpha := 360°$
$\alpha Old := RobotHeading()$
**while not** $BallVisible()$ **and** $\alpha > 5°$ **do**
$\alpha New := RobotHeading()$
$\alpha := \alpha - AngleDifference(\alpha Old, \alpha New)$
$sfSetDHeading(tDir \cdot 5°)$
$\alpha Old := \alpha New$

### Feldobservation

In Fällen, in denen kein Verhalten sinnvoll oder erfolgversprechend erscheint, kommt ein „Standardverhalten“ zum Einsatz, mit dem das Feld von einer Standardposition aus observiert wird. Falls sich der Spieler nahe des Spielfeldrandes befindet, so versucht er durch Schwenken über einen bestimmten Winkelbereich (zwischen TH-L und TH-R) neue Informationen zu erhalten. Ist er weiter von dem Spielfeldrand entfernt, so rotiert er zu diesem Zweck um die eigene Achse:

**Algorithmus 7.6** *ObserveField*
**Ablauf:**
**forever do**
**if** $DistanceRobotToPoint$(OBSERVE-FIELD-POS) $> MIN$ **then**
$GoToPosition$(OBSERVE-FIELD-POS)
**if** $DistanceRobotToBorder() > MIN$ **then**
$sfSetRVelocity(RVEL)$
**else**
$PanAngle$(TH-L, TH-R)

## 7.3 Spiel mit Ball

Befindet sich ein Spieler in der Nähe des Balls, so stehen ihm zwei Spielvarianten zur Verfügung, die in diesem Abschnitt beschrieben werden. Ist die Distanz zum gegnerischen Tor relativ groß, so wird zunächst mit dem Ball zum gegnerischen Tor gedribbelt. Befindet sich der Ball aber nahe des gegnerischen Tors, des Spielfeldrandes oder eines Objekts, so kommt eine geeignete Schußvariante zum Einsatz.

### 7.3.1 Ballmanipulation

Ist ein Spieler in Ballbesitz zu weit weg vom gegnerischen Tor, so startet er einen Angriff, indem zunächst von der aktuellen Roboterposition diejenige Linie zu einem Punkt im Tor ausgewählt wird, die zu jedem Hindernis auf dem Feld den größten Abstand hat. Dieser Linie folgt der Roboter nun mit maximaler Beschleunigung, bis er entweder wegen eines Hindernisses abbrechen muß oder sich dem gegnerischen Tor genügend genähert hat, um mit Aussicht auf Erfolg zu schießen. Damit auf sich ändernde Umweltbedingungen (z.B. auf die sich ändernde Position des Torwarts) reagiert werden kann, wird die Linie zum Tor bei Bedarf neu gewählt:

**Algorithmus 7.7** *PlayBall*
**Ablauf:**

```
p := BestPointInGoal()
while FrontClear() and TooFar(p) do
    if NewChoice() then
        p := BestPointInGoal()
    sfSetVelocity(MAXTVEL)
    a = AngleRobotToPos(p)
    sfSetDHeading(a)
ShootBall()
```

Hier bewertet die Funktion *TooFar*, ob der Roboter noch zu weit vom Tor entfernt ist, um den Ball zu schießen. Diese Entscheidung wird unter anderem auch von der Geschwindigkeit des Roboters abhängig gemacht, damit bei hohen Robotergeschwindigkeiten die bereits auf den Ball ausgeübte Beschleunigung ausgenutzt werden kann.

### 7.3.2 Schießen

Mit der aktuell verwendeten Hardware kann der Ball auf zwei verschiedene Arten „geschossen“ werden: Zum einen kann dem Ball durch schnelles Drehen des Roboters ein Impuls verliehen werden, zum anderen kann der Ball durch Auslösen des Kickermechanismus beschleunigt werden.

Befindet sich ein Spieler in Ballbesitz nahe des Spielfeldrandes oder blockiert ein Objekt die Bahn zum Tor, so wird der Ball durch eine Drehung in die geeignete Richtung zum gegnerischen Tor oder am Objekt vorbei befördert. In Fällen, in denen ein Spieler auf das gegnerische Tor ausgerichtet ist und kein Objekt die Bahn zum Tor stört, wird der Kickermechanismus ausgelöst:

**Algorithmus 7.8** *ShootBall*
**Ablauf:**
  **if** $DistanceBallToBorder() < MIN$ **or not** $FrontClear()$ **then**
    $TurnShoot()$
  **else**
    $KickerShoot()$

Aufgrund der Roboter- und Ballposition bestimmt die Funktion *TurnShoot* die Richtung, in die gedreht werden soll, stets so, daß der Ball zum gegnerischen Tor bewegt wird.

# Kapitel 8

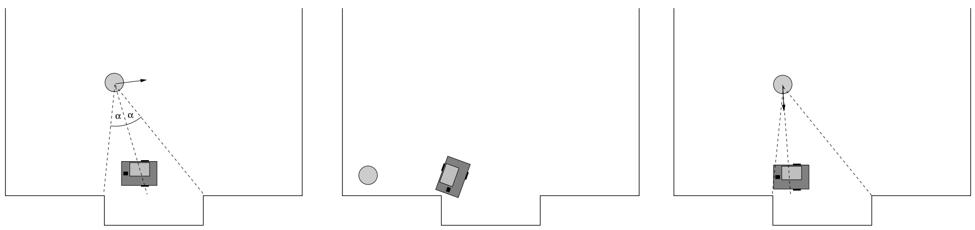
# Spielerverhalten, Taktik

Die sich durch hohe Dynamik auszeichnende Welt des Roboterfußballs verlangt von einem Spieler schnelle Reaktionen auf sich ständig ändernde Spielsituationen. Hierzu muß stets die aktuelle Spielsituation bewertet und darauf basierend entschieden werden, ob eine Aktion beibehalten oder eine neue Aktion veranlaßt werden soll.

Welche Taktik und welche Kriterien der Aktionsauswahl zugrunde liegen, wird zunächst gesondert für Torwart und Feldspieler besprochen. Während alle Feldspieler der gleichen Taktik folgen und sich die Spielerprogramme nur in wenigen Parametern unterscheiden, nimmt der Torwart mit einer speziellen Programmierung eine Sonderstellung ein. Da die Aktionsauswahl und -kontrolle für die Feldspieler jedoch komplexer strukturiert ist, wird in den zwei folgenden Abschnitten detaillierter gezeigt, auf welche Weise ein *Kontrollmodul* aus der aktuellen Spielsituation eine Aktion für einen Feldspieler ableitet und wie die hierbei zur Verfügung stehenden Aktionen strukturiert sind.

## 8.1 Torwart

Das primäre Ziel des Torhüters ist es, stets die Angriffsfläche für den Ball zu



Minimiere Angriffsfläche | Drehe zu Ball | Fange Ball ab

Abbildung 8.1: Das Torwartverhalten in Abhängigkeit der Position und Bewegung des Balls minimieren. Zu diesem Zweck bewegt sich der Torwart ausschließlich auf der Torlinie und dreht sich an den Torrändern in Richtung des Balls. Rollt der Ball jedoch mit einer Mindestgeschwindigkeit auf das Tor zu, so wird diejenige Position berechnet, an der die geschätzte Trajektorie des Balls die Torlinie schneidet und der Torwart bewegt sich zu dieser Position, um den Ball abzufangen. Abbildung 8.1 veranschaulicht das Verhalten des Torwarts.

## 8.2 Feldspieler

Als Teil des Gesamtteams versucht ein Feldspieler, gemäß einer *globalen Taktik* stets im Sinne der Mannschaft zu handeln. Die globale Taktik gibt jedoch lediglich die Rollenaufteilung unter den Spielern vor und leitet deren kooperative Verhalten. Auf welche Art und Weise ein Feldspieler seine Aufgabe im Team wahrnimmt, hängt von einer *lokalen Taktik* ab, die ausschließlich die Position des Spielers und des Balls betrachtet, um zu entscheiden, welche Aktion ein Spieler ausführen soll. Sowohl die globale, als auch die lokale Taktik werden im folgenden beschrieben.

### 8.2.1 Globale Taktik

Jeder der vier Feldspieler übt eine der Rollen *rechter Verteidiger*, *linker Verteidiger*, *rechter Stürmer* und *linker Stürmer* aus. Jede Rolle impliziert einen

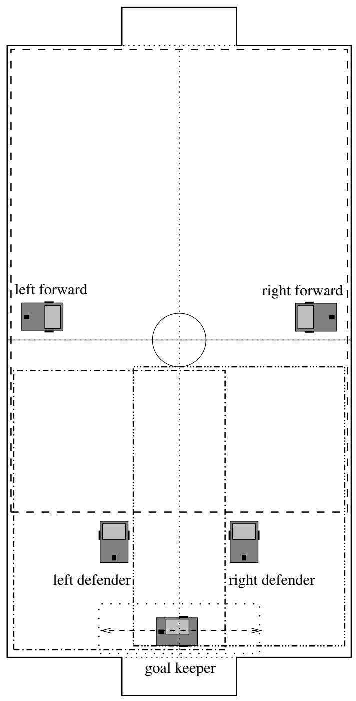



Abbildung 8.2: Die Kompetenzbereiche

*Kompetenzbereich*, innerhalb dessen der entsprechende Spieler für ein mannschaftsdienliches Spiel verantwortlich ist. Wie Abbildung 8.2 zeigt, überlappen sich die Kompetenzbereiche, um eine größere Flexibilität im Spiel zu erreichen und dennoch sicherzustellen, daß jeder Bereich des Spielfelds im Sinne eines guten Stellungsspiels von mindestens einem Spieler abgedeckt wird.

Die globale Taktik erlaubt einem Spieler nur dann seinen Kompetenzbereich zu verlassen, wenn er in Ballbesitz einen Angriff starten kann. Nach einem Angriff oder bei Ballverlust muß er unverzüglich wieder zu seinem Bereich zurückkehren.

Allein durch die Definition der Kompetenzbereiche kann deutlich Einfluß auf das Gesamtverhalten der Mannschaft ausgeübt werden: Konfigurationen für eine defensiv oder eine offensiv eingestellte Mannschaft sind leicht zu verwirklichen, ohne daß Programmcode geändert werden müßte.

Durch Kommunikation zwischen den Spielern werden die Möglichkeiten zu kooperativem Mannschaftsspiel erweitert: Befindet sich ein Spieler in guter Ballposition und ist im Begriff, den Ball zu spielen, so kann er dies den anderen Spielern mitteilen, die als Reaktion dem spielenden Roboter aus dem Weg gehen. Die Situation, daß sich zwei Roboter wegen sich überlappender Kompe-

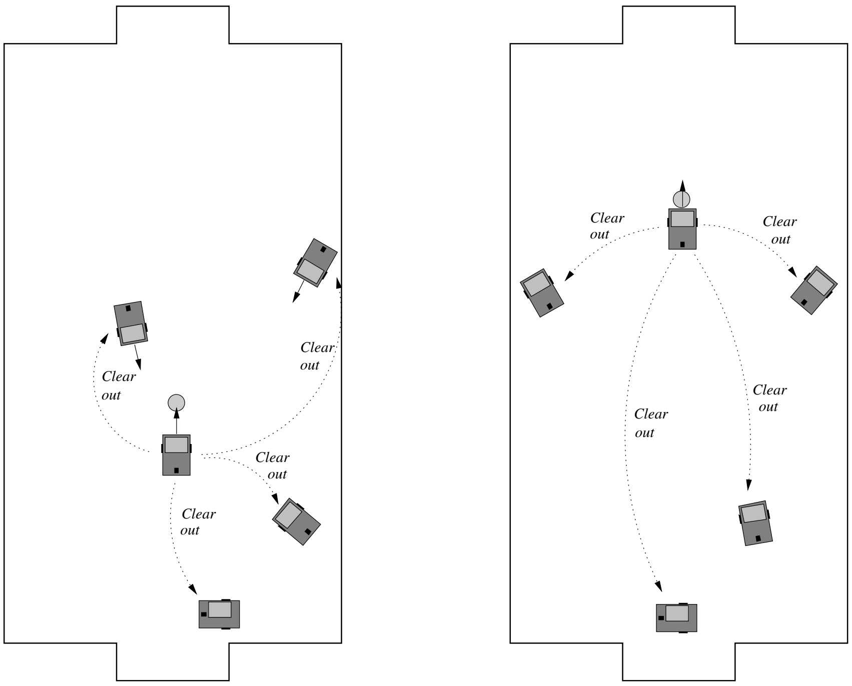



Abbildung 8.3: Der Spieler in guter Ballposition sendet *clear-out*-Nachrichten an die anderen Spieler, um nicht beim Spielen behindert zu werden.

tenzbereiche gegenseitig stören, wird so vermieden: Der Spieler mit schlechterer Position zum Ball weicht aus. Abbildung 8.3 veranschaulicht dieses Prinzip.

### 8.2.2 Lokale Taktik

Allein durch die Beschränkung des Aktionsradiuses eines Spielers auf seinen Kompetenzbereich und aufgrund der im folgenden vorgestellten lokalen Taktik erfüllt ein Spieler bereits ohne explizite Kommunikation eine mannschaftsdienliche Funktion.

Primäres Ziel für einen Feldspieler ist das Spielen des Balls in Richtung des gegnerischen Tors. Ein Feldspieler versucht stets, zu einer (wie in Abbildung 8.4 gezeigten) idealen Position hinter den Ball zu fahren, um von dort aus den Ball

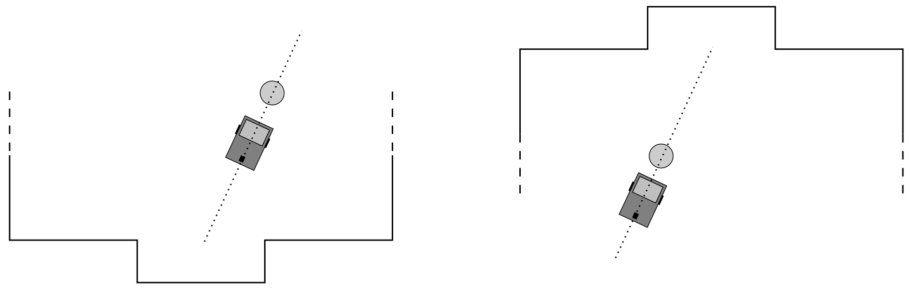


Abbildung 8.4: Die Ideale Position hinter dem Ball bei Verteidigung und Angriff.

zu spielen. Bei einem Angriff bewegt sich ein Spieler zunächst mit maximaler Beschleunigung auf das Tor zu und schießt, sobald er sich nahe genug des Tores befindet. Verteidigt ein Spieler, so blockiert er auf der idealen Position hinter dem Ball zunächst gegnerische Angriffe, um dann selbst zu versuchen, den Ball zu spielen.

Befindet sich der Ball außerhalb des Kompetenzbereichs eines Spielers, so wird der Ball von einer strategisch günstigen Position aus observiert, d.h. es wird versucht, den Ball stets im Blickfeld der Kamera zu behalten. Ein Suchverhalten kommt zum Einsatz, wenn die Position des Balls völlig unbekannt ist.

Registriert ein Spieler eine „Stall“-Situation[1], so befreit er sich aus dieser, indem er zunächst einem Pfad zu einer freien Stelle auf dem Spielfeld folgt. Da sich in der Praxis gezeigt hat, daß dies bei aggressiven gegnerischen Mannschaften dazu führt, daß sich ein Verteidiger von dem Gegner mitsamt Ball immer weiter zurückdrängen läßt, wurde das Stall-Verhalten aber insofern modifiziert, daß es nur dann zum Einsatz kommt, wenn nicht gerade der Ball gespielt werden soll.

Um bei eigenem Anspiel den Ball nicht sofort an den Gegner zu verlieren, wird zunächst die Aufstellung des Gegners bewertet und darauf basierend eine Anspielvariante ausgewählt, die den anspielenden Roboter entweder direkt durch die Mitte angreifen oder den Ball diagonal am Gegner vorbei spielen läßt.

[1]Ein Roboter „stalled“, wenn die Amperezahl der Motorstöme einen bestimmten Wert überschreitet. Dies tritt immer dann auf, wenn der Roboter auf unvorhergesehene Weise wegen eines Hindernisses stecken bleibt.

## 8.3 Das Kontrollmodul

Aufgabe des Kontrollmoduls eines Feldspielers ist die Umsetzung der taktischen Vorgaben in ausführbare Aktionen. Hierzu wird die aktuell von einem Spieler ausgeführte Aktion sowie die Weltmodellkonstellation und eventuell von dem Zentralrechner eintreffende Nachrichten betrachtet.

Wie Abbildung 8.5 zeigt, besteht das Kontrollmodul aus zwei Teilmodulen. Das *Entscheidungsmodul* bewertet die aktuelle Spielsituation und entscheidet,

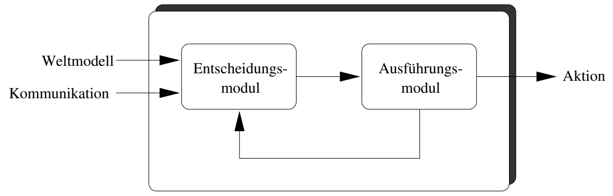



Abbildung 8.5: Das Kontrollmodul

ob die gerade ausgeführte Aktion beibehalten oder eine neue Aktion gestartet werden soll. Die Entscheidung wird dem *Ausführungsmodul* mitgeteilt, welches die Umsetzung in konkret ausführbare Befehle veranlaßt.

### 8.3.1 Das Entscheidungsmodul

Die Entscheidung, welche Aktion im nächsten Schritt ausgeführt werden soll, wird mit Hilfe des in Abbildung 8.6 gezeigten Entscheidungsbaums getroffen. Dabei wird nach rechts weisenden Pfeilen gefolgt, falls eine entsprechende Frage mit „ja“ beantwortet wurde und nach unten weisenden Pfeilen bei einem „nein“. Dies geschieht so lange, bis eine durch eine Ellipse gekennzeichnete Aktion erreicht ist.

Bevor der Entscheidungsbaum durchlaufen wird, werden zunächst die (in Abbildung 8.6 kursiv geschriebenen) relevanten Prädikate aufgrund von Informationen über die eigene Position, die Position des Balls, die gerade ausgeführte Aktion und Nachrichten des Zentralrechners aktualisiert. Allein die Bewertung dieser Prädikate führt im Anschluß dann zur Auswahl der angemessensten Aktion.

### 8.3.2 Das Ausführungsmodul

Die von einem Spieler ausführbaren Aktionen setzen sich aus einer oder mehreren Grundfertigkeiten zusammen, deren sequentielle Ausführung von dem Ausführungsmodul überwacht wird. Insgesamt stehen sieben verschiedene Aktionen mit folgender Funktionalität zur Auswahl:

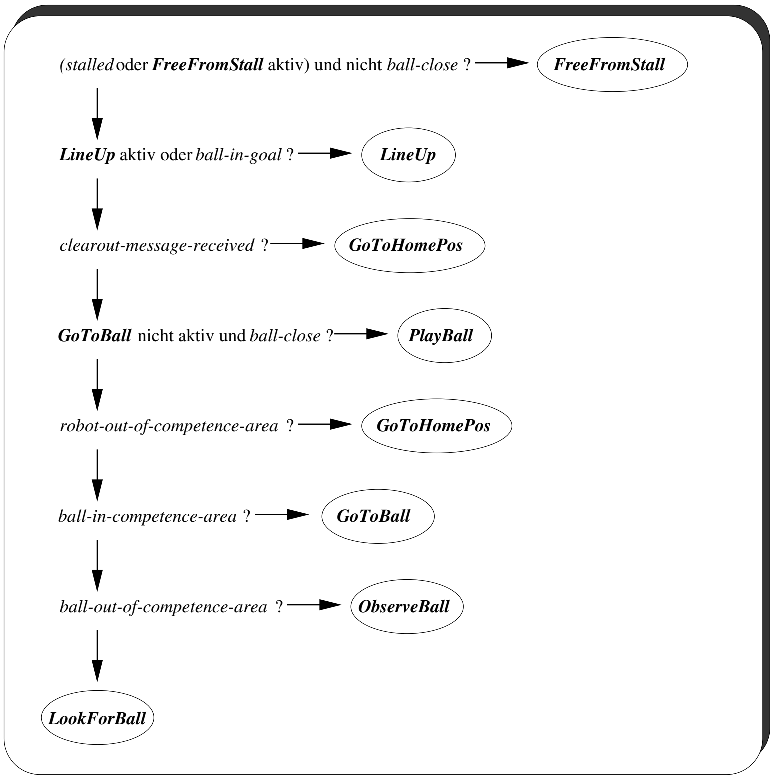



Abbildung 8.6: Das Entscheidungsmodul

**FreeFromStall**
Wähle unter sechs möglichen Pfaden den zu einer freien Position auf dem Spielfeld aus und folge diesem Pfad mittels *SetApproachVelHead*.

**LineUp**
Drehe Roboter einmal im Kreis, spiele dabei Musik und fahre schließlich mittels *GoToPosition* und *ApproachPosition* zur Spielausgangsposition.

**GoToHomePos**
Plane und folge mittels *GoToPosition* und *ApproachPosition* Pfad zur Standardposition.

**PlayBall**
Spiele den Ball mittels *PlayBall* und *ShootBall*.

**GoToBall**
Plane und folge mittels *GoToPosition* und *ApproachPosition* Pfad zur idealen Position hinter dem Ball.

**ObserveBall**
Fahre mittels *GoToPosition* zum Zentrum des eigenen Kompetenzbereichs und observiere den Ball mittels *ObserveBall*.

**LookForBall**
Suche mittels *SearchBall* und *ObserveField* nach dem Ball.

## 8.4 Ergebnisse

Obwohl die Taktik eines Feldspielers lediglich durch unterschiedliche Reaktionen auf verschiedene Klassen von Spielsituationen definiert ist, sind während eines Spiels teilweise Spielzüge zu beobachten, die in ihrer Komplexität weder vorgegeben, noch explizit geplant worden sind: Aktionen wie „den Ball von der Bande wegschießen, ihm folgen, um dann anzugreifen“ stellen *emergente Verhaltensweisen* dar, die allein durch die schnelle, jeweils angemessene Reaktion auf die aktuelle Spielsituation entstehen.

Auch ein effektives passives Stellungsspiel der Roboter ergibt sich auf natürliche Weise aus dem Spielfluß, ohne in den taktischen Vorgaben ausdrücklich berücksichtigt worden zu sein. Da ein Spieler in den meisten Fällen versucht, die ideale Position hinter dem Ball einzunehmen oder den Ball vom Zentrum des eigenen Kompetenzbereichs aus zu observieren, blockiert er stets wichtige Angriffswege für die Gegenspieler ohne selbst den Ball aktiv zu spielen.

Eine Form von „Paßspiel“ ist in Fällen zu beobachten, in denen ein Spieler den Ball zunächst observiert und erst dann aktiv wird, wenn der Ball in den eigenen Kompetenzbereich gelangt – etwa weil ein Mitspieler den Ball an einem Gegner vorbei in Richtung des gegnerischen Tors gespielt hat.

Mit dem in diesem Kapitel vorgestellten Spielverhalten wurde gezeigt, wie bereits mit einer kleinen Menge von Situations-Aktions-Beschreibungen ein effektives Spielerverhalten realisiert werden kann, wobei *emergente Verhalten* die Spielkompetenz weiter erhöhen. Durch geschickte Aufteilung der Kompetenzbereiche und Verwendung von Kommunikation zum Versenden von *clear-out*-Nachrichten kann sichergestellt werden, daß die Spieler ihre Aufgaben in der Mannschaft wahrnehmen, ohne dabei andere Mitspieler zu behindern.

# Kapitel 9

# Zusammenfassung und Ausblick

## 9.1 Zusammenfassung

In dieser Arbeit wurde das Roboterfußballteam des *CS-Freiburg* vorgestellt, wobei die vom Autor dieser Arbeit entwickelten Verfahren zur Selbstlokalisierung, Weltmodellierung, Pfadplanung und verhaltensbasierten Kontrolle ausführlich behandelt wurden.

Ausgehend von einem Überblick über die Gesamtarchitektur des Teams wurde zunächst beschrieben, wie aufgrund von Laserscannerdaten durch Scanüberdeckung die Position des Roboters auf dem Spielfeld geschätzt werden kann. Kalman-Filterung der Positionsschätzungen von Scanüberdeckung und Odometrie führen zu einem exakten Selbstlokalisierungsverfahren, das auf zuverlässige Weise die Position eines Roboters auf dem RoboCup-Spielfeld bestimmt.

Ein Spieler repräsentiert seine Sicht der Welt in Form eines Weltmodells, in dem aufgrund von Laserscannerdaten erkannte Objekte und der mit Hilfe von Kamera und Ultraschall wahrgenommene Ball modelliert werden. Die vorgestellten Verfahren erlauben die Verwaltung eines konsistenten und exakten Weltmodells, das eine zuverlässige Basis für Pfadplanung und verhaltensbasierte Handlungskontrolle darstellt.

Zur Flexibilität eines Spielers trägt die Fähigkeit bei, kollisionsfreie Pfade zu beliebigen Zielpunkten auf dem Spielfeld zu planen und zu verfolgen. Das Verfahren hierzu basiert auf einem erweiterten Sichtbarkeitsgraphen, der aufgrund der jeweils aktuellen Weltmodellbeschreibung generiert wird. Durch stetiges Neuplanen kann auf sich ändernde Umweltbedingungen reagiert und damit eine sichere und zuverlässige Navigation auf dem Spielfeld verwirklicht werden.

Es wurde gezeigt, welche grundlegenden Fertigkeiten für einen Spieler entwickelt und wie diese zu komplexen Verhalten kombiniert wurden, um auf verschiedene

Spielsituationen angemessen zu reagieren. Alle Spieler sind als prinzipiell autonome Agenten konzipiert, die allerdings aufgrund der ihnen zugewiesenen Rolle auf ein kooperatives Spiel innerhalb der Mannschaft festgelegt sind. Welche Aktion im Einzelfall auszuführen ist, entscheiden die Spieler einer lokalen Taktik folgend aufgrund ihrer lokalen Sicht der Welt. Das Spielerverhalten folgt einer gleichermaßen einfachen wie effektiven Taktik, die die Mannschaft als kompakte Einheit und jedes Mitglied als kompetenten Spieler auftreten läßt.

Mit der Implementation der vorgestellten Techniken wurde eine Steuersoftware für Roboterfußballspieler geschaffen, die als eine solide Basis für Weiterentwicklungen betrachtet werden kann und sich bereits als äußerst erfolgreich erwiesen hat: Wie die Spielergebnisse in Anhang B zeigen, ist das *CS-Freiburg-Team* bei offiziellen Wettbewerben bisher ungeschlagen und gewann sowohl die Deutsche Meisterschaft als auch die Weltmeisterschaft in der Klasse der Roboter mittlerer Größe.

## 9.2 Ausblick

Zwischen den RoboCup-Wettbewerben des letzten und diesen Jahres war in der Klasse der Roboter mittlerer Größe ein bedeutender Anstieg des Spielniveaus zu verzeichnen. So, wie bei Betrachtung der Spiele des letzten Jahres wichtige Erkenntnisse für den eigenen Ansatz gewonnen werden konnten, wurden bei den diesjährigen Spielen wichtige Erfahrungen gemacht, die einige bedeutende Verbesserungen des *CS-Freiburg-Teams* motivieren.

Der Verbesserung der grundlegenden Sensorinterpretation kommt hohe Priorität zu. Insbesondere die Ballerkennung könnte deutlich verbessert werden, wenn die Kameradaten mit einem Zeitstempel versehen würden. Im Sinne eines noch flexibleren Gesamtsystems wäre es wünschenswert, wenn die aufwendige Kalibrierung des Ballerkennungssystems vereinfacht und das System robuster gegen sich ändernde Umweltbedingungen gemacht werden könnte.

Im Kontext ausgereifterer Taktiken könnte die Pfadplanung derart weiterentwickelt werden, daß die Dynamik der Objekte und die aktuelle Spielerkonstellation berücksichtigt werden, damit zwischen Situationen, in denen ein schnelles und ungefähres Erreichen des Ziels ausreicht, und Situationen, in denen Exaktheit Priorität hat, unterschieden und entsprechend verfahren werden könnte.

Im Bereich der verhaltensbasierten Kontrolle sind aufgrund der diesjährigen Erfahrungen einige Verbesserungen denkbar, um ein erfolgreicheres und effektiveres Spiel zu verwirklichen. Zunächst müßte versucht werden, die Reaktionszeiten der Spieler deutlich zu verringern. Eine flexiblere Aktionsauswahl im Zusammenhang mit einer größeren Bandbreite von Aktionen und differenzierteren Klassen von Situationsbeschreibungen könnte zudem ein situationsgerechteres Reagieren auf eine größere Zahl von unterschiedlichen Spielkonstellationen ermöglichen.

Eine Kontrollinstanz, die explizite Handlungsplanung und Strategien mit flexibler, dynamischer Rollenzuweisung an die Spieler erlauben würde, könnte die Teamfähigkeiten der Spieler entscheidend erweitern und zu einem noch robusteren Team mit einem effektiveren Mannschaftsspiel führen.

Vielversprechend erscheint auch der Einsatz von Lernverfahren, sowohl für das Erlernen von Einzelverhalten, als auch für den Erwerb lokaler und globaler Strategien. Mit Hilfe genügend flexibler und allgemein gehaltener Situationsbeschreibungen und klar definierter Aktionsparameter stehen auf diesem Gebiet noch viele Möglichkeiten offen.

Im Sinne der Grundlagenforschung erscheint es, auch weiterhin die verwendeten Algorithmen so allgemein wie möglich zu gestalten, um sie auch in anderen Anwendungsgebieten einsetzen zu können. Insbesondere das Selbstlokalisierungsverfahren erscheint prädestiniert, auch in allgemeinen Umgebungen mit vielen Linienstrukturen erprobt zu werden. Ein solches Vorgehen unterstreicht den Sinn und Zweck von RoboCup, im Kontext der Grundlagenforschung Verfahren und Techniken zu entwickeln, die einer großen Zahl von Anwendungen in der Robotik von Nutzen sein können.

# Anhang A

# Softwarearchitektur

In diesem Kapitel wird ein Überblick über die Struktur der Softwarearchitektur gegeben. Abbildung A.1 zeigt eine hierarchische Übersicht über die Module, die auf Basis der von Kurt Konolige entwickelten *Saphira*-Bibliothek und des von Steffen Gutmann entwickelten *ScanStudios* in ANSI-C und der Saphira-Activity-Beschreibungssprache implementiert wurden.

Bem Entwurf der Softwarearchitektur wurde großer Wert auf Modularität und Wiederverwendbarkeit gelegt. Insbesondere die Module zur Kommunikation, Ballerkennung, Selbstlokalisierung, Weltmodellierung und Planung sind derart allgemein gehalten, daß eine Verwendung auch in anderen Projekten ohne große Modifikationen möglich ist. Die modulare Aufteilung findet sich auch im Verzeichnisbaum der RoboCup-Software wieder, der unter anderem folgende Unterverzeichnisse enthält:

- parameter: Parameter für Ballerkennung (gelernte Transformationstabellen) und Spielerverhalten (Spielfeldabmessung, Kompetenzbereiche, Spielerstandardpositionen)
- utility: Funktionen zur Lösung geometrischer Probleme (Tangentenberechnung etc.)
- robocup: Einlesen der Parameterdateien, Initialisierung des Systems (Steffen Gutmann)
- communication: Funktionen zum Versenden und Empfangen von Nachrichten per TCP/IP (Steffen Gutmann)
- kicker: Kickeransteuerung (Immanuel Herrmann)
- ballrecognition: Ballerkennungssoftware, Meßprogramm für die Kalibrierung derselben sowie Testprogramm (Augustinus Topor)
- selflocalization: Algorithmen zur Selbstlokalisierung mit Testprogramm

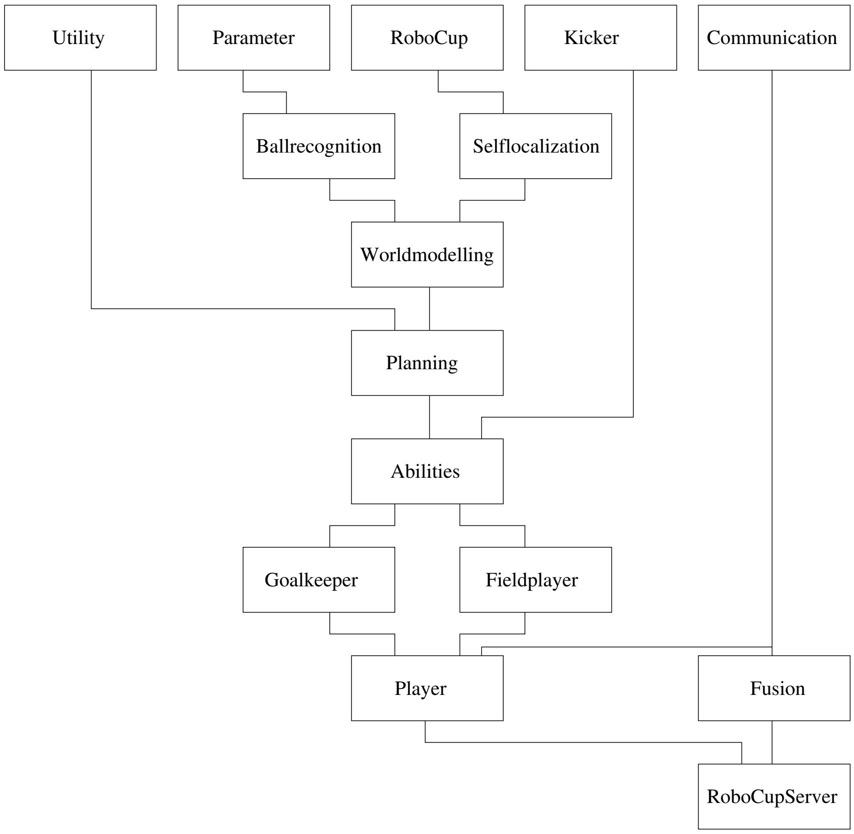



Abbildung A.1: Eine hierarchische Übersicht über die Softwaremodule

- worldmodelling: Funktionen zur Verwaltung des Weltmodells sowie Spielererkennung mit Testprogramm
- plan: Grapherzeugung und Pfadplanung sowie Testprogramm
- abilities: Spielergrundverhalten mit Testprogramm
- goalkeeper: Torwartverhalten mit Testprogramm (Steffen Gutmann)
- fieldplayer: Feldspielerverhalten mit Testprogramm
- player: RoboCup-Spielerprogramm
- fusion: Algorithmen für die globale Weltmodellierung (Wolfgang Hatzack)
- x-server: Graphische Benutzerschnittstelle (Frank Rittinger)

# Anhang B

# Ergebnisse bei offiziellen Spielen

In diesem Kapitel sind die Ergebnisse von offiziellen Spielen des *CS-Freiburg* zusammengefaßt. Das *CS-Freiburg-Team* verlor keines der Spiele und erzielte insgesamt 22 Tore bei nur 2 Gegentoren.

**Weltmeisterschaft „RoboCup“, Paris, 2.7-9.7.98:**

| | | |
|---|---|---|
| CSF - NAIST | = | 1 : 1 (nach Elfmeterschießen) |
| CSF - USC | = | 1 : 0 |
| CSF - Real Magicol | = | 3 : 0 |
| | | |
| CSF - Yale | = | 2 : 0 |
| CSF - Osaka | = | 3 : 0 |
| CSF - Tübingen | = | 2 : 0 |

**CS-Freiburg ist Weltmeister.**

**Deutsche Meisterschaft „VisionCup“, Stuttgart, 30.9-1.10.98:**

| | | |
|---|---|---|
| CSF - Ulm | = | 4 : 0 |
| CSF - Tübingen | = | 4 : 1 (nach Elfmeterschießen) |
| CSF - München | = | 2 : 0 |

**CS-Freiburg ist Deutscher Meister.**

# Abbildungsverzeichnis

# Literaturverzeichnis

[Andou, 1997] T. Andou. *Refinement of Soccer Agent's Positions Using Reinforcement Learning.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 373–388. Springer-Verlag, 1997.

[Andre und Teller, 1998] D. Andre und A. Teller. *Evolving Team Darwin United.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 317–329. Springer-Verlag, 1998.

[Aparício *et al.*, 1998] A. Aparício, R. Ventura, P. Lima, und C. Pinto-Ferreira. *ISocRob – Team Description.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 423–429. Springer-Verlag, 1998.

[Asada *et al.*, 1997] M. Asada, P. Stone, H. Kitano, A. Drogoul, D. Duhaut, M. Veloso, H. Asama, und S. Suzuki. *The RoboCup Agent Challenge: Goals and Protocols for Phase I.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 42–61. Springer-Verlag, 1997.

[Brooks, 1991] R.A. Brooks. *Intelligence without reason.* In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Seiten 569–595, 1991.

[Bui *et al.*, 1998] A.V. Bui, J. Gregersen, A. Jennings, J. Keen, L. Padgham, P. Parisi, und M. Wirth. *RMIT Raiders.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 483–489. Springer-Verlag, 1998.

[Burgard *et al.*, 1996] W. Burgard, D. Fox, D. Henning, und T. Schmidt. *Estimating the absolute position of a mobile robot using probability grids.* In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-96)*, Seiten 896–901, 1996.

[Burkhard *et al.*, 1997] H.D. Burkhard, M. Hannebauer, und J. Wendler. *AT Humbold – Development, Practice and Theory.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 357–372. Springer-Verlag, 1997.

[Cantor *et al.*, 1998] C. Cantor, A. Angela, J. Davis, C. Foster, N. Inala, L. Law, T. Mallberg, A. Miura, B. Ng, und J. Teevan. *RoboCup at Yale.* http://joyce.eng.yale.edu/Robocup/RoboTeam.html, 1998.

[Castellanos *et al.*, 1996] J.A. Castellanos, J.D. Tardós, und J. Neira. *Constrained-Based Mobile Robot Localization.* In *International Workshop on Advanced Robotics and Intelligent Machines (ARIM-96)*, 1996.

[Cox, 1991] I.J. Cox. *Blanche – An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

[Fox *et al.*, 1997] D. Fox, W. Burgard, und S. Thrun. *The Dynamic Window Approach to Collision Avoidance. IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[Guibas *et al.*, 1997] L.J. Guibas, R. Motwani, und P. Raghavan. *The Robot Localization Problem. SIAM Journal on Computing*, 26(4):1120–1138, 1997.

[Gutmann *et al.*, 1998a] J-S. Gutmann, W. Burgard, D. Fox, und K. Konolige. *An Experimental Comparison of Localization Methods.* In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS-98)*, Seiten 736–743, 1998.

[Gutmann *et al.*, 1998b] S. Gutmann, W. Hatzack, I. Herrmann, B. Nebel, F. Rittinger, A. Topor, T. Weigel, und B. Welsch. *The CS Freiburg Team.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 451–457. Springer-Verlag, 1998.

[Gutmann und Nebel, 1997] J-S. Gutmann und B. Nebel. *Navigation mobiler Roboter mit Laserscans.* In *Autonome Mobile Systeme*, Seiten 36–47. Springer-Verlag, 1997.

[Gutmann und Schlegel, 1996] J.S. Gutmann und C. Schlegel. *AMOS: Comparison of Scan Matching Approaches for Self-Localization in Indoor Environments.* In *IEEE Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots (Eurobot)*, Seiten 61–67, 1996.

[Gutmann, 1996] J-S. Gutmann. *Vergleich von Algorithmen zur Selbstlokalisierung eines mobilen Roboters.* Diplomarbeit, Universität Ulm, 1996.

[Hart *et al.*, 1968] P.E. Hart, N.J. Nilsson, und B. Raphael. *A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Herrman, 1998] I. Herrman. *Roboter-Schußvorrichtung.* Technischer Bericht, Universität Freiburg, 1998.

[Karch *et al.*, 1997] O. Karch, H. Noltemeier, M. Schwark, und T. Wahl. *Relokalisation – Ein theoretischer Ansatz in der Praxis.* In *Autonome Mobile Systeme*, Seiten 119–130. Springer-Verlag, 1997.

[Kitano *et al.*, 1997] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, und E. Osawa. *RoboCup: A Challenge Problem for AI and Robotics.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 1–19. Springer-Verlag, 1997.

[Klupsch *et al.*, 1998] M. Klupsch, T. Bandlow, M. Grimme, I. Kellerer, M. Lückenhaus, F. Schwarzer, und C. Zierl. *Agilo RoboCuppers: RoboCup Team Description*. In *RoboCup-98: Robot Soccer World Cup II*, Seiten 431–437. Springer-Verlag, 1998.

[Konolige *et al.*, 1997] K. Konolige, K. Myers, E. Ruspini, und A. Saffiotti. *The Saphira Architecture: A Design for Autonomy. Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235, 1997.

[Kraetzschmar *et al.*, 1998] G.K. Kraetzschmar, S. Enderle, S. Sablatnög, T. Boß, M. Ritter, H. Braxmayer, H. Folkerts, G. Mayer, M. Müller, H. Seidl, M. Klinger, M. Dettinger, R. Wörtz, und G. Palm. *The Ulm Sparrows: Research into Sensorimotor Integration, Agency, Learning, and Multiagent Cooperation*. In *RoboCup-98: Robot Soccer World Cup II*, Seiten 459–465. Springer-Verlag, 1998.

[Kuth *et al.*, 1998] A. Kuth, A. Bredenfeld, H. Guenther, H.U. Kobialka, B. Klaassen, U. Licht, K.L. Paap, P.G. Ploeger, H. Streich, J. Vollmer, J. Wilberg, R. Worst, und T. Christaller. *Team Description of the GMD RoboCup-Team*. In *RoboCup-98: Robot Soccer World Cup II*, Seiten 439–449. Springer-Verlag, 1998.

[Latombe, 1991] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[Lu und Milios, 1994] F. Lu und E.E. Milios. *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans*. In *IEEE Computer Vision and Pattern Recognition Conference (CVPR)*, Seiten 935–938, 1994.

[Luke *et al.*, 1997] S. Luke, C. Hohn, J. Farris, G. Jackson, und J. Hendler. *Co-evolving Soccer Softbot Team Coordination with Genetic Programming*. In *RoboCup-97: Robot Soccer World Cup I*, Seiten 398–411. Springer-Verlag, 1997.

[Maybeck, 1990] P.S. Maybeck. *The Kalman Filter: An Introduction to Concepts*. In I.J. Cox und G.T. Wilfong, Herausgeber, *Autonomous Robot Vehicles*, Seiten 194–204. Springer Verlag, 1990.

[Nakamura *et al.*, 1998] T. Nakamura, K. Terda, A. Shibita, J. Morimoto, H. Adachi, und H. Takeda. *The RoboCup-NAIST: A Cheap Multisensor-Based Mobile Robot with On-Line Visual Learning Capability*. In *RoboCup-98: Robot Soccer World Cup II*, Seiten 291–303. Springer-Verlag, 1998.

[Nakamura, 1997] T. Nakamura. *Development of Self-Learning Vision-Based Mobile Robots for Aquiring Soccer Robots Behaviors*. In *RoboCup-97: Robot Soccer World Cup I*, Seiten 257–275. Springer-Verlag, 1997.

[Nardi *et al.*, 1998] D. Nardi, G. Clemente, und E. Pagello. *ART Azzurra Robot Team*. In *RoboCup-98: Robot Soccer World Cup II*, Seiten 467–473. Springer-Verlag, 1998.

[Oller *et al.*, 1997] A. Oller, J.A. Ramon, A. Figueras, S. Esteva, Ll. de la Rosa, J. Humet, und E. Del Acebo. *Description of Rogi-Team.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 286–294. Springer-Verlag, 1997.

[Plagge *et al.*, 1998] M. Plagge, B. Diebold, R. Günther, J. Ihlenburg, D. Jung, K. Zahedi, und A. Zell. *Systemkonzept der Tübinger mobilen Roboter zum RoboCup 98.* Vortrag an der KI '98, Bremen, 1998.

[Price *et al.*, 1997] A. Price, A. Jennings, und J. Keen. *RoboCup97: An Omnidirectional Perspective.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 320–332. Springer-Verlag, 1997.

[Rittinger, 1998] F. Rittinger. *Der CS Freiburg RoboCup '98 Server.* Studienarbeit, Universität Freiburg, 1998.

[Shen *et al.*, 1997] W. Shen, J. Adibi, R. Adobbati, B. Cho, A. Erdem, H. Moradi, B. Salemi, und S. Tejada. *Autonomous Soccer Robots.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 295–304. Springer-Verlag, 1997.

[Shen *et al.*, 1998] W. Shen, J. Adibi, R. Adobbati, S. Lanksham, H. Moradi, B. Salemi, und S. Tejada. *Integrated Reactive Soccer Agents.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 251–263. Springer-Verlag, 1998.

[Sieber und Huber, 1980] H. Sieber und L. Huber. *Mathematische Begriffe und Formeln.* Ernst Klett Verlag, 1980.

[Smith *et al.*, 1990] R. Smith, M. Self, und P. Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics.* In I.J. Cox und G.T. Wilfong, Herausgeber, *Autonomous Robot Vehicles*, Seiten 167–193. Springer Verlag, 1990.

[Stone und Veloso, 1997] P. Stone und M. Veloso. *The CMUnited-97 Simulator Team.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 389–397. Springer-Verlag, 1997.

[Suzuki *et al.*, 1997] S. Suzuki, Y. Takahashi, E. Uchibe, M. Nakamura, C. Mishima, H. Ishizuka, T. Kato, und M. Asada. *Vision-Based Learning Towards RoboCup: Osaka University "Trackies".* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 305–319. Springer-Verlag, 1997.

[Suzuki *et al.*, 1998] S. Suzuki, T. Kato, H. Ishizuka, Y. Takahashi, E. Uchibe, und M. Asada. *An Application of Vision-Based Learning for a Real Robot in RoboCup – A Goal Keeping Behavior for a Robot with an Omnidirectional Vision and an Embedded Servoing.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 281–290. Springer-Verlag, 1998.

[Thrun und Bücken, 1996] S. Thrun und A. Bücken. *Integrating Grid-Based and Topological Maps for Mobile Robot Navigation.* In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Seiten 944–950, 1996.

[Topor, 1998] A. Topor. *Ballerkennung und Positionsschätzung des CS-Freiburg Teams bei RoboCup '98.* Studienarbeit, Universität Freiburg, 1998.

[Veloso *et al.*, 1997] M. Veloso, P. Stone, K. Han, und S. Achim. *The CMUnited-97 Small Robot Team.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 242–256. Springer-Verlag, 1997.

[Weiss *et al.*, 1994] G. Weiss, C. Wetzler, und E.V. Puttkamer. *Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans.* In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS-94)*, Seiten 595–601, 1994.

[Werger *et al.*, 1997] B. Werger, P. Funes, M. Schneider-Fontán, C. Witty, und T. Witty. *The Spirit of Bolivia: Complex Behavior Through Minimal Control.* In *RoboCup-97: Robot Soccer World Cup I*, Seiten 348–356. Springer-Verlag, 1997.

[Yokota *et al.*, 1998] K. Yokota, K. Ozaki, N. Watanabe, A. Matsumoto, D. Koyama, T. Ishikawa, K. Kawabata, H. Kaetsu, und H. Asama. *Cooperative Team Play Based on communication.* In *RoboCup-98: Robot Soccer World Cup II*, Seiten 491–496. Springer-Verlag, 1998.

# Erklärung

Name: Thilo Weigel Matr.Nr.: 8410639

Ich erkläre hiermit, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Freiburg, den 29. Dezember 1998