



Diplomarbeit der Informatik

# Kooperative Planung für autonome humanoide Fußballroboter

Gretta Hohl

Prüfer: Prof. Dr. Raúl Rojas  
Betreuer: Hamid Reza Moballegh

05. Oktober 2009



Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebene Hilfsmittel verwendet habe.  
Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Berlin, den 05. Oktober 2009

---

(Gretta Hohl)

Im Rahmen dieser Diplomarbeit wird ein Softwaresystem vorgestellt, welches einem Team von menschenähnlichen Robotern, sogenannten Humanoiden, ermöglicht autonom Fußball zu spielen. Im Gegensatz zum menschlichen Fußballspieler, der von Natur aus mit unterschiedlichen Spielsituationen umgehen kann, benötigt der Roboter alleine für die Wahrnehmung von Veränderungen eine Kamera oder ähnliche Sensoren, sowie eine spezielle Software. Um auch noch auf diese Veränderungen reagieren zu können, muss er ein Planungssystem besitzen, das anhand von Sensorinformationen, über die nächste Handlung der Maschine entscheidet und die Aktoren, welche für die Ausführung zuständig sind, steuert.

Für die Entwicklung der Planung sind die oben genannten Sensorinformationen notwendig, da sie Auskunft über die Lage der Umgebung, sowie den Zustand des Roboters geben. Abhängig vom Robotersystem können aus diesen Informationen globale Positionsdaten gewonnen werden, die für die Planung von großem Vorteil sind.

Im Laufe des letzten Jahres sind im Projekt FHumanoids, an der Freien Universität Berlin, zwei unterschiedliche Roboterplattformen entstanden. Als erstes ist für die ältere Generation ein von Grund auf neues Planungssystem entwickelt worden, welches später auf die nachfolgende Generation portiert und weiterentwickelt werden konnte. Im Vergleich zur älteren Generation verfügt die Neue über deutlich mehr Ressourcen, im Bereich der Rechenleistung, Sensorik und Motorik. Dies ermöglicht den Robotern sich selbst auf dem Spielfeld zu lokalisieren.

Im ersten Kapitel wird auf die Eigenschaften beider Plattformen, die Herausforderung der Aufgabe und die Motivation näher eingegangen. Das zweite Kapitel stellt unterschiedliche Ansätze für die Entwicklung von Planungssystemen vor. Weiterführend werden konkrete Architekturen vorgestellt, bevor im dritten Kapitel eine genaue Beschreibung der eingesetzten Architektur erfolgt. Im vierten Kapitel wird das Planungssystem der FHumanoids dargestellt und abschließend das Ergebnis der Entwicklung diskutiert.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>7</b>
1.1	Ein intelligenter Roboter . . . . .	7
1.2	Motivation . . . . .	7
1.2.1	Robocup: Die Idee . . . . .	8
1.2.2	Die Liga Humanoid KidSize . . . . .	8
1.3	Das Forschungsprojekt FHumanoids . . . . .	11
1.3.1	Die Roboterplattform 2008 . . . . .	11
1.3.2	Die Roboterplattform 2009 . . . . .	15
<b>2</b>	<b>Verhaltensarchitekturen für autonome Roboter</b>	<b>18</b>
2.1	Der biologische Hintergrund . . . . .	18
2.2	Konzepte der Planung . . . . .	18
2.2.1	Reaktiv . . . . .	18
2.2.2	Deliberativ . . . . .	20
2.2.3	Hybrid . . . . .	21
2.3	Relevante Planungsarchitekturen . . . . .	22
2.3.1	Subsumption Architektur . . . . .	22
2.3.2	Hierarchical dynamical systems to control reactive behavior . . . . .	23
2.3.3	Autonomus Robot Architecture (AuRA) . . . . .	25
2.3.4	Hybrid Mobile Robot Architecture with Integrated Planning and Control. . . . .	27
2.4	Diskussion . . . . .	28
<b>3</b>	<b>Die Architektur des Planungssystems</b>	<b>30</b>
3.1	Concurrent Scenario-Based Planning (CSBP) . . . . .	30
3.1.1	Das Szenario . . . . .	31
3.1.2	Transparente Szenarien . . . . .	32
3.1.3	Probleme mit Szenarien . . . . .	33
3.1.4	Die Nebenläufigkeit von Szenarien . . . . .	35
3.2	Die Ebenen der FHumanoids Planung . . . . .	36
3.2.1	Die Bewegung . . . . .	36
3.2.2	Das Verhalten . . . . .	38
3.2.3	Die Rolle . . . . .	38
3.2.4	Die Strategie . . . . .	39
3.3	Kommunikation und Steuerung . . . . .	39
3.3.1	Der Kontrollpfad . . . . .	40

3.3.2	Der Statuspfad . . . . .	41
3.3.3	Der Identifikationspfad . . . . .	41
3.4	Die kooperativen Aspekte . . . . .	42
<b>4</b>	<b>Die Planung der Fumanoids</b>	<b>45</b>
4.1	Der Stürmer . . . . .	46
4.1.1	Das Finden des Balles . . . . .	48
4.1.2	Die Suche nach dem Ball . . . . .	50
4.1.3	Das Aufstehen . . . . .	53
4.1.4	Sich dem Ball annähern . . . . .	55
4.1.5	Das Dribbeln . . . . .	59
4.1.6	Das Schießen . . . . .	61
4.2	Der Flügelspieler . . . . .	64
4.2.1	Das seitliche Unterstützen des Stürmers . . . . .	65
4.3	Der Unterstützer . . . . .	67
4.3.1	Das frontale Unterstützen des Stürmers . . . . .	70
4.4	Diverse Spielstrategien . . . . .	71
4.4.1	Ein Stürmer und zwei Flügelspieler . . . . .	72
4.4.2	Ein Stürmer, ein Flügelspieler und ein Unterstützer . . . . .	74
4.4.3	Die Strategie mit Torwart . . . . .	75
4.5	Die globale Planung . . . . .	76
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>77</b>

# 1 Einleitung

## 1.1 Ein intelligenter Roboter

Ein Roboter ist eine automatisch handelnde Maschine die menschliche Arbeit ersetzt [25]. Die Komplexität der verrichteten Arbeit kann dabei als Maß für die Intelligenz eines Roboters dienen. Es sind Maschinen konstruiert worden die uns befähigen weltweit zu kommunizieren, an unsere Vorhaben erinnern, oder sich von alleine fortbewegen können und noch einiges mehr. Vermutlich gibt es keinen Roboter, der alleine alle diese Funktionalität mit sich bringt und dabei voll automatisch agiert. Ein solcher Roboter könnte idealerweise für jede beliebige Arbeit eingesetzt werden und wäre einem menschlichen Arbeiter ebenbürtig.

Dieser Grad der Entwicklung stellt den Höhepunkt eines intelligenten Roboters dar. Heute sind die Grenzen der Roboter allerdings noch klar zu erkennen und die Vielseitigkeit eines Menschen ist durch einen Roboter bisher unerreicht. Die Entwicklung steckt in den Kinderschuhen, aber sie schreitet schnell voran. Selbst die Entwicklung von menschenähnlichen Robotern, sogenannte Humanoide oder Androide, erleben derzeit einen rasanten Fortschritt. Bevor es in der Zukunft zur Realisierung eines Roboters kommt, der in der Lage ist einen menschlichen Arbeiter in allen Disziplinen zu ersetzen, wird die Menschheit noch viele technische, aber vor allem auch ethische und soziale Fragen klären müssen.

## 1.2 Motivation

Das Fußballspielen erfordert vom Menschen volle Konzentration und Körpereinsatz. Gleichzeitig müssen mehrere Fähigkeiten koordiniert werden wie visuelle und akustische Erkennung, Kommunikation, Verstand und Körperbewegungen. Der Mensch benötigt Geschicklichkeit den Ball in Richtung Tor zu dribbeln und im richtigen Moment zu schießen. Im Gegensatz zum Menschen bringt ein Roboter ganz andere Voraussetzungen zum Fußballspielen mit. Er besitzt kein Wissen darüber was ein Ball ist, wie ein Tor auszusehen hat oder welchen Zweck sie erfüllen. Auch Spielregeln sind für ihn vollkommen unbekannt. Ein Roboter der Fußball spielt, wird mit vielen Problemen konfrontiert [15]. Eines davon ist, zwischen Gegnern, Mitspielern und neutralen Personen, wie dem Schiedsrichter zu unterscheiden. Außerdem sind z.B. die Bewegungen der Spieler und des Balles auf dem Feld unvorhersehbar. Die Aufgabe wird dadurch komplexer, dass Ereignisse parallel auftreten und der Bedarf an Kommunikation und an Teamwork zwischen den Robotern besteht.

### 1.2.1 Robocup: Die Idee

Robocup [30] ist das internationale Event zur Förderung der Forschung und Lehre im Bereich Robotik, Künstliche Intelligenz und aller damit verbundenen Felder. Es handelt sich hierbei, um einen Wettbewerb, bei dem Menschen ihr fachliches Wissen zum Thema Robotik miteinander messen können. Die Turnierteilnehmer sind jedoch nicht die Menschen selbst, sondern die von Ihnen gebauten Roboter oder entwickelten Programme im Bereich der Künstlichen Intelligenz.

Das Turnier ist nach dem Wissensstand der Teilnehmer in mehrere Disziplinen strukturiert.

- Den Anfang macht die Liga **RoboCupSoccer**, wo die Roboter zum Fußballspielen gegeneinander antreten. Die Spielregeln sind eine modifizierte Version der FIFA Regeln und werden dem aktuellen Stand der Entwicklung angepasst.
- In der Liga **RoboCupRescue** treten Roboter gegeneinander an, die sich auf unebenem Gelände fortbewegen können, um mit Hilfe diverser Sensorik nach Opfern zu suchen. Sie sollen in Katastrophengebieten zum Einsatz kommen.
- In der Liga **RoboCup@Home** messen sich Roboter miteinander, die dem Menschen im Haushaltsbereich zur Seite stehen sollen.
- Eine ganz besondere Liga stellt **Robocup Junior** dar, wo Kinder dazu motiviert werden sich mit Technik und Robotik vertraut zu machen.

Zur Gewährleistung des wissenschaftlichen Fortschritts im Turnier ist das Robocup Komitee gegründet worden. Es entwirft die Regeln jeder einzelnen Liga und passt die Ansprüche des Turniers dem aktuellen Entwicklungsstand an. Auch das Aussehen und die Ausrüstung der Roboter werden genauestens festgelegt. So wird eine Chancengleichheit der teilnehmenden Teams garantiert.

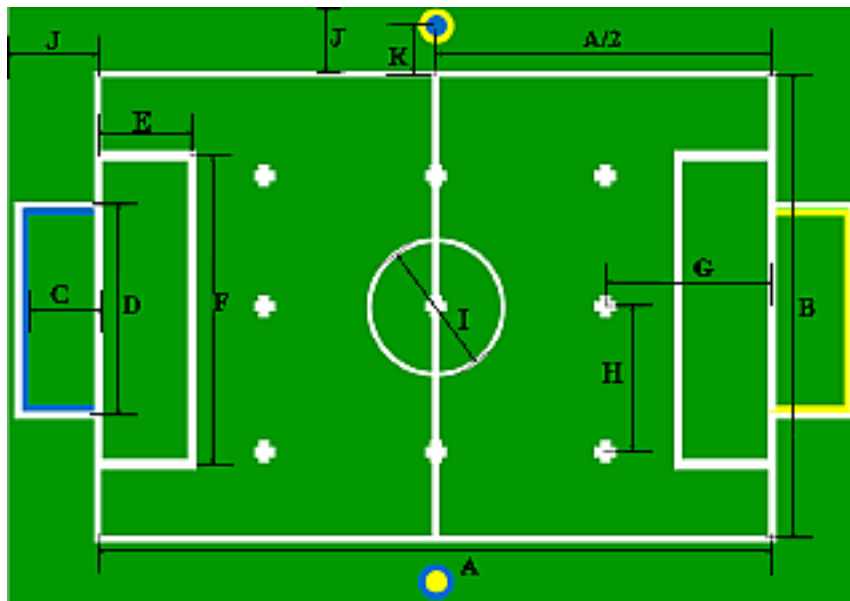
Das offizielle Ziel von Robocup ist es im Jahr 2050 eine Mannschaft von autonomen humanoiden Fußballrobotern gegen die menschlichen Fußballweltmeister im Spiel antreten zu lassen und diese zu besiegen.

### 1.2.2 Die Liga Humanoid KidSize

In der Liga Humanoid von RoboCupSoccer wird zwischen den 30-60 cm großen KidSize und den 100-160 cm großen TeenSize Robotern unterschieden. Die FHumanoids gehören mit einer Größe von knapp unter 60 cm in die erste der beiden Kategorien.

Teams, die in dieser Liga mitspielen, treten mit autonomen Robotern an. Diese besitzen Sensoren, die synchron zu dem menschlichen Wahrnehmungsvermögen sind. Der Körperbau muss bestimmten Proportionen entsprechen, weiterhin dürfen die Kameras nur im Kopf des Roboters platziert werden und der Schwerpunkt der Konstruktion muss bestimmte Voraussetzungen erfüllen. Beispielsweise darf ein Humanoide nicht am Wettkampf teilnehmen, wenn er mit Ultraschall- oder Infrarotsensoren ausgestattet ist. Diese sind nur einige der zu erfüllenden Kriterien, die das Robocup Komitee für die Liga

Humanoid KidSize festgelegt hat. Die genauen Voraussetzungen für die Teilnahme am Wettbewerb sowie auch die Spielregeln können in [31] nachgelesen werden.



Markierung	Bedeutung	Wert (cm)
A	Feld Länge	600
B	Feld Breite	400
C	Tor Tiefe	50
D	Tor Breite	150
E	Strafraum Länge	60
F	Strafraum Breite	300
G	Entfernung zur Strafschussposition	180
H	Breite der Neustartposition	100
I	Durchmesser des Mittelkreises	120
J	Breite des Seitenrahmens (min.)	70
K	Entfernung zwischen Säule und Feld	40

Abbildung 1.1: Das Humanoid KidSize Spielfeld [31] aus der Vogelperspektive.

Ein Spiel dieser Liga findet auf einem 6x4 m großen Fußballfeld statt, siehe Abbildung 1.1, wobei auf den Rasen verzichtet und stattdessen ein grüner Teppich verwendet wird. Das Feld besitzt neben, den aus der FIFA bekannten Feldlinien zusätzliche Markierungen. Diese sind: die Färbung der Tore mit Blau bzw. Gelb, die Platzierung zweier farbiger Säulen an beiden Seiten der Mittellinie und die Markierungen der Strafschusslinie durch weiße Kreuze. Diese Markierungen dienen den Robotern als Orientierungshilfe. Ein Ziel dieser Liga ist es, in Zukunft auf einem original Fußballfeld zu spielen, weshalb die Orientierungspunkte, nach und nach abgeschafft werden. Zum Beispiel sind für die Spiele

beim Robocup 2009, die zwei seitlichen weißen Kreuzmarkierungen der Strafflinie entfernt worden. Im kommenden Jahr ist es geplant die Säulen an der Mittellinie auf die Hälfte ihrer Größe zu reduzieren und die Tore mit einem farbigen Rahmen und einem weißen Netz zu ersetzen.

Die Spielzeit besteht aus zwei Halbzeiten á 10 min und die Halbzeitpause beträgt 5 min. Während des Spiels dürfen maximal drei Roboter einer Mannschaft gegeneinander antreten. Auf diese Roboter darf während der Spielzeit keinerlei Einfluss ausgeübt werden. Sie müssen den Ball selbstständig finden, die eigenen Mitspieler identifizieren, den Gegnern ausweichen und Tore schießen. Wie im richtigen Fußball entscheiden die erzielten Tore über Sieg oder Niederlage.

### **Wissenschaftliche Herausforderungen der Liga**

Eine der großen Herausforderungen bei der Entwicklung von humanoiden Fußballrobotern ist es, ihnen das *Laufen* auf zwei Beinen zu ermöglichen. Die Größe des Roboters, sein Gewicht und die Position seines Schwerpunktes spielen hierbei eine wichtige Rolle. Je größer der Roboter ist, umso schwieriger ist es die Laufbewegung zu stabilisieren. Die Realisierung eines guten Laufalgorithmuses ist die Voraussetzung einer erfolgreichen Teilnahme in dieser Liga.

Die zweite grundlegende Fähigkeit des Roboters stellt die *Erkennung der Objekte* auf dem Spielfeld dar. Ein Gegenstand kann aufgrund seiner Farbe identifiziert werden, z.B. Orange für den Ball und Gelb bzw. Blau für die Tore. Die Objekterkennung wird durch unscharfe Bildaufnahmen erschwert, die durch die Bewegung der Roboter verursacht werden.

Besitzt ein Roboter die Fähigkeiten zum Laufen und zur Objekterkennung, so kann mit der Entwicklung eines *Planungssystems* begonnen werden. Das Planungssystem befindet sich auf der obersten Stufe der Softwarehierarchie und ist somit in seiner Funktionalität von allen anderen Systemkomponenten abhängig. Fehler, die auf dieser Ebene auftreten, können auch aus einem der vielen Systemkomponenten stammen. Dies erschwert die Entwicklungsarbeit. Weiterhin ist es auch eine Herausforderung alle anderen Systemkomponenten so miteinander zu verbinden, dass die Roboter bei der Erfüllung ihrer Aufgabe schnell, stabil und robust bleiben. Bei einer Anzahl von sechs Spielern auf einem kleinen Spielfeld spielt Teamarbeit eine essenzielle Rolle.

Einige Roboter, die in dieser Liga antreten, verfügen neben den aufgezählten Grundfunktionen auch über die Fähigkeit zur *Selbstlokalisierung*. Dies gilt auch für die FHumanoids der Robotergeneration 2009, siehe Abschnitt 1.3.2. Dieses System bringt besonders im Bereich der Planung viele Vorteile mit sich. Beispielsweise kann ein Roboter durch die globalen Koordinaten des Balles sowie die seiner Mitspieler entscheiden wer die beste Angriffsposition besitzt. Die Schwierigkeit bei der Entwicklung der Selbstlokalisierung besteht darin, dass die Umgebung sich ständig verändert. Zum Beispiel können andere Roboter die Orientierungspunkte verdecken oder der Roboter hat sich selber, durch die Verfolgung des Balls, in eine ungünstige Position bewegt.

## 1.3 Das Forschungsprojekt FManoids

An der Freien Universität Berlin, in der Arbeitsgruppe Künstliche Intelligenz, hat die Entwicklung autonomer Roboter eine langjährige Tradition.

Im Herbst des Jahres 2006 ist das so genannte FManoids Projekt ins Leben gerufen worden. Das Ziel ist es menschenähnliche autonome Roboter zu entwickeln, welche das Fußballspielen beherrschen. Das Projekt ist mit einer kleinen Besetzung von fünf Personen und zwei Roboter gestartet worden. Zum aktuellen Zeitpunkt zählt das Projekt die bereits vierte Generation von entwickelten Robotern und beschäftigt ein großes Team von fünfzehn Personen.

Für diese Arbeit sind zwei Robotergenerationen aus dem FManoids Projekt von Bedeutung: die Generation aus dem Jahr 2008 und 2009. Alle Roboter einer Generation sind baugleich konstruiert und besitzen dieselbe Software.

In den folgenden Abschnitten wird die Hardware und einige wichtigen Aspekte der Software dieser Roboterplattformen beschrieben.

### 1.3.1 Die Roboterplattform 2008

#### Hardware

Die Grundlage der mechanischen Struktur eines Roboters bildet das Bioloid Kit und die Dynamixel Motoren der koreanischen Firma Robotis inc. [32]. Insgesamt sind in einem Roboterkörper achtzehn Motoren eingebaut. Sechs sind im Oberkörper und die restlichen zwölf im Unterkörper untergebracht. Der Vorteil dieser Konstruktion ist ein höherer Freiheitsgrad im Bein- und Fußbereich, was sich positiv auf die Laufbewegung auswirkt.

Durch das Laufen werden die, in den Beinen platzierten Motoren, stark beansprucht, da sie das gesamte Gewicht der verbauten Körperteile tragen. Um dem Gewicht entgegen zu wirken, sind im Unterkörper die stärkeren Dynamixel RX-28 Motoren eingebaut worden. Im Gegensatz zum Bein- und Fußbereich sind im Oberkörper die leistungsschwächeren Dynamixel AX-12 Motoren verwendet worden. Die Eigenschaften der beiden Motoren können in der Abbildung 1.2 nachgelesen werden.

Ungefähr 20% des mechanischen Skeletts sind aus Produkten des Hauses Robotis inc.

Motorart	Gewicht (g)	Getriebeübersetzung	Max Drehkraft (Nm)	Geschwindigkeit (s/60°)	Auflösungsgrad
Dynamixel AX-12	55	1:254	16.5 @10V	0.196	0.35
Dynamixel RX-28	72	1:193	28.3 @12V	0.167	0.35

Abbildung 1.2: Die Eigenschaften der eingesetzten Dynamixel AX-12 und RX-28 Motoren [18].

zusammengesetzt worden. Die Restlichen 80% sind vom Team entworfen und angefertigt worden. Dabei sind schwarze Kunststoffplatten unterschiedlicher Breite benutzt worden. In der Abbildung 1.3 ist der konstruierte Roboter dargestellt.

Die Rolle des Oberkörpers ist es instabile Zustände auszubalancieren. Er beherbergt die Sensorik und die Recheneinheit des Roboters. Eine Kamera und mehrere Servomotoren dienen als einzige Sensoren, so dass nur begrenzt Zustandsinformationen verfügbar sind. Jeder Servomotor ist durch den Motorbus direkt ansprechbar und kann diverse Informationen über seinen eigenen Zustand, z.B. über die aktuell erreichte oder die angestrebte Motorposition, liefern. Aus diesen beiden Werten können u.a. Informationen über die Belastung des Motors gewonnen werden. Weiterhin besteht die Möglichkeit einige Eigenschaften, wie z.B. die Drehgeschwindigkeit oder die Steifheit des Servomotors einzustellen. Die Motoren bieten dadurch viel Freiraum in der Entwicklung und liefern anbei wertvolle Informationen, durch die indirekt auf den Gesamtzustand des Roboters geschlossen werden kann.

Die, im Kopf des Roboters montierte Lochkamera, auf Englisch *pinhole camera* liefert

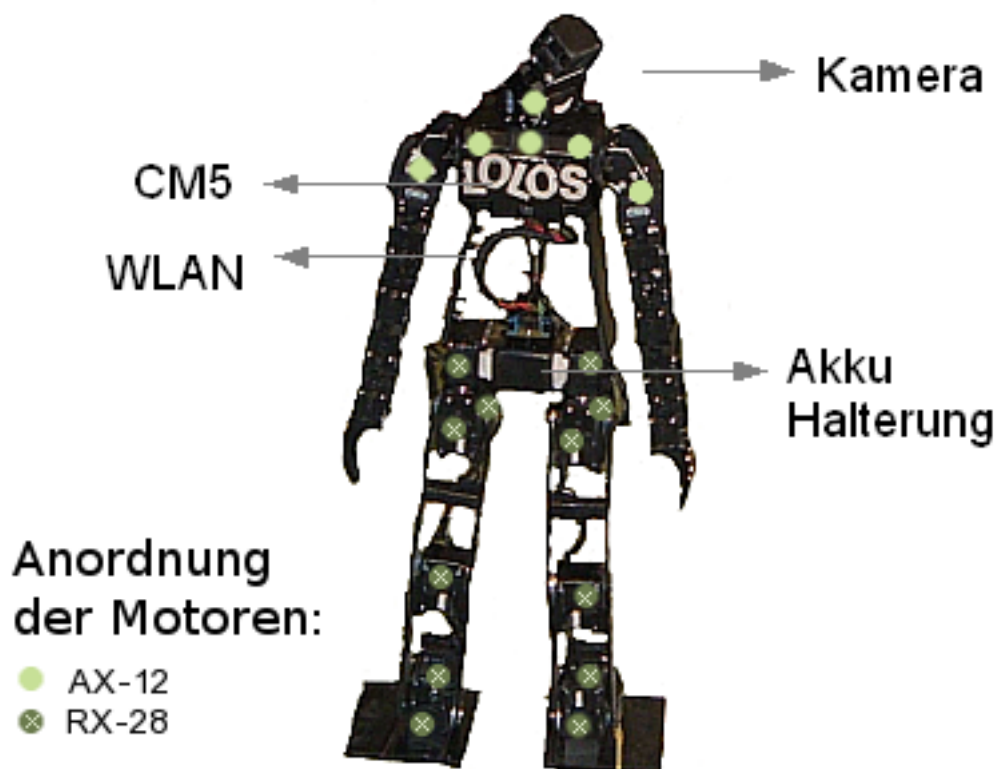


Abbildung 1.3: Ein im FHumanoids Projekt entwickelter Roboter für die Teilnahme am Robocup 2008 China.

achtmal pro Sekunde ein Bild mit einer Auflösung von 160x120 Pixel. Vor der Lochkamera ist ein Objektiv mit einem Öffnungswinkel von 80 Grad angebracht worden. So kann der Roboter mehr von seiner Umgebung aufnehmen.

Direkt mit der Kamera ist ein Mikroprozessor verbunden, welcher eine Vorverarbeitung der Bilddaten und die Manipulierung der Kameraparameter ermöglicht. Das HaViMo [26]

genannte Modul entlastet die Hauptrecheninheit bei der rechen- und speicherintensiven Bilderverarbeitung. Die, von dem Kameramodul gelieferten Bilder, geben Informationen über die Position und die Orientierung des Roboters auf dem Spielfeld. Als Hauptrechen-

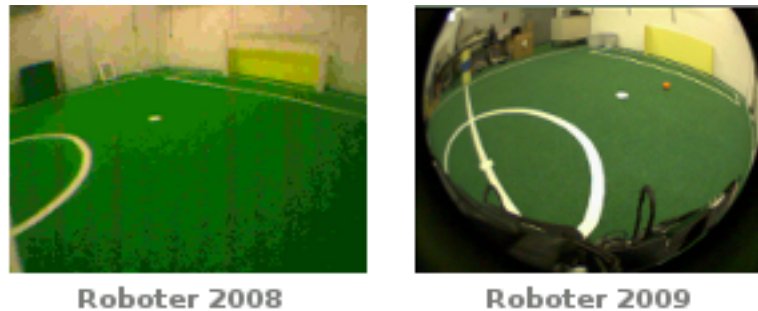


Abbildung 1.4: Ein Vergleich der Kamerabilder von einem Roboter der Plattform 2008 (Links) und 2009 (Rechts).

einheit wird das im Bioloid Kit mitgelieferte CM5 verwendet. Hinter dieser Bezeichnung verbirgt sich ein auf 16 MHz getakteter ATmega 128 Mikroprozessor. Auf dem Gehäuse des CM5 befinden sich Druckknöpfe, die mittels einer Schnittstelle programmierbar sind. Sie bieten eine einfache Möglichkeit für den manuellen Start und Stopp des Roboters.

Ein Vorteil dieser Plattform stellt die Verfügbarkeit vieler Funktionen zur Kommunikation zwischen Rechner und Motoren dar. Außerdem erleichtert das geringe Gewicht des CM5 die Stabilisierung der mechanischen Konstruktion. Sie hat den Nachteil, eine geringe Rechen- und Speicherkapazität zu besitzen und so viele Funktionalitäten im Bereich der Bilderverarbeitung nicht ermöglichen zu können. Auch eine Selbstlokalisierung entfällt aufgrund der geringen Rechenleistung.

Damit die Roboter kommunizieren können, ist ein WLAN Modul in ihrem Oberkörper angebracht worden. So können die Roboter untereinander Pakete austauschen und in einer Spielsituation kabellos Start/Stopp angesteuert werden. Die WLAN Karte ist direkt mit dem CM5 verbunden. Der beschriebene Roboter wird von einem LIPO Akku betrieben, welcher in der Hüfte des mechanischen Skeletts angebracht ist. Der Akku liefert 2000 mA Strom und 12 V Spannung, so kann der Roboter je nach Betriebsart bis zu 60 Minuten versorgt werden.

## Software

Die Entwicklung der Programme erfolgt in der Programmiersprache C, mit Hilfe der Entwicklungsumgebung Codevision. Ein Terminal von Robotis inc. ermöglicht, durch eine serielle Verbindung, den direkten Zugriff auf den CM5. Auf diese Weise kann das neue Programm auf den Mikroprozessor geschrieben und gestartet werden.

Einen wichtigen Bestandteil der Software stellt das im Projekt Fumanoids entwickelte System für Multithreading dar. Es wurde im Rahmen einer Bachelorarbeit [21] speziell für den CM5 der Roboter entwickelt. Insgesamt können bis zu fünf Threads parallel ausgeführt werden. Die in Abbildung 1.5 dargestellte Struktur, beschreibt die Architektur

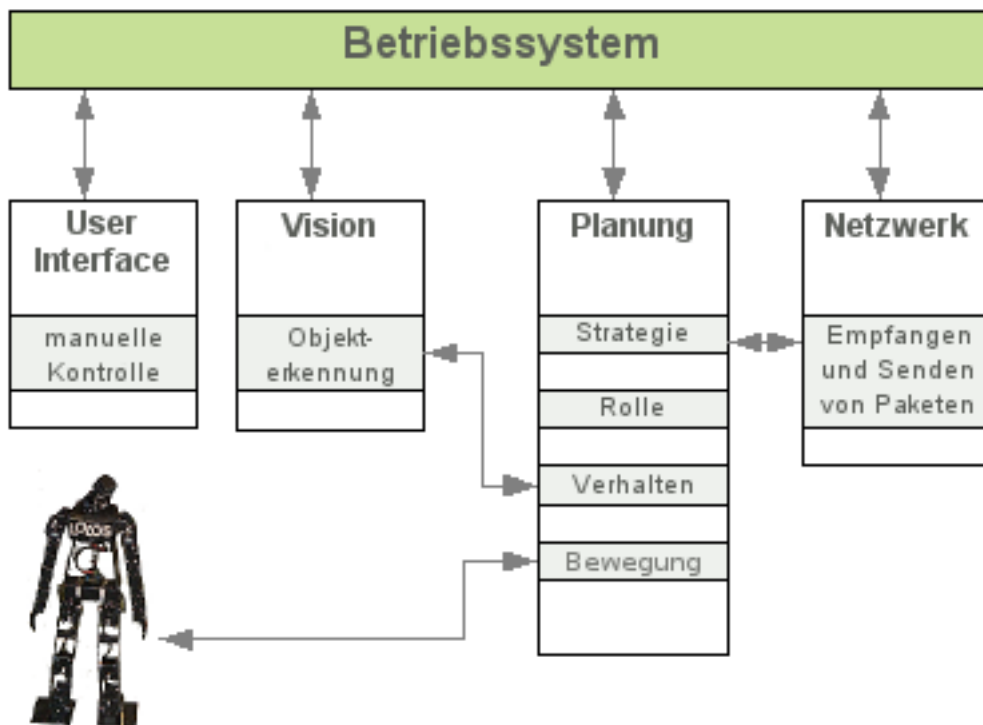


Abbildung 1.5: Die Struktur der Software, auf einem FHumanoids Roboter der Generation 2008

der Robotersoftware. Das Programm lässt sich in fünf Komponenten: Betriebssystem, User Interface, Vision, Planung und Netzwerk aufteilen.

- Das **Betriebssystem** beherbergt die, für die Hardware notwendigen Einstellungen, wie z.B. die Steifheit jedes Motors. Es ist ebenfalls zuständig für die Initialisierung des Systems für Multithreading und das Starten der Kamera-, Netzwerk-, und Planungsmodule.
- Das **User Interface** repräsentiert die Benutzerschnittstelle des Roboters. Hier können für unterschiedliche Eingaben diverse Programmteile, wie z.B. Testfälle gestartet werden. Dieser Systemabschnitt reagiert auch auf die Aktivierung der Druckknöpfe.
- Die **Vision** ist für die Verarbeitung, der von der Kamera gelieferten Bilder, zuständig. Sie identifiziert anhand der Farbwerte wichtige Elemente wie z.B. das Tor, den Ball, die Säulen, Gegner bzw. Hindernisse und stellt sie anschließend zur allgemeinen Nutzung bereit.

- Die **Planung** stellt die künstliche Intelligenz des Roboters dar. Hier werden alle Sensorinformationen gesammelt, um auf deren Grundlage die Entscheidungen für die nächste Handlung des Roboters zu treffen. Weiterhin werden die, über das Netzwerk erhaltenen Daten anderer Roboter verwertet um eine Kooperation zwischen den Spielern zu ermöglichen. Dieses System ist im Rahmen dieser Diplomarbeit entwickelt worden.
- Das **Netzwerk** stellt die Verbindung zwischen der am Roboter angebrachten WLAN Karte und dem Access Point her. Sie ermöglicht den Austausch von Datenpaketen zwischen zwei Robotern sowie zwischen einem externen Rechner und einem Roboter. Das Modul konstruiert die Datenpakete und gewährleistet ihren sicheren Transfer. Ein Paket kann z.B. die Zustandsdaten des Roboters oder auch die Informationen über dessen Umfeld beinhalten.

### 1.3.2 Die Roboterplattform 2009

#### Hardware

Der neue Roboter stellt eine vollständige Eigenkonstruktion dar und wurde im Rahmen mehrerer Abschlussarbeiten entwickelt. Das mechanische Skelett besitzt die selben Proportionen, bestehend aus einem großzügig gebauten Unterkörper und einem kleineren Oberkörper. Es werden die stärkeren Modelle RX-28 und RX-64 von den Robotis inc. Dynamixel Motoren verwendet. Die Eigenschaften dieser Servomotoren werden in der Abbildung 1.6 dargestellt.

Der Einsatz von leistungsfähigeren Motoren im Unterkörper und schwächeren im Oberkörper hat sich bei der vorherigen Robotergeneration als Vorteilhaft erwiesen. So verfolgt auch die aktuelle Generation dieselbe Art der Einteilung von Servomotoren. Insgesamt sind in einem Roboter einundzwanzig Motoren vorzufinden, sieben Dynamixel RX-28 und vierzehn RX-64. Die Verbindungen zwischen den Motoren sind speziell entworfen und angefertigt worden. Dabei sind wie im vergangenen Jahr schwarze Kunststoffplatten diverser Breite eingesetzt worden. Auch die ergänzenden Körperteile, wie Füße, Arme, Akkuhalterung bestehen ebenfalls aus diesen Kunststoffplatten.

Ein wichtiges Konstruktionsmerkmal im Unterkörper ist der Aufbau jedes einzelnen

Motorart	Gewicht (g)	Getriebeübersetzung	Max Drehkraft (Nm)	Geschwindigkeit (s/60°)	Auflösungsgrad
Dynamixel RX-28	72	1:193	28.3 @12V	0.167	0.35
Dynamixel RX-64	125	1:200	64.4 @15V	0.188	0.35

Abbildung 1.6: Die Eigenschaften der eingesetzten Dynamixel RX-28 und RX-64 Motoren [19].

Kniegelenkes aus zwei Motoren. Die sogenannten doppelten Knie ermöglichen die symmetrische Bewegung der Beine sowohl nach Vorne als auch nach Hinten. Dies bringt beim

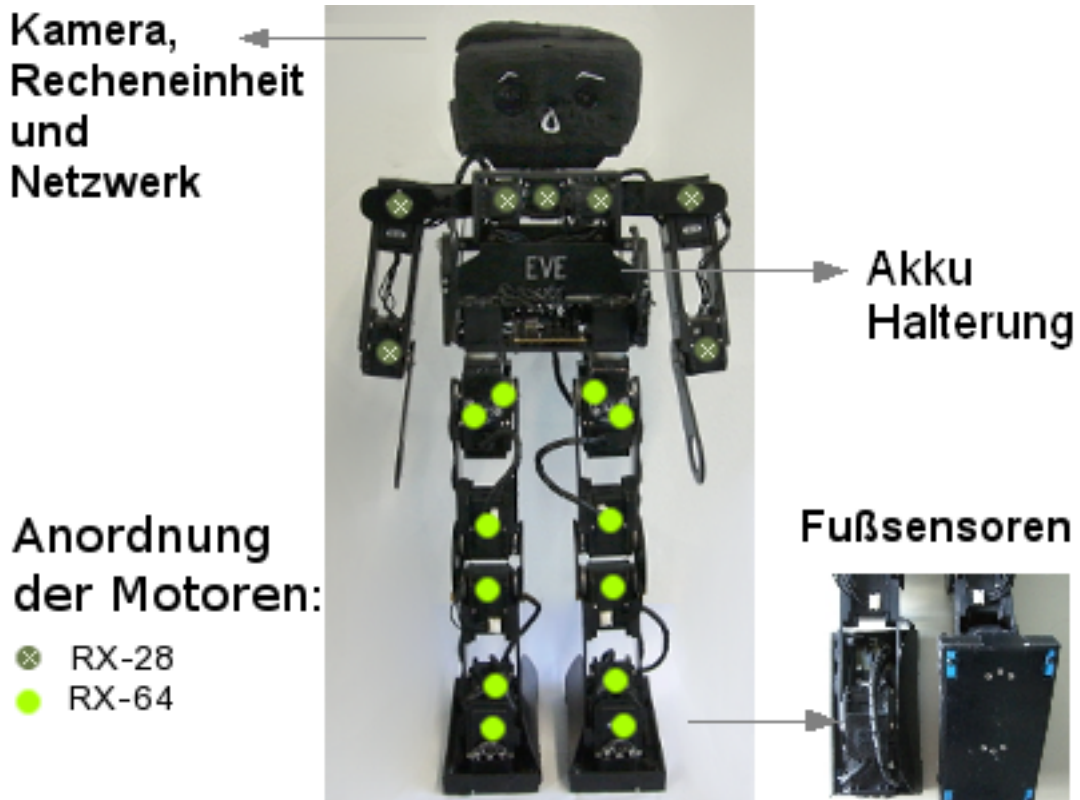


Abbildung 1.7: Ein Roboter der FUmoids entwickelt für die Teilnahme am Robocup 2009 Graz.

Laufen große Vorteile mit sich. Ein omnidirektionales Laufen wird so in allen Richtungen gleich gut umsetzbar, wobei die doppelten Knie die Geschwindigkeit der Bewegungsabläufe erhöhen.

Eine weitere Neuheit stellt die, in den Fußplatten eingebauten, Drucksensoren dar. Sie liefern die Information, wann welcher Fuß auf dem Boden aufgesetzt ist. Sie ist von Bedeutung für die Stabilisierung der Laufbewegung sowie für die Erkennung von Stürzen. Im Vergleich zu der älteren Generation zeigen sich Unterschiede an den beiden Ellbogengelenken die, zur Unterstützung der Aufstehbewegung benötigt werden. Ebenfalls neu ist die Platzierung des Akkus und die Abwesenheit der Recheneinheit in der Brust. Auch bei dem aktuellen Modell besteht die Hauptrolle des Oberkörpers darin die Bewegungen zu stabilisieren.

Die Abbildung 1.8 zeigt die Konstruktion des Kopfes bzw. des Kameramoduls, bestehend aus der Recheneinheit, der Kamera und der Netzwerkkarte. Das Kameramodul, das gleichzeitig zwei Lochkamas betreiben kann, ist im Rahmen einer Diplomarbeit [9] entstanden. Da die Entfernung zwischen den beiden Kameras gering ist, ist keine Stereovision möglich. Aus diesem Grunde wird nur das Signal einer Kamera verarbeitet. Sie liefert bis zu zwölf Mal pro Sekunde ein Bild, mit einer Auflösung von 640x480. Vor der

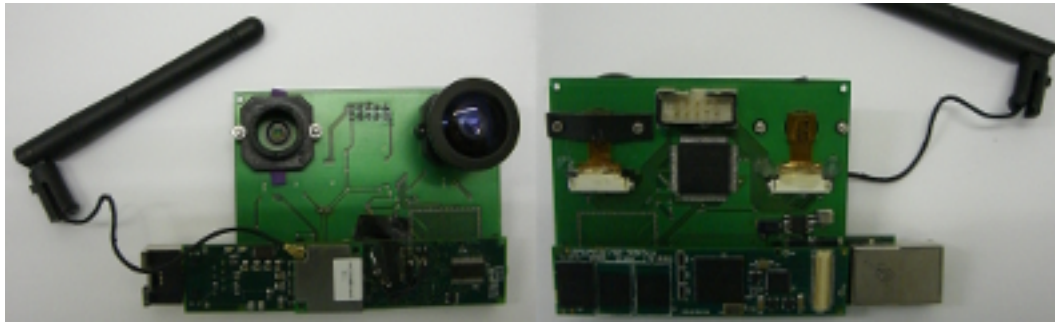


Abbildung 1.8: Das Kameramodul des Roboters zusammen mit der Recheneinheit und dem Netzwerkmodul.

oben genannten Kamera ist ein Fischauge der Firma Lensation [28], mit einem Öffnungswinkel von knapp unter 180 Grad montiert worden.

Im Vergleich zu der älteren Konstruktion findet hier keine Vorverarbeitung der Bilder statt. Der Grund liegt in der ausreichenden Leistungs- und Speicherkapazität der Hauptrecheneinheit. Hierbei handelt es sich um ein Verdex Pro XL6P der Firma Gumstix [27] mit einer Leistung von 600 MHz XScale ARM Prozessor, 128 MB RAM und 32 MB Flash. Dank dieser Voraussetzung ist es möglich gewesen eine Selbstlokalisierung für den Roboter zu gewährleisten. An dem Verdex Pro ist eine Netzwerkkarte derselben Firma zusammen mit dem dazugehörigen WLAN Modul angeschlossen worden. Mittels dieser Konstruktion können die Roboter untereinander kommunizieren. Der Roboter wird von einem LIPO Akku betrieben, der 3200 mA Strom und 14 V Spannung liefert. So kann der Roboter je nach Betriebsart bis zu 5 Stunden versorgt werden.

## Software

Die optimierte Hardware des neuen Roboters eröffnet eine Vielzahl von Möglichkeiten in der Softwareentwicklung (z.B. Selbstlokalisierung). Auf dem Verdex Pro XL6P ist eine OpenEmbedded [29] Linux Distribution installiert worden. So wird die individuelle Konfiguration von dem Linux Kernel ermöglicht. Die Linux Distribution hat einen Wechsel auf die Programmiersprache C++ ermöglicht, die große Vorteile durch Abstraktion, Vererbung und Datenkapselung bietet.

Im Rahmen einer Studienarbeit [23] ist die Software der früheren Generation auf das neue System migriert worden. Dabei ist besonders auf die Erhaltung der Architektur und der Planungsmodule geachtet worden. In die Software ist eine Komponente für die Motorkommunikation und Motoransteuerung integriert worden. Eine Ansteuerung der Motoren erfolgt alleine über diese Komponente.

Eine Erweiterung erfolgt durch das System zur Selbstlokalisierung, das die Bestimmung von globalen Positionsdaten ermöglicht. Dieses System ist im Rahmen einer Diplomarbeit [13] umgesetzt worden.

Um den Roboter zu steuern oder neu zu programmieren wird nicht mehr die serielle Verbindung verwendet sondern das Netzwerk.

# 2 Verhaltensarchitekturen für autonome Roboter

## 2.1 Der biologische Hintergrund

Die Natur stellt die wichtigste Inspirationsquelle für die Forschung dar. Für die Robotik sind die Menschen und die Tiere die interessantesten Forschungsmodelle. Besonders der Mensch kennzeichnet sich durch das Gehen auf zwei Beinen, seiner Fähigkeit zu lernen, seinen Verstand und seine Psyche. Diese Eigenschaften machen ihn zu einer besonderen Schöpfung der Natur. Bereits seit Jahren versuchen Wissenschaftler Roboter mit menschlichen Fähigkeiten auszustatten. Beispielsweise ist in Japan ein Roboter, mit sozialen Kompetenzen gebaut worden [16] und [24]. Auch in den USA ist eine psychologische Theorie *The Theorie of Mind* auf einem Roboter umgesetzt worden [22]. Es sind Roboter gebaut worden, die das Verhalten von bestimmten Tierarten nachahmen können [12]. Ein weiteres Beispiel ist das Modell der Kooperation zwischen Robotern, das durch die Kommunikation zwischen Wölfen oder Bienen inspiriert worden ist [8].

## 2.2 Konzepte der Planung

### 2.2.1 Reaktiv

Das Reiz- Reaktion-Prinzip gilt sowohl beim Tier als auch beim Menschen und beschreibt das spontane Verhalten auf ein unerwartetes Geschehen. Die Form der Reaktion kann nicht geplant werden, da keine Informationen über das Ereignis vorliegen. Beispielsweise wird ein Lebewesen in einer lebensbedrohlichen Situation nicht stehen bleiben, um über die nächste Handlung zu grübeln, sondern spontan flüchten. Jede Sekunde die mit Nachdenken verbunden ist, bedeutet Zeitverlust und erhöht die Wahrscheinlichkeit der Situation zu erliegen.

Eine reaktive Planung arbeitet nach dem Muster dieses biologischen Prinzips. Anhand einer Momentaufnahme aus der subjektiven Perspektive des Roboters wird die Entscheidung, über die nächste Handlung getroffen. Durch die Einschränkung der Momentaufnahme, kann nur eine kleine Menge an Parameter, für die Planung einer Reaktion verwendet werden. Oft reicht schon eine einzelne Variable aus, um eine Reaktion auszulösen. Eine ausführliche Beschreibung der reaktiven Planung kann in dem Buch *Behavior-Based Robotics* [2] nachgelesen werden.

Das Beispielszenario, eines sehr einfachen reaktiven Verhaltens, wird in der Abbildung 2.1 dargestellt. Die Aufgabe des Roboters besteht darin, einen Weg zu finden, auf dem er das Labyrinth verlassen kann. Das Verfahren bietet aber leider nicht immer eine Lösung.

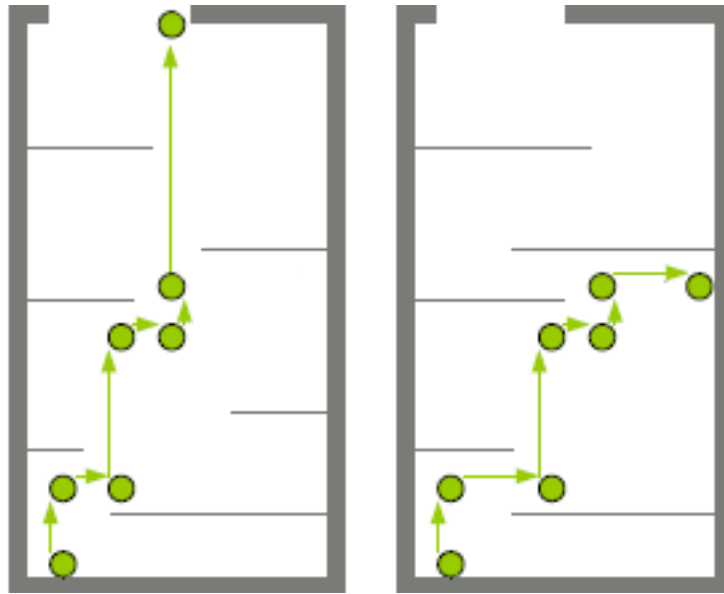


Abbildung 2.1: Darstellung einer erfolgreichen reaktiven Planung Links und einer erfolglosen rechts. Die reaktive Planung erfolgt nach den folgenden zwei Regeln:  
 1. Roboter bewegt sich gerade aus; 2. Trifft er auf ein Hindernis, bewegt er sich nach Rechts.

### Vorteile

Durch die Entscheidungsfindung anhand einer Momentaufnahme von der Situation ist eine Zwischenspeicherung alter Werte nicht mehr nötig. Infolgedessen braucht eine reaktive Planung wenig Speicherplatz. Die Entscheidungen können sofort anhand möglichst einfacher Kriterien getroffen werden, so dass auf hohe Rechenkapazität verzichtet werden kann. Eine der wichtigsten Vorteile dieser Planung ist die schnelle Antwortzeit. Viele Systeme können nicht lange auf eine Entscheidung warten, wie z.B. die Echtzeitsysteme. Aus diesem Grund wird das System häufig eingesetzt.

### Nachteile

Der reaktive Ansatz liefert nicht immer eine Lösung für das Gesamtproblem, obwohl ein Lösungsweg existiert. Der Grund hierfür liegt darin, dass das Problem aus der subjektiven Perspektive des Roboters betrachtet wird. Es wird eine Lösung alleine für ein Unterproblem erstellt. Dies kann sich negativ auf die Gesamtlösung auswirken. Ein Beispiel hierfür ist in der Abbildung 2.1 dargestellt.

### 2.2.2 Deliberativ

Im Gegensatz zu dem reaktiven Ansatz handelt es sich bei dem deliberativen Ansatz um ein planendes Konzept. Dieser Ansatz benötigt eine globale Sicht auf die Welt. Von dieser Perspektive aus wird eine interne Repräsentation der Welt erzeugt, welches als Grundlage der Planung dient. Mit Hilfe von Planungsalgorithmen können so unterschiedliche Lösungen für ein Problem gefunden werden. Beispielsweise könnte für das Problem der Wegfindung ein Pfad mittels A\* Algorithmus und Floyd Warshall berechnet werden. So besteht auch die Möglichkeit den optimalen Lösungsweg zu finden.

In Abbildung 2.2 ist die Problematik des Roboters im Labyrinth mit einem deliberativen Konzept gelöst worden. Die Grafik deutet an, dass es sich hierbei um eine Reihe von komplexen Berechnungen handelt. Zu berücksichtigen ist, dass es sich hier um einen diskreten Raum handelt. Tatsächlich steigt die Komplexität in einem kontinuierlichen Raum sehr stark an, siehe hierzu [7]. Es gibt zwei Möglichkeiten zur Minimierung der Komplexität: indem das Verfahren randomisiert oder der Aktionsraum diskretisiert wird.

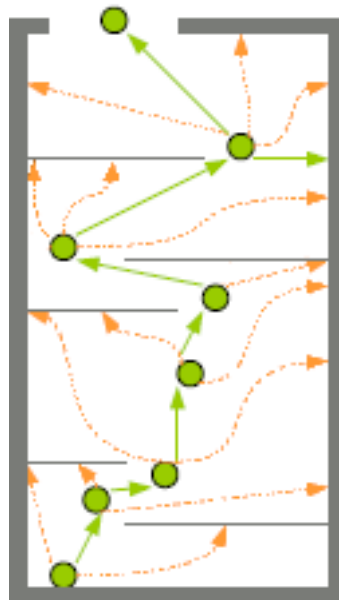


Abbildung 2.2: Eine deliberative Planung nach den folgenden zwei Regeln: 1. Suche alle Wege ab (gepunktete Pfeile); 2. Geh den kürzesten Weg zum Ziel (durchgezogene Pfeile).

#### Vorteile

Der größte Vorteil dieses Konzeptes besteht darin, dass sofern ein Lösungsweg existiert, dieser garantiert gefunden wird. Sollte es keine Lösung geben, so wird auch diese Information erhalten. Außerdem bietet das deliberative Konzept die Möglichkeit die optimale

Lösung zu berechnen. Die Voraussetzung hierfür ist eine qualitative Repräsentation der Welt.

### **Nachteile**

Wie bereits erwähnt, steigt die Komplexität eines deliberativen Ansatzes schnell an. Besonders beim autonomen mobilen Roboter wird dies schnell zum Problem, da sie in einem kontinuierlichen Raum agieren. Die Planungsalgorithmen benötigen viel Speicher- und Rechenkapazität, womit die Großzahl der autonomen Konstruktionen nicht dienen können. Einige Systeme scheitern bereits an der globalen Repräsentation der Welt.

Ein zusätzlicher Nachteil besteht darin, dass keine schnellen Ergebnisse erwartet werden können. Eine perfekte Planung nimmt viel Zeit in Anspruch, etwas das beispielsweise für Echtzeitsysteme nicht akzeptabel ist.

### **2.2.3 Hybrid**

Durch die sich ergänzenden Eigenschaften des reaktiven und deliberativen Ansatzes, liegt die Idee einer Kombination sehr nahe. Das Resultat ist ein hybrider Ansatz. Ein guter hybrider Ansatz wird dadurch ausgezeichnet, dass er möglichst viele Vorteile und möglichst wenig Nachteile von beiden Konzepten übernimmt. Das Beispiel für einen hybriden Ansatz, des in Abschnitt 2.2.1 vorgestellten Problems, wird in der Abbildung 2.3 dargestellt.

Das deliberative und das reaktiven Konzept können auf unterschiedliche Arten verknüpft werden. Besitzt ein System ausreichende Rechenkapazität und ist es nicht sofort auf eine Antwort der Planung angewiesen, so können mehr Entscheidungen von dem deliberativen Modul umgesetzt werden. Üblicherweise werden Antworten jedoch in Echtzeit erwartet, weshalb der reaktiven Komponente Vorrang gegenüber dem deliberativen Modul gegeben wird. Der Grund hierfür ist die schnelle Reaktionszeit der reaktiven Komponente, sowie die Tatsache, dass weniger Ressourcen benötigt werden. Somit erhält bei der Planung die deliberative Komponente eine beratende Rolle. Es besteht ebenfalls die Möglichkeit beiden Modulen gleiche Rechte zu geben, wobei sich die Systeme gegenseitig kontrollieren.

### **Vorteile**

Der Entwickler kann dem deliberativen und reaktiven Modul jeweils einen Faktor für ihre Wichtigkeit zuordnen. So lassen sich individuelle Konzepte leicht entwickeln und den Anforderungen anpassen.

### **Nachteile**

Für die deliberative Komponente muss eine globale Perspektive erzeugt werden, die aber eine bestimmte minimale Rechen- und Speicherkapazität erfordert. Demzufolge ist ein hybrides Konzept nicht für alle Systeme geeignet.

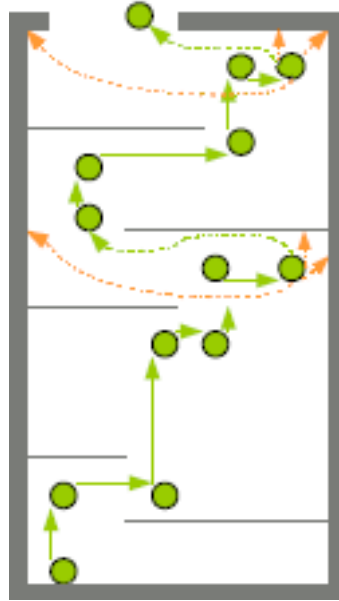


Abbildung 2.3: Ein hybrides Konzept, bei welchem der reaktive Teil (durchgezogene Pfeile) folgenden zwei Regeln folgt: 1. Gehe gerade aus; 2. Wenn Hindernis -> gehe Rechts; Der deliberative Teil (gepunktete Pfeile) definiert die dritte Regel: 3. Wenn der Weg nach unten führt -> Suche den Weg, der wieder nach oben führt.

Im Falle der FHumanoids verfolgt die Planung einen hybriden Ansatz, bei der die deliberative und reaktive Komponente gleichberechtigt sind. Die Architektur ist so konzipiert, dass die Planung gleichzeitig ein globales Ziel verfolgen und von reaktiven Systemteilen beeinflusst werden kann.

## 2.3 Relevante Planungsarchitekturen

Die relevanten Planungsarchitekturen sollen einerseits zum besseren Verständnis der Möglichkeiten, Planungskonzepte umzusetzen und andererseits als Vergleich zu der im Kapitel 3 beschriebenen Planungsarchitektur der FHumanoids dienen.

### 2.3.1 Subsumption Architektur

Die Subsumption Architektur ist eine Planungsarchitektur nach dem reaktiven Ansatz. Sie ist von Brooks [6] 1986 als einer der ersten verhaltensbasierten Architekturen veröffentlicht worden. Das System ist in hierarchisch geordnete Module strukturiert, wobei jedes Modul als endlicher Zustandsautomat umgesetzt wird. Eine Darstellung der Ebenen ist in der Abbildung 2.4 zu sehen. Für die Konstruktion der Ebenen sollen folgende Regeln beachtet werden:

1. Als erstes wird die Ebene Null konstruiert.
2. Erst wenn eine Ebene vollständig abgeschlossen ist, kann die Konstruktion einer übergeordneten Ebene angefangen werden.
3. Eine abgeschlossene Ebene darf nicht verändert werden.

Die Regeln gewährleisten die Konstruktion eines stabilen Systems, bei dem die bereits abgeschlossenen Komponenten als funktionsfähig und fehlerfrei gelten. Bei der Entwicklung kommt oft, erst in einer fortgeschrittenen Phase, der Bedarf nach bestimmten Features auf. In dieser Architektur ist die Integration solcher Features im Nachhinein nicht mehr möglich. Das Problem wird versucht durch bestimmte Rechte, die die übergeordnete Ebene über die untergeordnete Ebene besitzt, zu kompensieren. Im Folgenden werden diese Rechte aufgezählt:

- Die übergeordnete Ebene  $n$  darf die Ausgabe der unteren Ebene unterdrücken und ersetzen. Sie übt damit eine *Suppression* auf die untere Ebene aus.
- Die übergeordnete Ebene kann die Ausgabe der unteren Ebene verhindern. Dies wird als *Inhibition* bezeichnet.

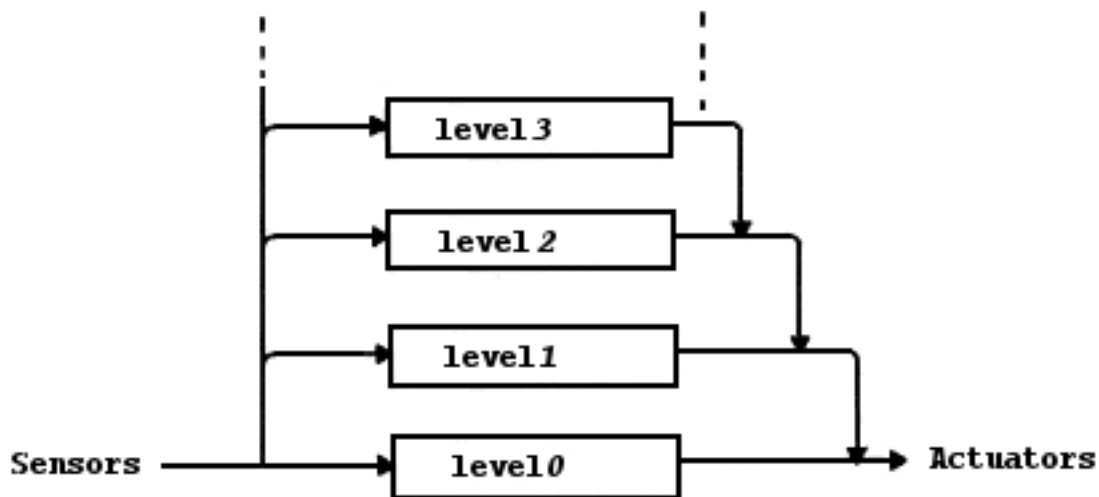


Abbildung 2.4: Die Darstellung einer Subsumption Architektur [6].

Die Hierarchie wächst von unten nach oben und gewinnt mit jeder Ebene an Fähigkeiten dazu. Außerdem steigt mit jeder Hierarchiestufe der Grad der Abstraktion. Die Ebene *Null* ist also die primitivste während die Ebene  $n$  die komplexeste ist.

### 2.3.2 Hierarchical dynamical systems to control reactive behavior

Die von Behnke [5] veröffentlichte Architektur stellt eine Erweiterung, der von Jäger entworfenen reaktiven Dual Dynamics [11] Architektur dar. Der Vorteil besteht darin,



- Die **physische Aktoren** können Veränderungen in der Umwelt erzeugen. Sie dürfen nur von der untersten Ebene angesprochen werden.
- Die **nicht physischen Aktoren** stellen eine virtuelle Konstruktion dar. Sie können nur in den höheren Ebenen eingesetzt werden, um Werte an die Sensoren einer unteren Ebene zu senden. Dies ist in der Abbildung 2.5 als *Internal Feedback* dargestellt.

Ein Verhalten wird in Form einer Schleife implementiert wobei, je nach Höhe der Ebene, eine andere Wartezeit innerhalb der Schleife eingesetzt wird.

### 2.3.3 Autonomus Robot Architecture (AuRA)

Das hybride System ist das erste Mal 1987 [1] und genau zehn Jahre später in einer überarbeiteten Form [3] erneut veröffentlicht worden. In den nächsten Zeilen wird die aktuellere Auflage der AuRA Architektur beschrieben.

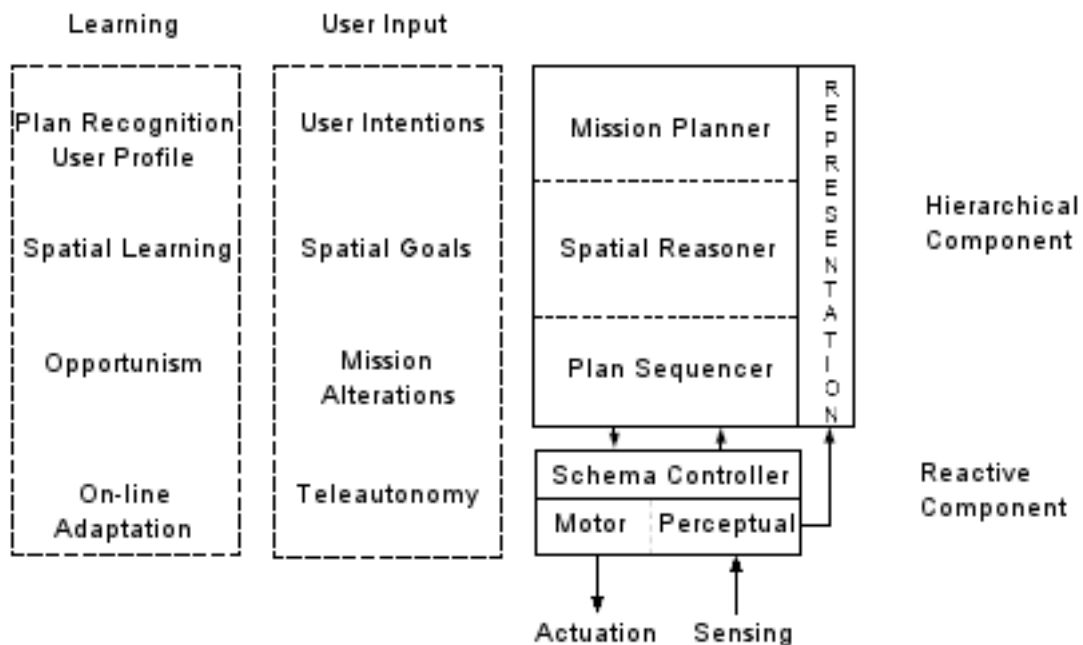


Abbildung 2.6: Die Architektur AuRA von R. C. Arkin [1].

Wichtig für das Verständnis dieser Architektur ist es, erst einmal, den von Arkin eingeführten Begriff *motor schema*, zu verstehen. Hierbei handelt es sich um einen sehr verbreiteten Ansatz durch den Systeme, die das Ausweichen von Hindernissen ermöglichen, realisiert werden. Das Verfahren wird auch für die Planung der FUManoide zu diesem Zweck eingesetzt.

**Ein Motor Schema** ist die Repräsentation von Verhaltensmuster in Form von Vektorfelder. Hierfür werden Objekte in der Umgebung mit positiven oder negativen Ladungen versehen. Alle Ziele erhalten eine parametrisierte Anzahl positive Ladungen und die Hindernisse negative Ladungen. Durch die Aufeinanderwirkung von Kräften entstehen Magnetfelder, deren Feldlinien das Vektorfeld definieren. Ein Verhaltensmuster ist eine Aktion in der Welt, das als ein Vektorfeld oder eine Kombination mehrerer Vektorfelder dargestellt werden kann. Siehe hierzu die graphische Darstellung 2.7. Motor Schemas sind voneinander unabhängig und deswegen können mehrere Schemas parallel zueinander aktiviert werden. Die Berechnung der Bewegungsrichtung  $\vec{v}$ , aus mehreren parallelen Schemas erfolgt folgendermaßen:

$$\vec{v} = f(g_1\vec{v}_1 + g_2\vec{v}_2 + g_3\vec{v}_3 + \dots + g_n\vec{v}_n) \quad (2.1)$$

$\vec{v}_i$ : der Repräsentant des Vektorfeldes  $i = 1 \dots n$

$g_i$ : Gewicht des Vektors  $\vec{v}_i$

$f(x)$ : Normalisierungs- Funktion

Aus jedem Vektorfeld  $i$  wird ein Vektor  $\vec{v}_i$  ausgesucht, der am besten die Ziele des Verhaltens repräsentiert. Dieser Vektor wird mit dem Gewicht  $g_i$  versehen und geht dann in die Berechnung der Bewegungsrichtung  $\vec{v}$  mit ein. Die Gewichtung der Vektoren bestimmt wie wichtig das repräsentierte Verhaltensmuster ist. Die Gewichte werden üblicherweise von dem Entwickler festgelegt, können aber auch dynamisch gesetzt werden.

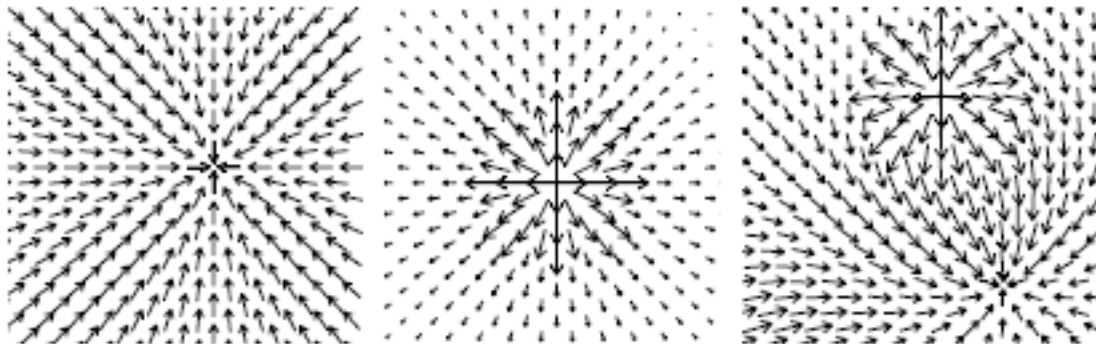


Abbildung 2.7: Beispiele für Motor Schema [4]: Das Vektorfeld auf der linken Seite stellt ein Schema für das *move-to-goal* Szenario dar. Hier wird der Roboter vom Ziel angezogen. In der Mitte ist ein Feld für *avoid-obstacle* zu sehen, wo die Vektoren den Roboter vom Hindernis wegstoßen. Das rechte Vektorfeld zeigt die Kombination *move-to-goal-with-obstacle-avoidance*.

Auch der Input von Sensoren wird in Schemas auf Englisch *perceptual schema* umgewandelt (übersetzt: Wahrnehmungsschema).

## Die deliberative Systemkomponente

Die deliberative Systemkomponente ist ein dreistufiges hierarchisches System, das nach seinem Abstraktionsgrad geordnet ist. Folgende Komponenten sind am System beteiligt:

1. Der **Mission Planner** ist für die Planung eines Lösungsweges auf der abstraktesten Systemebene zuständig. Hier werden bereits Informationen über die Fähigkeiten des Roboters berücksichtigt. Die Ausgabe des Systems ist dafür zuständig das nächste Ziel des Roboters anzugeben.
2. Der **Spatial Reasoner** wird in der ersten Implementierung als Navigator bezeichnet. Mittels Kartographie wird der Weg des Roboters zu einem Zielpunkt berechnet. Als Resultat wird eine Sequenz von Vektoren in Richtung Ziel zurückgegeben.
3. Der **Plan Sequencer** wird in der ersten Implementierung als Pilot bezeichnet. Er wandelt eine Sequenz von Vektoren in die entsprechenden Verhaltensmuster bzw. *motor schemas* um. Diese Sequenz wird der reaktiven Komponente übergeben.

## Die reaktive Systemkomponente

Die Aufgabe der reaktiven Komponente ist es, das System zu überwachen und zu kontrollieren. Sie ordnet jedem Verhaltensmuster, die für sie notwendigen Informationen der Sensoren zu. Dies geschieht auf Basis der Vektordarstellung von *motor schema* und *perceptual schema*. Anhand der Gleichung (2.1) wird anschließend die Bewegungsrichtung des Roboters berechnet. Die reaktive Komponente wird auch als *Schema Controller* bezeichnet.

### 2.3.4 Hybrid Mobile Robot Architecture with Integrated Planning and Control.

Die von Kian Hsiang Low [14] entwickelte hybride Architektur ist speziell für autonome mobile Roboter entwickelt worden. Der Entwicklung liegt das Szenario der Wegfindung in einer Welt, mit nicht vorhersehbaren Ereignissen, zugrunde. Dabei soll der Roboter auf Hindernisse achten und ihnen rechtzeitig ausweichen.

Wie in der Abbildung 2.8 zu sehen ist, gibt es eine klare Trennung der deliberativen von der reaktiven Komponente. Weiterhin ist zu erkennen, dass die Berechnungen der deliberativen Planung als Eingabe für das reaktive System dienen. Auf diesem Wege bekommt der reaktive Teil des Systems eine höhere Priorität zugewiesen. Er besitzt die Entscheidungsfreiheit über die Ergebnisse der Planung.

## Die deliberative Systemkomponente

Die deliberative Systemkomponente ist für die langfristige Planung eines Weges, vom Startpunkt *A* zum Zielpunkt *B*, zuständig. Die Berechnung erfolgt auf der Grundlage einer Repräsentation der Welt, das dem System als Eingabe übergeben wird. Als Resultat wird eine Sequenz von sogenannten, *Checkpoints* erzeugt. Dabei gilt, dass die Gesamtheit aller Checkpoints den Weg von Punkt *A* nach Punkt *B* beschreibt.

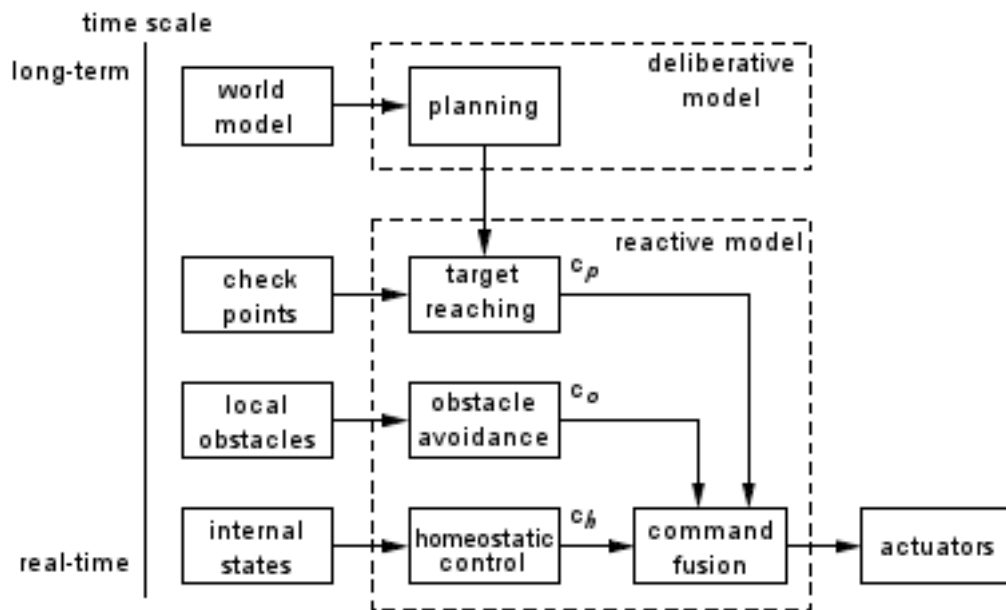


Abbildung 2.8: Eine Darstellung der von Low [14] veröffentlichten hybriden Architektur.

### Die reaktive Systemkomponente

Die reaktive Systemkomponente entscheidet über die nächste Handlung des Roboters und steuert die Aktoren an. An ihr ist eine direkte Verbindung zu den Aktoren des Roboters gekoppelt. Da keine andere Systemkomponente einen Zugriff auf die Aktoren besitzt, müssen alle anderen Berechnungen den Weg durch das reaktive System gehen. Die Komponente erhält als Eingabe:

- Die, von der deliberativen Komponente berechneten Checkpoints,
- Die aktuellen Informationen über die anwesenden Hindernisse,
- Die Informationen über den internen Zustand des kompletten Robotersystems.

Durch diese Daten wird der Weg zum Zielpunkt neu geplant. Den Hindernissen, die sich im Weg des Roboters befinden, wird dabei ausgewichen. Der Roboter kann durch den Einsatz eines neuronalen Netzes zur Kontrolle der Fortbewegungs-Aktoren auf einen sanften Weg durch die *Checkpoints* geführt werden.

## 2.4 Diskussion

Aus den vorgestellten Architekturen wird die Verteilung der reaktiven Planungsmodule auf unterschiedliche Abstraktionsebenen ersichtlich. Eine Ausnahme stellt die *AuRA* Architektur dar, die eine Abstraktionshierarchie in der deliberativen Komponente besitzt.

Bei allen Systemen wird die Idee, die primitivste Ebene mit den Aktoren in Verbindung zu setzen, unterstützt. Ausschließlich die Module dieser Ebene besitzen das Recht die Aktoren anzusteuern. Diese Struktur schafft Transparenz in der Softwarearchitektur und vereinfacht so die Systementwicklung.

Im Gegensatz zu den Aktoren werden die Sensoren für alle Hierarchieebenen zugänglich gemacht, wie im Falle der *Subsumption* Architektur. Zusätzlich besitzen einige Architekturen virtuelle Sensoren, die alleine ausgewählten Hierarchieebenen Informationen liefern dürfen. Zu diesen Architekturen gehört die *Hierarchical dynamical systems to control reactive behavior* Architektur, die aus Sensordaten sogenannte Wahrnehmungen berechnet. Diese Wahrnehmungen stehen alleine den höheren Ebenen zur Verfügung. Ähnlich ist es bei den hybriden Architekturen, wie der *Hybrid Mobile Robot Architecture with Integrated Planning and Control*. Dort dient das Ergebnis der deliberativen Komponente als Eingabe der höchsten reaktiven Hierarchieebene.

Im Falle der reaktiven Systeme erfolgt die Steuerung der unteren Module durch die übergeordneten Module. Dasselbe Prinzip kann in der mehrschichtigen deliberativen Komponente von *AuRA* wiedergefunden werden. Bei beiden hybriden Systemen hat die reaktive Planung die letzte Entscheidung über die nächste Aktion des Roboters. Allerdings wird bei *AuRA* deutlich mehr Vertrauen in das deliberative System gesetzt als bei der in Abschnitt 2.3.4 vorgestellten Architektur.

# 3 Die Architektur des Planungssystems

## 3.1 Concurrent Scenario-Based Planning (CSBP)

Die CSBP ist eine noch unveröffentlichte hybride Planungsarchitektur von H. Moballegh [17], die auf der Grundlage von Szenarien, siehe Abschnitt 3.1.1 aufgebaut ist. Anhand der dort vorgestellten Definition kann dem Szenario eine planende Eigenschaft zugeordnet werden. Dadurch stellt es die *deliberative Komponente* der hybriden Architektur dar. Die Szenarien werden in einer Abstraktionshierarchie, ähnlich der in Abschnitt 2.3 vorgestellten Architekturen von Brooks [6], Behnke [5] und Arkin [3] aufgeteilt. Im Vergleich zu der steigenden Abstraktion ist ein gleichzeitiger Abstieg der Hardwareabhängigkeit zu beobachten, siehe Abbildung 3.1. Eine wichtige Eigenschaft der Architektur besteht darin, dass einer Ebene mehrere Szenarien zugeordnet werden können.

Alle Ebenen der Hierarchie können gleichzeitig ausgeführt werden, da ihnen mindestens



Abbildung 3.1: Die Hierarchie des CSBP: Der Abstraktionsgrad steigt während die Hardwareabhängigkeit sinkt.

ein eigener Thread zugeordnet wird. Alle Threads werden mit demselben Takt abgearbeitet und bekommen gleich viel Speicherplatz zugewiesen. Auf diese Weise ist eine Gleichberechtigung der Ressourcen für jede Ebene hergestellt. Das System für Multithreading ermöglicht es alle Ebenen durchgehend aktiv zu halten. Folglich können alle wichtigen Faktoren wie z.B. der Zustand des Roboters oder die Veränderungen der Umwelt permanent wahrgenommen und die entsprechenden Reaktionen ausgelöst werden. Hier wird die *reaktive Komponente* der hybriden Architektur erkennbar. Bei der CSBP Architektur wird das deliberative System mit dem reaktiven System verschmolzen. Abhängig vom Zustand des Roboters und der Situation teilen sich die beiden Systeme die Steuerung. Weitere Informationen hierzu können in Abschnitt 3.2 und die Rolle des Multithreading kann in dem Abschnitt 3.1.4 nachgelesen werden.

Bei der Verteilung der Rechen- und Speicherkapazität sind alle Ebenen gleichberechtigt. Dennoch gilt diese Gleichberechtigung nicht für den Zugriff auf die Aktoren. Wie in

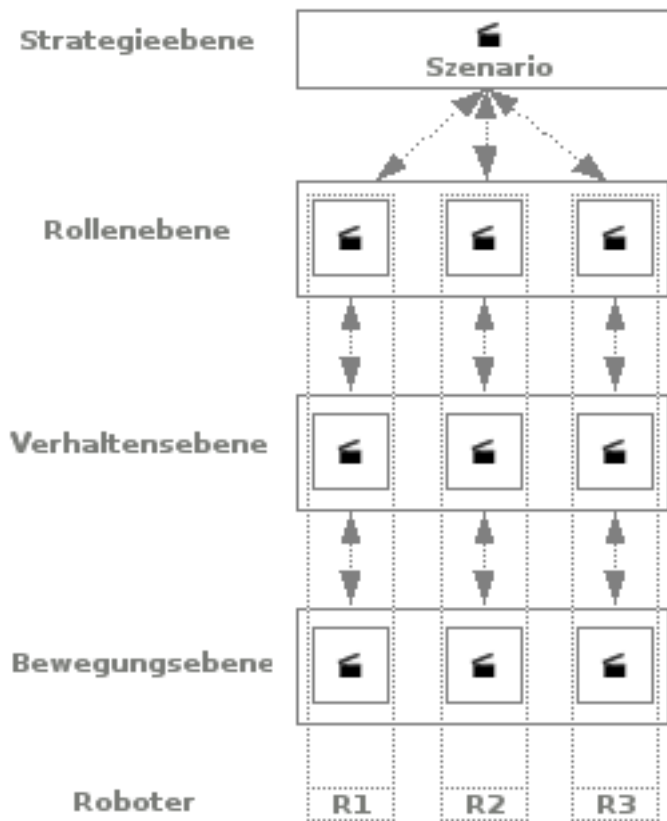


Abbildung 3.2: Eine Darstellung der Planungsarchitektur CSBP.

Abschnitt 2.3 dargestellt, ist es sinnvoll diesen Zugriff nur einem einzigen Modul zuzuweisen. Dieses eine Modul ist das primitivste der Hierarchie, siehe hierzu den Zugriff auf die physikalischen Aktoren auf Seite 24 oder die AuRA Architektur von Arkin auf Seite 25. Diesen Ansatz verfolgt auch die CSBP Architektur. Sie erlaubt nur der untersten Ebene die direkte Steuerung der Aktoren und stellt den Lesezugriff allen Ebenen frei. Die Ebenen sind miteinander in einer Kette verbunden. Jede Ebene kennt seinen Nachfolger und seinen Vorgänger mit dem sie direkt Informationen austauschen darf. Weiterhin hat eine Ebene der Stufe  $i$  eine beschränkte Kontrolle über die Ebene der Stufe  $i - 1$ . Eine detaillierte Beschreibung der Kommunikation sowie der indirekten Steuerung von Ebenen wird im Abschnitt 3.3 erläutert.

### 3.1.1 Das Szenario

#### Definition

Ein Szenario ist ein Programm, das zur Lösung einer Aufgabe dient. Dieses Programm startet den Lösungsvorgang und ist erst beendet, wenn die Aufgabe abgeschlossen ist. Das Beenden einer Aufgabe kann in unterschiedlichen Zuständen erfolgen wie z.B. er-

folgreich, erfolglos, gehindert von dem Ereignis X, etc.

### Beispiel

Ein Roboter hat die Aufgabe einen Ball in ein dafür vorhergesehenes Tor zu dribbeln. Sein Weg von der Startposition bis zum Tor ist von unbekannter Länge. Weiterhin können ihm Hindernisse den Weg zum Ziel versperren. Um den Schwierigkeitsgrad zu minimieren, wird davon ausgegangen, dass die Hindernisse unbeweglich sind. Die Aufgabe ist dann erfüllt, wenn der Roboter es geschafft hat, den Ball innerhalb der konvexen Hülle des Tores zu bringen.

Die folgende Funktion  $f$  ist so definiert, dass sie alle Unterprobleme der Aufgabe erfüllt. Zusätzlich initiiert sie die Aufgabe und terminiert erst, wenn sie vollständig erfüllt ist. Aus diesem Grund kann die Funktion  $f$  ein Szenario genannt werden.

```
f :
    while ( ball is not inside the goal ) do
        if ( ball is blocked ) do
            release ball
        else
            push the ball in goal direction
```

Im Gegensatz zu der oben genannten Funktion wird die folgende Funktion  $g$ , obwohl sie alle Unterprobleme behandelt, nicht nach einem einmaligen Aufruf die vollständige Aufgabe erfüllen. Deshalb kann sie nicht als ein Szenario bezeichnet werden.

```
g :
    if ( ball is inside the goal ) do
        return
    if ( ball is blocked )
        release ball
    else
        push the ball in goal direction
```

Es besteht die Möglichkeit durch den mehrfachen Aufruf der Funktion  $g$  ein Szenario zu erzeugen, das äquivalent zu dem Szenario  $f$  ist. Siehe hierfür die Beispielfunktion  $h$ :

```
h :
    while ( ball is not inside the goal ) do function g
```

### 3.1.2 Transparente Szenarien

Ein Szenario ist transparent, wenn es in sogenannten *Unterszenarien* strukturiert werden kann. Ein Unterszenario stellt ein Szenario dar, das einen Teil der zu erfüllenden Aufgabe umsetzt. Es startet die Ausführung einer Unteraufgabe und terminiert wenn dieses vollständig beendet ist.

Es muss beachtet werden, dass die Gesamtheit aller Unterszenarien  $f_i$  eines Szenarios  $f$  dieselbe Aufgabe erfüllt, wie  $f$  selber. Siehe hierzu die Gleichung (3.1).

$$f = f_1 \circ f_2 \circ f_3 \dots f_n \quad (3.1)$$

Die Voraussetzung für eine Aufteilung des Szenarios  $f$  ist, dass die von ihr zu erfüllende Aufgabe  $A$  modularisierbar ist. Diese Aussage impliziert die Gleichung (3.2). Wenn die Aufgabe nicht modularisierbar ist, so wird sie und das dazu gehörige Szenario als *primitiv* bezeichnet.

$$A = A_1 \circ A_2 \circ A_3 \dots A_n \quad (3.2)$$

Jede Unteraufgabe  $A_i$  dient als Grundlage für die Entwicklung eines Unterszenarios  $f_i$ . Die Gesamtheit der entstandenen Unterszenarien dienen zur Lösung der Gesamtaufgabe  $A$  und beschreiben das Szenario  $f$ , siehe Gleichung 3.3. In einem solchen Fall wird  $f$  als transparent bezeichnet.

$$A_1 \rightarrow f_1, A_2 \rightarrow f_2, A_3 \rightarrow f_3, \dots, A_n \rightarrow f_n \Leftrightarrow A \rightarrow f \quad (3.3)$$

Um ein besseres Bild über die Transparenz zu bekommen, wird das von Seite 32 bekannte Szenario verwendet. Da das Szenario  $f$  nach der aktuellen Definition nicht als transparent bezeichnet werden kann, wird sie in die folgende transparente Funktion  $f_t$  umgeformt:

$f_t$  :

```

while ( ball is not inside goal ) do
    while ( ball is not free ) do
        release ball
    while ( ball is not blocked ) do
        push the ball in goal direction

```

Zu beachten ist, dass bei der Umformung, die Aufgabe im Hintergrund des Szenarios erhalten bleibt.

### 3.1.3 Probleme mit Szenarien

Durch die Arbeit mit den Szenarien sind zwei Schwachstellen des Konzeptes entdeckt worden. Zum einen sind Schwingungen durch das Umschalten zwischen den Szenarien bzw. Unterszenarien erzeugt worden. Zum anderen sind Inkonsistenzen bei den Fallunterscheidungen innerhalb eines Szenarios festgestellt worden. Beide Schwachstellen werden in diesem Abschnitt näher erläutert.

#### Schwingungen

Ein Schaltmechanismus wird durch die Modularisierung des Szenarios bzw. der Vereinigung mehrerer Szenarien zu einem komplexen System benötigt. Dieser Mechanismus realisiert die Verbindung zwischen den einzelnen Modulen indem er zwischen den beteiligten Szenarien hin und her schaltet. Abhängig vom Auftreten einer Situation wird das dazu passende Szenario aktiviert. Je nachdem wie die Realisierung der Schaltung und

der Szenarien zuvor erfolgt ist, kann es dabei zu Problemen kommen. Diese Probleme entstehen dadurch, dass zwei Szenarien sich gegenseitig in der Bearbeitung ihrer Aufgabe behindern. Ein solcher gegenseitiger Ausschluss zweier Szenarien wird als Schwingung bezeichnet.

Geringe Schwingungen verlängern die Abarbeitungszeit des Szenarios und damit auch die Zeit, in der ein Roboter seine Aufgabe erfüllt. Diese Schwingungen können mit der Zeit nachlassen, so dass das System zum Normalzustand zurückkehren kann. Im schlechtesten Fall kann eine Schwingung zu einem Deadlock zwischen den Szenarien führen. Eine solche Situation macht die Erfüllung der Aufgabe nicht mehr möglich.

**Beispiel** In dem Beispiel auf der Seite 32 wurde ein Szenario beschrieben, bei der ein Ball in ein Tor gedribbelt werden sollte. Um eine Schwingung darzustellen, ist eine Veränderung der alten Aufgabenstellung vorgenommen worden. Nun soll der Ball durch einen Schuss in das Tor befördert werden. Um diese Aufgabe zu bewältigen, stehen dem Roboter zwei Schussapparate zur Verfügung: einer Links und einer Rechts. Damit die Schussbewegung erfolgreich ausgeführt werden kann, bedarf es einer perfekten Positionierung von Ball und Schussapparat. Im folgenden Szenario  $s$  ist die Aufgabe des Roboters die passende Schussposition einzunehmen.

$s$  :

```
while ( ball is not in shot position ) do
  while( abs ( ball position - Left shot position ) < c ) do
    position for left shot
  while( abs ( ball position - Right shot position ) < c ) do
    position for right shot
```

Abhängig von dem Wert der Konstante  $c$  tritt durch das Szenario eine Schwingung auf. Der Schwingungseffekt besteht darin, dass der Roboter zwischen der linken und rechten Schussposition hin und her schwankt ohne einen festen Entschluss fassen zu können. Im ungünstigsten Fall entsteht ein Deadlock und die Schussbewegung wird niemals ausgeführt.

### Inkonsistenz

Während der Erfüllung einer nicht primitiven Aufgabe nimmt das System unterschiedliche Zustände an und ihre Gesamtheit bildet einen Zustandsraum. Um die Aufgabe erfolgreich zu erfüllen, muss das Szenario eine Entscheidung für alle Zustände aus diesem Raum bereithalten. Im entgegengesetzten Fall wird das Szenario als inkonsistent bezeichnet.

**Beispiel** Ein Roboter hat die Aufgabe sich zu einem Ball hinzubewegen. Folgendes Szenario kann hierfür definiert werden:

$f$  :

```
while ( ball not reached ) do
```

```
go to ball
```

Dieses Szenario ist inkonsistent, da kein Zustand behandelt wird, in dem der Ball verloren geht. Eine korrekte Variante ist:

f':

```
while ( can see the ball AND ball not reached ) do
  go to ball
```

Die Größe des Zustandsraumes ist eng mit der Komplexität der Aufgabe verbunden, die wiederum stark von den Eigenschaften der Welt abhängig ist. Beispielsweise hat ein Roboter viel mehr Schwierigkeiten damit im Sonnenlicht Fußball zu spielen, als in einer geschlossenen Halle mit konstanten Lichtverhältnissen.

In einem großen System sind inkonsistente Szenarien schwer zu identifizieren. Der Grund hierfür ist, dass einige Zustände nur selten angenommen werden und so Fehler lange Zeit verborgen bleiben können. Durch das Auftreten eines solchen Fehlers kann der Folgezustand nicht mehr vorhersehbar sein, sogar die Beschädigung der Hardware könnte verursacht werden. Es ist ebenfalls möglich, dass der Zustand beibehalten wird und ein Deadlock entsteht.

### 3.1.4 Die Nebenläufigkeit von Szenarien

Die heutigen Roboter haben eine so komplexe Planung, dass sie sogar mehrere Aufgaben gleichzeitig ausführen können. Meistens werden die Aufgaben sequentiell abgearbeitet, einige Robotersysteme erlauben aber auch eine parallele Ausführung. Die Realisierung der parallelen Ausführung sollte dabei nicht notwendigerweise mit dem Einsatz eines Mehrprozessorsystems zusammenhängen, bei der jeder Prozessor eine Aufgabe zugeteilt kriegt. Die Planungsarchitektur kann so konzipiert sein, dass eine parallele Abarbeitung nur auf einem Prozessor möglich ist.

Die CSBP Architektur definiert eine Menge von hierarchisch strukturierten Ebenen, die in einem Multithreadsystem abgearbeitet werden. Die Hierarchie setzt sich aus einer steigenden Abstraktion und einer gleichzeitig sinkenden Hardwarenähe zusammen, siehe Abbildung 3.1. Auf jede Ebene der Hierarchie sammeln sich eine Menge von Szenarien, die alle unterschiedliche Fähigkeiten des Roboters beschreiben. Die Szenarien einer Ebene haben einen gemeinsamen Abstraktionsgrad und eine gemeinsame Hardwarenähe. Auf dieser Basis wird eine minimale Spezifikation für eine Ebene vorgegeben. Die minimale Spezifikation beschreibt, die zu implementierenden Schnittstellen, die eine Verbindung zu den unteren bzw. oberen Ebenen herstellt. Außerdem gibt sie Auskunft über Ereignisse, die auf dieser Ebene behandelt werden müssen.

Abhängig von den zur Verfügung stehenden Ressourcen und den Aufgabenstellungen, wird jeder Ebene mindestens ein Thread zugeordnet. Für den Großteil der Zeit werden gleichzeitig mehrere Szenarien aktiv. Der Grund hierfür liegt in der Verwendung des Multithreadsystems und der Verteilung der Threads auf alle Ebenen der Hierarchie. Diese Szenarien decken gleichmäßig alle Hierarchiestufen ab und achten auf Veränderungen im Zustand des Roboters und seiner Umwelt. Tritt eine Veränderung auf, so wird diese In-

formation sofort an die zuständige höhere Ebene weitergeleitet. Das aktive Szenario auf dieser Ebene entscheidet über die Reaktion auf die Veränderung und leitet sie sofort ein.

## 3.2 Die Ebenen der FUMANOIDS Planung

Die Planung der FUMANOIDS ist auf den vier Ebenen Bewegung, Verhalten, Rolle und Strategie aufgebaut, wobei der Abstraktionsgrad in dieser Reihenfolge abnimmt. Jedem dieser Ebenen wurde genau ein Thread zugewiesen, so dass gleichzeitig ein Szenario auf einer Ebene aktiv sein kann. Die Schnittstellen zwischen den Ebenen, Bewegung-Verhalten und Verhalten-Rolle wurden so konzipiert, dass ein Szenario der unteren Ebene ein Feedback an die obere senden muss. Einzelheiten hierzu können in Abschnitt 3.3 nachgelesen werden.

### 3.2.1 Die Bewegung

Die Bewegung repräsentiert die unterste Ebene in der Planungsarchitektur. Sie ist dadurch das hardwarenächste und am wenigsten abstrakte Modul. Weiterhin hat die Bewegung einen direkten Zugriff auf alle Aktoren und begrenzt auf einige Sensoren des Roboters. Die Szenarien dieser Ebene manipulieren die Werte der Aktoren und hören die Daten der Sensoren ab. Es wird zwischen den statischen und den dynamischen Bewegungen unterschieden, wobei ein Szenario entweder die eine oder die andere Art der Bewegung unterstützt.

#### Die statische Bewegung

Eine statische Bewegung ist eine konstante Matrix  $M$  von Motorpositionen, die in festen Zeitschritten zu den Servomotoren geschickt werden.

$$M = \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \dots & p_{1,n} & t_1 \\ p_{2,1} & p_{2,2} & p_{2,3} & \dots & p_{2,n} & t_2 \\ p_{3,1} & p_{3,2} & p_{3,3} & \dots & p_{3,n} & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{m,1} & p_{m,2} & p_{m,3} & \dots & p_{m,n} & t_m \end{pmatrix} \quad (3.4)$$

Dabei ist  $n$  die Anzahl der Motoren,  $m$  die Anzahl der Bewegungsschritte,  $p_{i,j}$  die Motorposition des Motors  $j$  in dem Bewegungsschritt  $i$  und  $t_i$  die Zeit um den Bewegungsschritt  $i$  auszuführen. Es gilt  $i = 1 \dots m$  und  $j = 1 \dots n$ .

Die Werte in der Bewegungsmatrix werden durch Handarbeit aufgenommen, dies geschieht anhand des im FUMANOIDS Projekt entwickelten *Motionprogramms*. Um einen Bewegungsschritt aufzunehmen, wird der Roboter in die gewünschte Position bewegt und danach die Motorpositionen von dem Motionprogramm ausgelesen. Zu beachten ist hierbei die Belastung der Motoren und das Gleichgewicht der Lage. Das Motionprogramm ermöglicht erstmal die Aufnahme, dann das Abspielen, das Manipulieren, das Speichern

und anschließend das Laden der Bewegungen.

Die Ausführung einer solchen Bewegung besitzt höchste Priorität in der Abarbeitung. Wird die Ausführung abgebrochen so bewirkt dies, dass der Roboter in instabile Positionen kommt und führt möglicherweise zu einem Sturz oder zur Schädigung der Hardware. Ein Sprung innerhalb der Matrix ist nicht möglich, da so eine Instabilität der Bewegung verursacht werden kann.

Der Vorteil einer statischen Bewegung liegt in einer schnellen Entwicklungszeit weil keine besondere Logik erforderlich ist. Im Gegensatz dazu liegt der Nachteil darin, dass auf die Änderungen der Umwelt oder des inneren Zustandes keine Reaktionen möglich sind.

Diese Art von Bewegung wird als Bestandteil eines Szenarios umgesetzt, wobei ihre statische Eigenschaft dem Szenario nicht vollständig übermittelt wird. Das Szenario hat die Möglichkeit außerhalb der Komponente dynamisch auf innere und äußere Veränderungen zu reagieren.

### Die dynamische Bewegung

Bei der dynamischen Bewegung kommt im Gegensatz zu der statischen Bewegung keine konstante Matrix zum Einsatz. Hier wird ein Vektor  $\vec{v}$  verwendet, dessen Wert sequenziell neu berechnet wird. Auf die Berechnung haben äußere und innere Wahrnehmungen einen Einfluss, dadurch kann die dynamische Bewegung dem Szenario gleichgesetzt werden.

$$\vec{v} = ( p_1, p_2, p_3, \dots, p_n, t ) \quad (3.5)$$

$$p_i = \text{normalize}(c_0 p_\alpha + c_1 p_\beta + c_2 S_\gamma + c_3 S_\lambda), i = 1 \dots n$$

$p_i$  stellt die neue Position für den  $i$ -ten Motor dar,  $p_\alpha$  ist seine alte Position und  $p_\beta$  repräsentiert die erwünschte Motorposition. Weiterhin stellen  $S_\gamma$  die Sensorwerte, die den Zustand des Roboters zeigen und  $S_\lambda$  die Sensorwerte, die den Zustand der Umwelt zeigen, dar. Alle Werte fließen parametrisiert, mittels den Konstanten  $c_0$  bis  $c_3$  in die Berechnung mit ein, wonach eine Normalisierung auf den Motorwertbereich stattfindet. Die dynamische Bewegung reagiert mit einer passenden Korrektur auf äußere und innere Zustandsänderungen. Beispielsweise ist das Laufen bei den Robotern der FUmoids als dynamische Bewegung realisiert worden. Wird der Roboter während der Laufbewegung geschubst, so nimmt er die Instabilität wahr und korrigiert seine Bewegung um so erneut eine stabile Lage zu erreichen. Im Gegensatz hierzu würde der Roboter bei einer statischen Implementierung stürzen.

Ein Beispiel für eine innere Zustandsänderung ist der Befehl einer höheren Ebene die Laufgeschwindigkeit zu verdoppeln. Bereits in der darauf folgenden Sequenz können die veränderten Motorpositionen für die neue Geschwindigkeit an die Motoren gesendet werden. Bei einer statischen Bewegung muss das System die vollständige Abarbeitung der Matrix abwarten. Die Dauer beträgt mindestens  $t = t_1 + t_2 + \dots + t_m$ , siehe Gleichung (3.4).

Der große Vorteil einer dynamischen Bewegung liegt in der schnellen Reaktionszeit sowohl auf innere als auch auf äußere Einflüsse. Der Nachteil besteht in einer langen Entwicklungszeit und dem Bedarf einer komplexen Logik.

### 3.2.2 Das Verhalten

Die Verhaltensebene besitzt einen höheren Abstraktionsgrad als die Bewegungsebene und hat keinen direkten Zugriff auf die Aktoren. Die Manipulierung der Aktoren erfolgt indirekt über die *Bewegungsebene*. Hierfür treffen die Szenarien auf der Verhaltensebene die Entscheidung, wann welches Bewegungsszenario aufgerufen oder abgebrochen wird. Des Weiteren stehen der Verhaltensebene alle Sensordaten zur Verfügung wie z.B. Änderungen in der Umwelt und innere Zustandsänderungen, die die Entscheidungsfindung unterstützen.

Das Verhaltensszenario bleibt aktiv während die von ihr gestarteten Bewegungen ausgeführt werden. Sobald ein Bewegungsszenario beendet ist, sendet es ein Feedback an die Verhaltensebene. Dieses Feedback enthält wertvolle Informationen über den Zustand und den Hintergrund der Terminierung. Das Feedback beeinflusst die Entscheidungen des aktiven Verhaltensszenarios und wird in besonderen Fällen an die nächst höchste Ebene weitergeleitet.

Einige Beispiele für Verhaltensszenarien sind: *Ball finden*, *hinter den Ball gehen* oder *Dribbeln*. In allen aufgezählten Verhaltensszenarien werden die Informationen der Kamera eingesetzt um die Kopf- bzw. Körperbewegung des Roboters auszurichten.

### 3.2.3 Die Rolle

Auf der Rollenebene befinden sich Szenarien, die unterschiedliche taktische Spielrollen beschreiben. Ein Rollenszenario ähnelt einer Spielrolle aus einem echten Fußballspiel ( z.B. Stürmer und Torwart ), an dem der hohe Abstraktionsgrad dieser Ebene erkennbar wird.

Ein Rollenszenario besitzt weder direkten Kontakt zu den Aktoren noch zu den Sensoren des Roboters. Szenarien dieser Ebene haben die Entscheidungsgewalt über die Auswahl, das Starten und das Stoppen der Szenarien auf der Verhaltensebene.

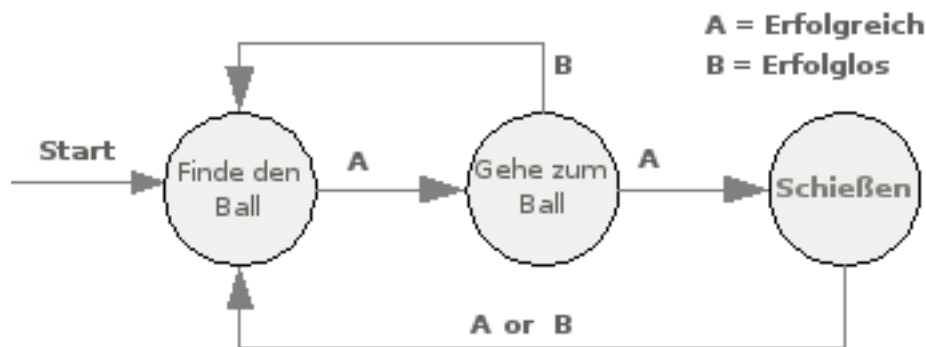


Abbildung 3.3: Der Zustandsautomat einer minimalistischen Stürmerrolle.

Eine Rolle ist intern als Zustandsmaschine definiert, die in einem endlosen Zyklus ausgeführt wird. Ein Beispiel dafür ist in der Abbildung 3.3 zu sehen. Der Automat besitzt

eine ideale logische Abfolge von Verhaltensszenarien. Sie wird dynamisch geändert, sobald ein Verhaltensmuster erfolglos terminiert. Die Änderung der Abfolge ist direkt an dem Feedback der Verhaltensebene gekoppelt.

### 3.2.4 Die Strategie

Die Strategieebene stellt die letzte und höchste Stufe der Hierarchie dar. Hier werden Szenarien eingestuft, die Spielstrategien implementieren. Die Strategieebene bekommt keinen direkten Feedback von den anderen Ebenen, ihr sind jedoch die aktuell aktiven Szenarien auf jeder Ebene bekannt. Anhand dieser Information kann sie den Zustand des Roboters und auch die, der Außenwelt herleiten.

Beispielsweise können die Stürmerrolle, das Verhalten der Schussvorbereitung und die Laufbewegung aktiv sein. Hieraus ergeben sich folgende Informationen:

Die Stürmerrolle	→	Der Roboter wird einen Angriff ausführen, sobald er den Ball gefunden hat.
Das Verhalten der Schussvorbereitung	→	Er befindet sich im Ballbesitz und ist kurz davor einen Schuss auszuführen. Die eigenen Spieler dürfen ihn auf keinen Fall dabei behindern.
Die Laufbewegung	→	Er hat noch keine Schussposition erreicht.

Anhand der Netzwerkverbindung kann jeder spielende Roboter, Informationen über seine aktuell laufenden Szenarien übertragen. Diese Informationen werden in den Strategieszzenarien verwertet, um dem Roboter eine sinnvolle Rolle im Spiel zuzuweisen. Im ganzen Spielverlauf wird die Rollenzuweisung dynamisch von dem Strategieszzenario geregelt.

**Beispiel** Der Roboter A und der Roboter B werden für ein Spiel gestartet. Beide erhalten von dem Strategieszzenario dieselbe Initialrolle: *Stürmer*. Nach kurzer Zeit erreicht Roboter A den Ball. Das Strategieszzenario von Roboter B bemerkt es und wechselt von der Stürmerrolle auf die Torwartrolle. Auf diese Weise kann Roboter A ungestört nach vorne stürmen, während Roboter B das Tor verteidigt.

## 3.3 Kommunikation und Steuerung

Innerhalb des Planungssystems können drei Kommunikationswege identifiziert werden, die sich anhand der transportierten Information voneinander unterscheiden. Es handelt sich hierbei, um den *Kontrollpfad*, den *Statuspfad* und den *Identifikationspfad*. Sie sind in der Abbildung 3.4 dargestellt. Im Folgenden werden alle drei Pfade sowie ihr Beitrag zu der Steuerung des Systems beschrieben.

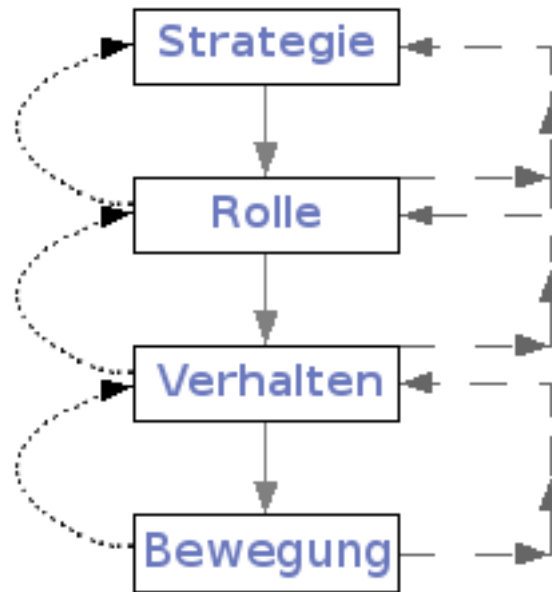


Abbildung 3.4: Darstellung der Kommunikationswege: Kontrollpfad (durchgezogene Linie), Statuspfad (gepunktete Linie), Identifikationspfad (gestrichelte Linie) in der Planungsarchitektur der FUsmanoids.

### 3.3.1 Der Kontrollpfad

Der Kontrollpfad kann auch als Steuerpfad bezeichnet werden, da hier die Steuerbefehle transportiert werden. Die Steuerung der Planung teilt sich dabei auf die einzelnen Hierarchieebenen auf, wodurch jede Ebene seinen Teil zur Steuerung beiträgt.

Das System wird in der höchsten Ebene durch den Start eines Strategieszenarios angestoßen. Dieses Szenario wird solange innerhalb einer Endlosschleife ausgeführt bis ein Abbruchbefehl durch menschliches Handeln eintritt. Es besitzt ebenfalls über die Rollenebene eine Start und Stopp Kontrolle sowie die Entscheidungsmacht, welches Rollenszenario zum Einsatz kommt. Über die tiefer liegenden Ebenen besitzt die Strategie keine direkte Macht. Dennoch kann sie indirekt durch die Wahl einer bestimmten Rolle, die Menge der dort einsetzbaren Szenarien einschränken.

Angenommen, das Strategieszenario entscheidet sich dafür, die Stürmerrolle zu aktivieren, so ist das Verhaltensszenario *Verteidige das Tor* und die Bewegung *Hinwerfen auf die Torlinie* ausgeschlossen worden. Sie sind erst bei der Aktivierung der Torwartrolle relevant. Analog verhält es sich bei den restlichen Ebenen. Die Rollenebene kontrolliert die Verhaltenebene und diese wiederum die Bewegungsebene.

Die Realisierung erfolgt anhand von sogenannten *Switch-Methoden*, die auf der Basis von Threads arbeiten. Die Methode bekommt einen Pointer auf das neue Szenario übergeben und startet den Thread mit diesem Modul neu. Jede Ebene bekommt ihre eigene Switch-Methode zugewiesen. So ergeben sich für den Austausch eines Szenarios der jeweiligen Ebene, die analog implementierten Methoden *switchMotion*, *switchBehavior*, *switchRole*

und *switchStrategy*.

Die Switch-Methode der Ebene  $X$  darf nur von der Ebene  $X + 1$  aufgerufen werden. Die *switchStrategy* Methode wird von der Benutzerschnittstelle aufgerufen und ermöglicht den oben erwähnten Anstoß des Systems sowie die schnelle Änderung der Strategie.

### 3.3.2 Der Statuspfad

Auf dem Statuspfad werden Informationen über den aktuellen Status eines Szenarios gesendet. Der Datenfluss startet in der untersten Ebene und ist dabei nach oben gerichtet. Von dieser Ebene aus werden *Statusinformationen* an das aktive Verhaltensszenario geschickt, das wiederum einen Status an die Rollenebene sendet. Der Pfad führt weiter bis zur Strategieebene, jedoch wird die letzte Verbindung zwischen Rolle und Strategie von den FUMANoids aktuell nicht genutzt. Mögliche Anwendungen werden in der Zusammenfassung auf Seite 77 diskutiert.

Ähnlich dem Kontrollpfad erreichen die Informationen nur die direkt benachbarte Ebene und besitzen einen indirekten Einfluss auf die restlichen Hierarchieebenen. Beispielsweise kann die Bewegungsebene die Statusinformation *Roboter ist gestürzt* an das Verhaltensszenario senden. Dieser erkennt dann, dass durch das Hinfallen des Roboters die Erfüllung seiner Aufgabe nicht mehr möglich ist. Das Verhaltensszenario reicht das Problem an die höhere Ebene weiter und stoppt mit derselben Statusinformation. Die Rollenebene besitzt den nötigen Abstraktionsgrad zur Lösung des Problems und startet das neue Verhaltensszenario *Aufstehen*.

Zur Umsetzung des Statuspfades ist eine vordefinierte Menge von Konstanten erforderlich, die als Statusinformationen gesendet werden dürfen. Es gibt folgende grundlegende Statusinformationene: *Aktiv*, *Erfolgreich*, *Erfolglos* und *Gestürzt*. Jede Ebene besitzt eine eigene globale *Statusvariable*, in die die Statusinformation des Szenarios geschrieben wird. Diese Variable wird beim Start eines neuen Szenarios mit dem Wert *Aktiv* initialisiert. Umgesetzt wird dies durch die Erweiterung der bereits bekannten *Switch-Methode*. Theoretisch kann nun global auf jede Statusvariable zugegriffen werden, praktisch umgesetzt greifen aber nur die berechtigten Ebenen auf die Variable zu.

### 3.3.3 Der Identifikationspfad

Die Statusinformationen reichen alleine für die Planung der höheren Ebenen nicht aus. Es ist erforderlich, dass eine Ebene das aktuell laufende Szenario aller unteren Ebenen genau identifizieren kann. Für diesen Zweck wird der Identifikationspfad genutzt.

Jedes Szenario besitzt eine eindeutige ID, die eine gleichzeitige Zuordnung zu einer Ebene herstellt. Diese ID wird durch den Start des Szenarios allen übergeordneten Ebenen gesendet und für die Dauer der Abarbeitung sichtbar gemacht.

Angenommen der Roboter führt eine Laufbewegung aus während die Rolle einen Wechsel der Verhaltensszenarien veranlasst. Weiterhin sei für das neue Szenario eine Laufbewegung erforderlich, so ist der Abbruch der Bewegung überflüssig. Dank der ID-Information vermag das neue Verhalten die aktuelle Bewegung als die Laufbewegung zu identifizieren. So wird ein Stopp und ein Neustart verhindert, der bei schneller Durchführung Instabi-

lität verursacht.

Wie in der Abbildung 3.4 zu sehen ist, ist der Identifikationspfad ausschließlich nach oben gerichtet. Die Verwertung der ID-Informationen gewinnt gleichzeitig mit dem Anstieg der Ebenen an Abstraktion. Die Strategieebene verwendet die ID-Daten für die globale Planung während auf der Verhaltensebene lokale Entscheidungen getroffen werden.

Der Identifikationspfad ist bei der Robotergeneration 2008 mittels eines *Enum* und einer entsprechenden globalen Variable für jede Ebene umgesetzt worden. Jedes Szenario besitzt eine eindeutig identifizierbare Konstante, die es in dem *Enum* seiner Ebene repräsentiert. Diese Konstante wird beim Start des Szenarios, der globalen Variable der Ebene zugewiesen.

Aufgrund der vorteilhaften Eigenschaften von C++ kann bei der Robotergeneration 2009 eine bessere Lösung gefunden werden [23]. Hier wird innerhalb des Konstruktors von jedem Szenario eine eigene ID festgelegt, die durch die Klassenmethode *getID()* ausgelesen werden kann.

Die ID-Informationen werden nicht nur lokal genutzt sondern auch über das Netzwerk an alle anderen Roboter übertragen. So kennt jeder Roboter den Zustand seiner Mitspieler. Das bringt große Vorteile in der Spielplanung mit sich, die im folgenden Abschnitt beschrieben werden. In der Abbildung 3.5 können die gesendeten Signale in einer zeitlichen Abfolge auf dem Identifikationspfad beobachtet werden.

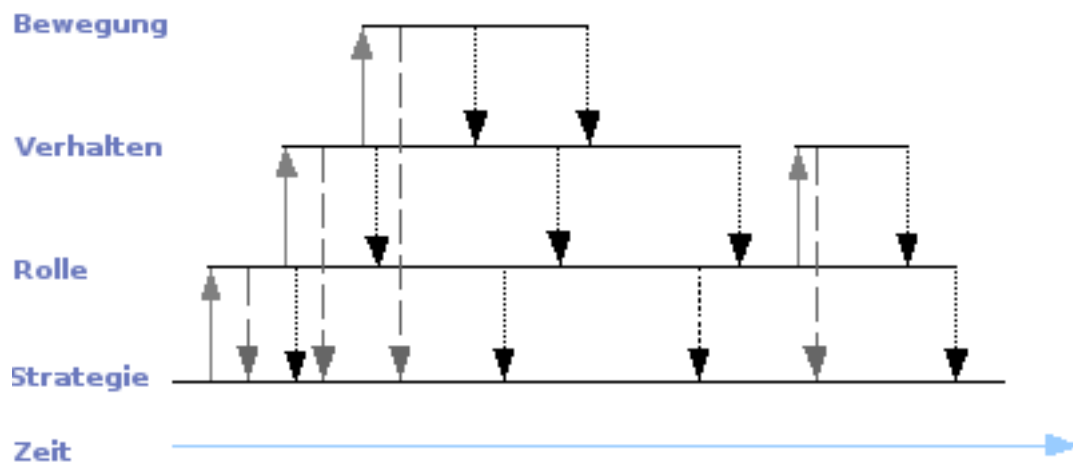


Abbildung 3.5: Ein zeitbedingtes Funktionsdiagramm der Planungsarchitektur der FURMANOIDs. In der Abbildung sind die durchgezogenen Linien Kontrollsignale, die gepunkteten Statussignale und die gestrichelten ID-Signale.

### 3.4 Die kooperativen Aspekte

Die CSBP Architektur eignet sich durch ihre Eigenschaften und Struktur gut für den Einsatz bei Multiagentensystemen. Auf jeden Roboter ist dasselbe Programm gespielt,

wobei alle selbstständig Entscheidungen treffen, die das Team unterstützen. Dabei besteht die Besonderheit in der Verteilung der Entscheidungsgewalt auf unterschiedliche Abstraktionsstufen. Die Strategieebene ist die höchste Abstraktionsebene und für die Entscheidungen im Bereich des Teamspiels zuständig.

Wie im Abschnitt 3.3.3 bereits beschrieben, kennt diese Ebene die aktiven Szenarien aller Agenten und zwar zu jedem Zeitpunkt. Die Übertragung der ID-Informationen von einem Roboter zum Anderen geschieht mittels eines Datenpaketes. Die Daten werden über WLAN mittels Broadcast an das eigene Team gesendet.

Diese Verlängerung des Identifikationspfades zu anderen Robotern wird als der *virtuelle Identifikationspfad* bezeichnet. Die, über diesen Pfad, gesendeten Daten sind ebenfalls *virtuell*. In dem Datenpaket befinden sich weitere für die Planung nützliche Informationen:

- Die ID des Roboters,
- Die globale und relative Position des Balles,
- Die globale Position des Roboters auf dem Feld,
- Die Farbe des gegnerischen Tors, die Anstoßinformationen, die Farbe des eigenen Teams, etc.

Mit Hilfe dieses Datenpaketes können ausreichend Informationen über den Zustand, die Position und auch die Priorität der spielenden Roboter gewonnen werden. Das Strategieszenario hat einen globalen Blick auf die aktuelle Spielsituation und kann mit einer passenden Rollenverteilung reagieren. Die Abbildung 3.6 verdeutlicht den Zusammenhang und die Funktionsweise mehrerer Agenten während des Spielverlaufes.

Es können diverse Kombinationen von stark offensiven zu stark defensiven Spielstrategien entwickelt werden. Je nach Bedarf kann eine neue Strategie eingesetzt werden, wobei das Umschalten nur einen Funktionsaufruf benötigt.

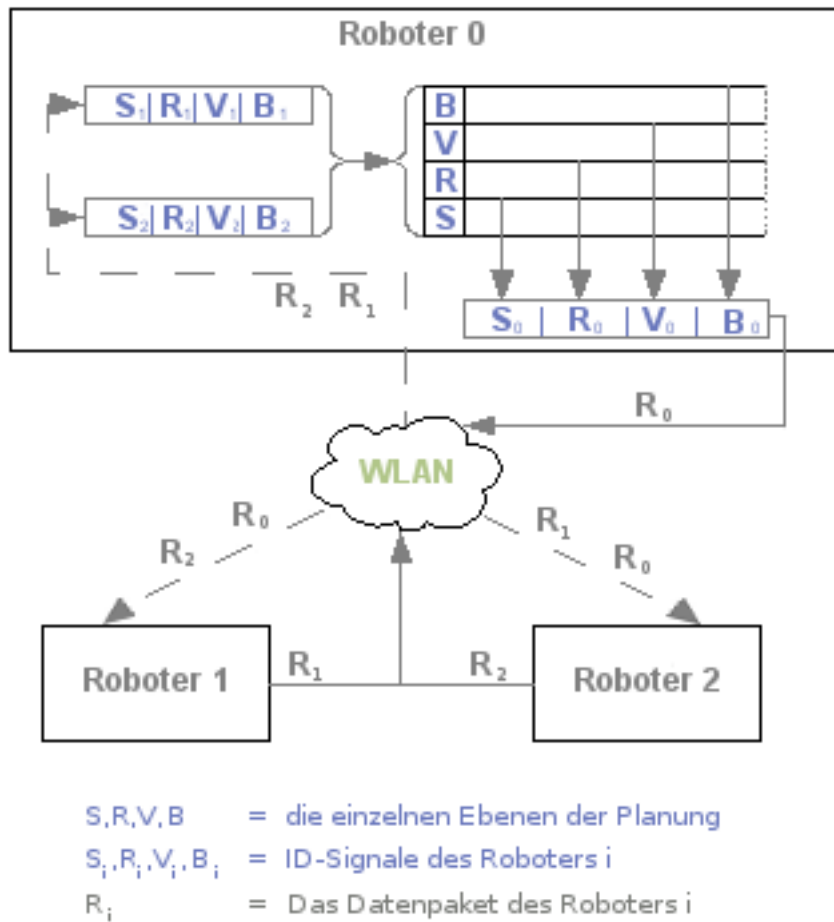


Abbildung 3.6: Die Darstellung des Informationsflusses zwischen drei Robotern und des internen Datenflusses der Planungsarchitektur.

## 4 Die Planung der FUMANOIDS

In diesem Kapitel wird das Planungssystem der FUMANOIDS Roboter vorgestellt. Das System wird aus einer Vielzahl von Szenarien gebildet, die sich über die komplette Planungshierarchie erstrecken. Im Rahmen dieser Diplomarbeit ist die Planung der Roboter- generation 2008 entwickelt worden, welches im Rahmen der Studienarbeit von D. Seifert [23] auf die Nachfolgeneration portiert worden ist. Dabei sind als Teil dieser Arbeit die Inhalte der alten Szenarien, den neuen Ressourcen angepasst worden. Weiterhin sind Optimierungen und auch neue Szenarien entstanden.

Wegen dem großen Umfang der Entwicklungsarbeit sind Teile des Systems im Jahr 2009 als Thema weiterer Abschlussarbeiten vergeben worden. Hierzu gehören z.B. die Entwicklung einer dynamischen Schussbewegung, die vollständige Torwart- und Verteidigerrolle samt Verhalten- und Bewegungsszenarien.

Der Schwerpunkt dieser Arbeit ist die Kooperation zwischen den einzelnen Spielern, die das Zusammenspiel der Roboter als Team ermöglicht. Für diesen Zweck sind unterschiedliche Rollen entwickelt worden wie z.B. *der Stürmer*, *der Flügelspieler* oder *der Unterstützer*. Sie sind als Szenarien der Rollenebene implementiert worden und werden in diesem Kapitel detailliert in ihrer Funktionsweise erklärt.

Damit, die Roboter gemeinsam als Team funktionieren können, müssen die folgenden zwei Bedingungen erfüllt werden:

- Bedingung 1: Der einzelne Roboter muss die Fähigkeit besitzen, jede der oben genannten Rollen zu spielen.
- Bedingung 2: Mehrere Roboter müssen unter Verwendung dieser Rollen miteinander zusammenspielen.

Für die Verwirklichung der ersten Bedingung werden die unteren Hierarchieebenen *Verhalten* und *Bewegung*, passend zu den einzelnen Rollen, aufgebaut. Es ist sinnvoll als erstes die Rolle zu definieren, danach die dafür benötigten Verhaltensmuster und zuletzt die erforderlichen Bewegungen.

Auf diese Weise besitzt der Entwickler eine Spezifikation an der sich die Entwicklung orientieren kann. Sie gewährleistet die Begrenzung der Funktionalität und Abstraktion in einem Szenario, so dass eine Zuordnung in die Hierarchie möglich bleibt.

Die Szenarien, die sich auf den unteren Hierarchieebenen befinden, besitzen einen großen Wiederverwendungswert. Beispielsweise wird für die Laufbewegung des Roboters nur ein Szenario implementiert. Analog verhält es sich auch mit dem Verhaltensszenario *Gehe zum Ball*, das in den meisten Rollenszenarien zum Einsatz kommt.

Für einen guten Überblick über die Lösung zu der ersten Bedingung werden von dem Abschnitt 4.1 bis 4.3 die einzelnen Rollenszenarien vorgestellt. Zu jedem Szenario werden

die noch unbekanntes Verhaltensszenarien beschrieben, die für die Realisierung notwendig sind. Anschließend wird auf die einzelnen Bewegungsszenarien eingegangen.

Bei den FUMANoids ist für die Erfüllung der zweiten Bedingung der *Identifikationspfad*, die *Netzwerkverbindung*, die *Strategie*- und die *Verhaltensebene* von Bedeutung. Der Identifikationspfad beschreibt einen Kommunikationsweg in der Planungsarchitektur durch den alle Ebenen untereinander verbunden sind. Über diesen Pfad werden die ID's, der aktuell laufenden Szenarien, verschickt. Des Weiteren besteht über das Netzwerk eine virtuelle Verbindung zum Identifikationspfad anderer Roboter. Durch diese virtuelle Verbindung können die ID-Informationen anderer Roboter empfangen und die eigenen gesendet werden. Das Datenpaket, das über das Netzwerk gesendet wird, enthält neben den ID-Informationen weitere relevante Daten. Hierzu gehört die relative und absolute Ballposition, die eigene Position, die Teamfarbe etc. siehe Abschnitt 3.4.

Die Verarbeitung der Informationen geschieht bei den FUMANoids zu 25% in den Verhaltensszenarien und zu 75% in der Strategieebene. Beispielsweise verwenden die Verhaltensszenarien *Seitlich Unterstützen* Abschnitt 4.2.1 und *Frontal Unterstützen* Abschnitt 4.3.1 diese Informationen um mit dem Stürmerroboter und untereinander zu kooperieren. Da der Großteil der Auswertung auf der höchsten Hierarchiestufe geschieht, werden in Abschnitt 4.4 einige der realisierten Strategieszenarios detailliert beschrieben und ihre Funktionsweise erklärt.

## 4.1 Der Stürmer

Der Stürmer ist die wichtigste Rolle im offensiven Fußballspiel und ist deshalb bei den FUMANoids als erstes realisiert worden. Das Ziel dieser Rolle ist es schnellstmöglich zum Ball zu gelangen, den sicher in Tornähe zu bringen und ein Tor zu erzielen. Ein Stürmerroboter kümmert sich nur indirekt um die Verteidigung, indem er den Ball in Bewegung hält und ihn in Richtung des gegnerischen Tores führt. Verliert er den Ball, so versucht er ihn wieder zu finden und ihn erneut unter seine Kontrolle zu bringen.

Das Zustandsdiagramm aus der Abbildung 4.1 zeigt die Funktionsweise der Stürmerrolle. Dabei stellen die Zustände, die zur Realisierung notwendigen Verhaltensszenarien, dar. Der *Stürmer* wird mit dem Zustand *Finde den Ball* gestartet, in dem aus einer festen Position nach dem Ball gesucht wird. Ist der Ball gefunden worden, wird dieses Szenario beendet und ein zweites Szenario ausgeführt, um den Roboter in die Ballnähe bzw. hinter den Ball zu bewegen. Nach der Ausführung des Verhaltens *Annäherung zum Ball* befindet sich dieser zwischen Roboter und dem gegnerischen Tor. Hier ist der Roboter zum Tor ausgerichtet und der Ball in seiner direkten Nähe. Bevor das Schussverhalten aktiv wird, kann der Roboter entscheiden ob er sich dem Tor nähert. Fällt die Entscheidung positiv aus, so bewegt der Roboter den Ball in die gewünschte Entfernung. Dafür wird das Szenario *Dribbeln* eingesetzt. Anschließend kann aus der gewünschten Entfernung das Verhaltensszenario *Schießen* aktiviert werden. Nach einer Feinjustierung der Schussposition wird die statische Bewegung, siehe Abschnitt 3.2.1 für den Schuss abgespielt. Sollte der Roboter ein Szenario aus bestimmten Gründen nicht erfolgreich ausführen kön-

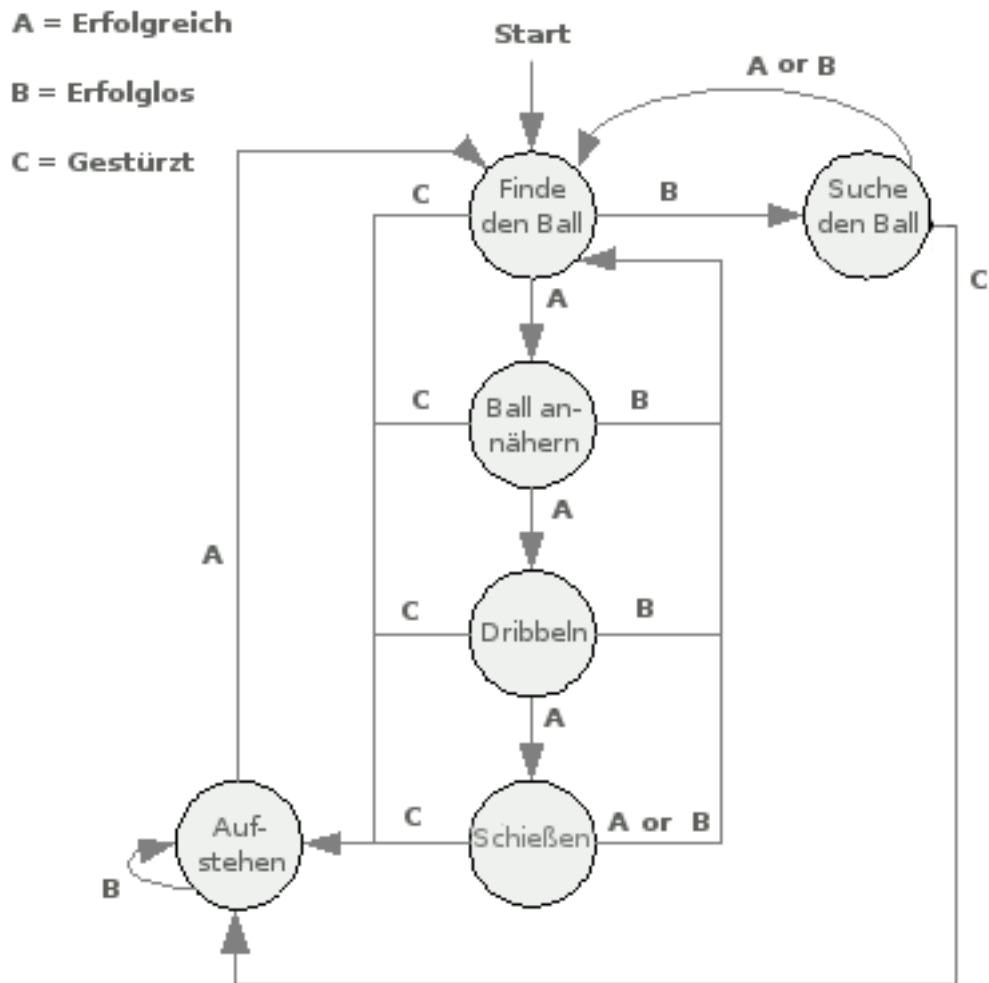


Abbildung 4.1: Das Zustandsdiagramm der Stürmerrolle, bei der die Zustände den Verhaltensszenarien und die Übergänge dessen Feedback entsprechen.

nen, so hat das Rollenszenario entsprechende Reaktionen dafür vorgesehen. Zum Beispiel besteht jederzeit die Gefahr, dass der Roboter stürzt. In diesem Fall wird das Verhaltensszenario sofort abgebrochen und von der Rolle das Verhalten *Aufstehen* gestartet. Um häufige Stürze des Roboters zu vermeiden, ist die Erkennung und das Ausweichen von Hindernissen in die Verhaltensszenarien integriert worden. Auf diese Weise können, die zu Instabilität führenden Kollisionen, mit den eigenen oder gegnerischen Spielern vermieden werden.

Das Rollenszenario wird so lange wiederholt bis das Szenario der höheren Ebene kein Unterbrechungssignal sendet. In den folgenden Abschnitten werden alle Szenarien der Reihe nach erläutert.

### 4.1.1 Das Finden des Balles

Dieses Szenario hat die Aufgabe den Spielball auf dem Feld zu finden. Die Voraussetzung dafür ist, dass der Roboter eine gute Kalibrierung hat und alle wichtigen Farben und Objekte richtig erkannt werden. Zusätzlich müssen bestimmte Eigenschaften der Hardware genau bekannt sein, z.B die Ausrichtung der Kamera, ihr Öffnungswinkel, ihre Auflösung, ihre Bildrate sowie die Freiheitsgrade vom Kopf des Roboters.

Das Szenario startet aus der aktuellen Lage des Roboters mit der Analyse einer Bild-

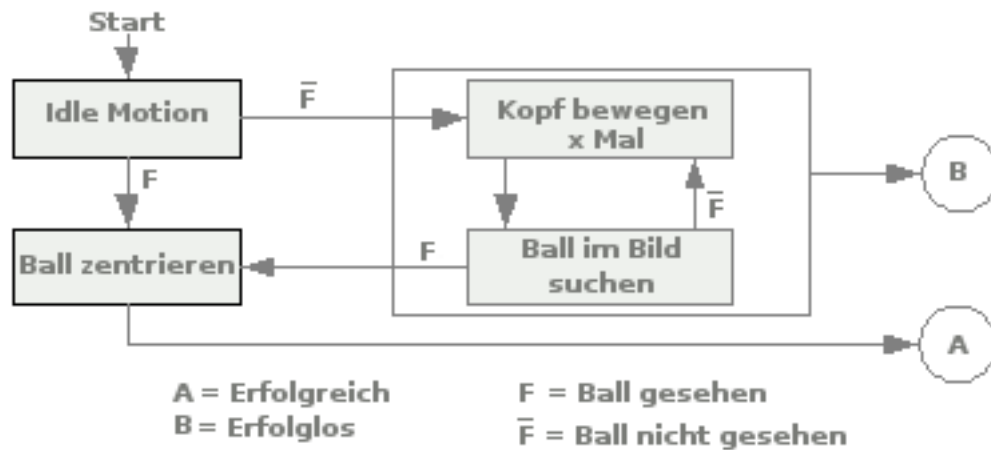


Abbildung 4.2: Das Funktionsdiagramm des Szenarios: Finde den Ball.

aufnahme. Hierfür wird eine von der Vision Komponente, siehe Abschnitt 1.3.1, bereitgestellte Funktion verwendet.

Ist der Ball gefunden worden, so wird er im Sichtfeld des Roboters zentriert wonach das Szenario mit dem Status *Verhalten Erfolgreich* terminiert. Für das Zentrieren wird die Methode `centerBall()` verwendet, die stark von den Freiheitsgraden der Kopfkonstruktion abhängig ist. Die Methode erhält als Eingabe die Position des Balles im Bild der Kamera. Es wird die Differenz zwischen dem Bildzentrum und der Ballkoordinate gebildet. Damit der Ball in das Zentrum des Sichtfeldes gelangt, muss der Kopf um diese Differenz gedreht werden. Hierfür wird der Wert normalisiert und in die Einheit der Drehwinkel übertragen. Für jeden Freiheitsgrad des Kopfes wird ein solcher Winkel berechnet, so dass die Servomotoren gleichzeitig die Zielposition ansteuern können.

Für die Liga Humanoid KidSize ist ein maximaler Drehwinkel der Kopfbewegung festgelegt worden. Wird der Ball hinter diesem Winkel gefunden, so kann er nicht zentriert werden. In diesem Fall bewegt sich der Kopf des Roboters bis zu der maximal erlaubten Position.

Im Fall, dass der Ball nicht im Blickfeld des Roboters liegt, wird ein Scannmechanismus eingeleitet und so möglichst viel von der Umgebung eingefangen. Hierfür ist ein großer Öffnungswinkel der Kamera hilfreich, denn je kleiner der Winkel, um so mehr Kopfbewegungen sind erforderlich. Weiterhin kann im Kopfbereich der Einsatz mehrerer Motoren notwendig sein, um so den Sichtbereich zu vergrößern.

Es ist eine Abfolge von Kopfbewegungen erzeugt worden, die in kürzester Zeit die Umwelt abscannt und gleichzeitig zuverlässige Bilder aufnimmt. Als Maß für die Bewegung ist die Bilddimension verwendet worden. Sie sind als Drehwinkel übersetzt worden und stellen den Wert, der Motorbewegung zwischen zwei Bildaufnahmen, dar. Bei der neuen Robotergeneration ist es durch den großen Öffnungswinkel ausreichend, eine horizontale Bewegung auszuführen. Im Gegensatz dazu ist bei der älteren Generation eine zweidimensionale Bewegung erforderlich gewesen. Sie ähnelt dem Muster eines Rechtecksignals bei dem jede Änderung der Amplitude eine Bildaufnahme bewirkt hat. Auf diese Weise ist ein Sichtwinkel im nahen und fernen Bereich vollständig abgescannt worden. Der Roboter führt die Bewegungsreihenfolge genau einmal im Verlauf des Szenarios aus. Kann der Ball gefunden werden, so wird er in das Zentrum des Sichtfeldes gebracht. Im entgegengesetzten Fall bricht das Szenario mit dem Status *Verhalten Erfolglos* ab. Das laufende Rollenszenario entscheidet darüber, wie weiter verfahren werden soll.

### Die Idle Position

Die Idlebewegung ist die erste Bewegung, die der Roboter ausführt. Es handelt sich hierbei um eine statische Bewegung bei der, jeder Motor genau einmal gedreht wird. Das Ergebnis der Bewegung ist die gerade Haltung des Roboters.

Alle Spieler der FUMANOIDs werden so eingestellt, dass ihre Haltung für den Beobachter gleich erscheint. Für die Einstellung wird eine *Offset Datei* verwendet, in der jedem einzelnen Motor ein konstanter Offset zugewiesen wird. Anhand dieser Offsets werden die Roboter aneinander angeglichen, so dass dieselbe statische Bewegung für alle Roboter nutzbar wird.

**Die Umsetzung der statischen Bewegung:** Um Speicherplatz zu sparen, sind die statischen Bewegungen als, eindimensionale konstante Arrays, in dem Flash gespeichert worden. Dies ist bei der Generation 2008 wegen geringen Ressourcen besonders wichtig gewesen. Um eine einfache Wartung der Arrays zu gewährleisten, ist eine repräsentative Gleichung für die Beschreibung der Länge benutzt worden. Sie definiert die Länge, als die Summe von dem Produkt der Anzahl von Zeilen und Spalten der Bewegungsmatrix und einen Stoppwert. Letzteres dient als ein Signal für das Ende des Arrays im Speicher. Da alle Werte innerhalb der Matrix eine positive Zahl darstellen, ist bei den FUMANOIDs als Stoppwert  $-1$  eingesetzt worden.

Um eine statische Bewegung abzuspielen, verwenden alle Szenarien auf der Bewegungsebene in ihrer Implementierung die Methode *playMotion(staticMotion)*. In dieser Methode wird aus der vorgegebenen Zeit, der aktuellen Motorposition und der Zielposition für jeden einzelnen Motor eine Geschwindigkeit berechnet. Sie beschreibt wie schnell sich der Motor drehen muss um den gewünschten Zielwert pünktlich zu erreichen. Der Servomotor bekommt sowohl die Zielposition als auch die Drehgeschwindigkeit zugewiesen und kann so die gewünschte Bewegung ausführen. Bei dem alten Roboter ist das Array nur für die Dauer der Bewegung im RAM zwischengespeichert worden. So kann auf die Daten zugegriffen werden ohne dauerhaft den RAM zu belasten.

## Die Kopfbewegung

In der älteren Fassung der Implementierung ist für die Kopfbewegung ein eigenes Szenario vorgesehen worden. Dies ist aber nicht immer praktikabel gewesen, weil bei den FUmoids nur ein einziger Thread einer Hierarchieebene zugeteilt ist. In dem Thread ist zu 80% der Fälle das Szenario für die Laufbewegung aktiv. Nur in wenigen Fällen wie z.B. dem, des oft im Tor still stehenden, Torwarts von 2008 ist dieses Szenario von nutzen gewesen.

Es ist also eine Lösung erforderlich gewesen um die gleichzeitige Bewegung der Kopfmotoren und die Laufbewegung des Roboters zu ermöglichen. Für diesen Zweck ist bei der Laufmaschine der älteren Generation die Menge der Parameter ergänzt worden. Jedem Kopfmotor ist ein eigener Parameter zugewiesen worden.

Bei der neuen Generation ist der Kopf des Roboters abstrakt in einer Klasse *Head* repräsentiert worden, so ist sie für die Kopfbewegung verantwortlich [23]. Es gibt eine direkte Verbindung zwischen dieser und einer zweiten Klasse, die das Senden der Daten auf dem seriellen Bus verwaltet. Auf diese Weise können jederzeit die Servomotoren, die den Kopf des Roboters bewegen, angesprochen werden.

### 4.1.2 Die Suche nach dem Ball

In dem Fall, dass der Roboter den Ball nicht finden kann, wird das Verhalten *Suche den Ball* von der Rolle gestartet. Ihr Ziel ist es, den Roboter zu einem Punkt auf dem Feld zu bringen, wo er mit hoher Wahrscheinlichkeit auf den Ball trifft. Während der Positionierung achtet der Roboter kontinuierlich auf das Erscheinen des Balles.

Der Kern des Szenarios besteht darin, dass der Roboter sich zu einem ausgewählten

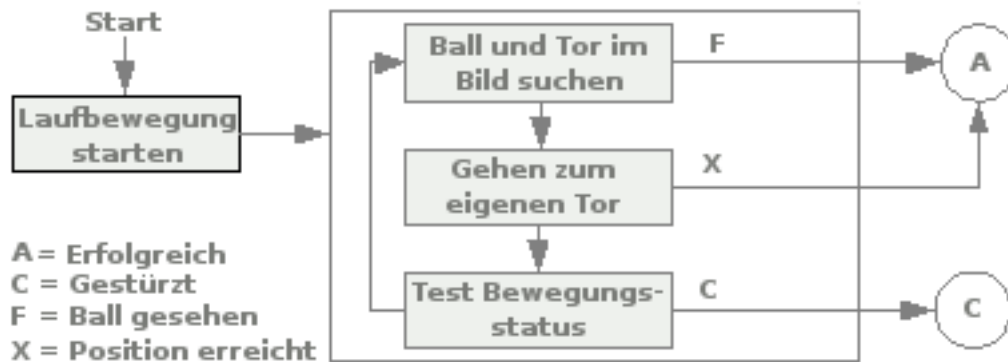


Abbildung 4.3: Das Funktionsdiagramm des Verhaltens: Suche den Ball, ohne der Nutzung globaler Positionsdaten.

Punkt auf dem Spielfeld bewegt. Die Komplexität der Aufgabe variiert abhängig davon, wie viele Daten zu welchem Zeitpunkt von der Umwelt bekannt sind. Zwischen den beiden Robotergenerationen sind diesbezüglich große Unterschiede vorhanden, entsprechend basieren auch die Implementierungen auf unterschiedlichen Ansätzen. Während bei

der älteren Generation nur eine relative Weltanschauung vorliegt, ist bei der Neuen eine globale Sicht vorhanden. Sie ermöglicht eine ganz andere Art der Planung.

Bei dem älteren Robotermodell ist die Zielposition, für das Verhalten *Suche den Ball*, in die Nähe des eigenen Tores gesetzt worden. Diese Position basiert auf der Annahme, dass der Ball von dem gegnerischen Team zum eigenen Tor gebracht wird und dort deswegen Unterstützung erforderlich ist. Diese Roboter haben eine sehr geringe Bildauflösung und auch keine Möglichkeit sich während des Spiels zu lokalisieren [10]. Sie orientieren sich anhand der aktuell sichtbaren Markierungen, dabei ist die Menge der zuverlässigen Markierungen auf die beiden Tore begrenzt. Eine tornahe Position kann somit zuverlässiger erreicht werden als alle anderen Koordinaten auf dem Feld.

Der Roboter benötigt direkte Sicht auf das Tor, um sich zu orientieren. Verliert er diese, so versucht er erst einmal ihn wiederzufinden. Gelingt ihm das nicht, so sucht er nach dem anderen Tor. Wenn keines der Tore gefunden werden kann, so wird der Roboter orientierungslos und spielunfähig.

In der Abbildung 4.3 ist die Funktionsweise des Szenarios der Ballsuche dargestellt. Für das Verständnis des Verfahrens ist die Funktions- und Verwendungsweise des Laufszenarios, welches im Abschnitt 4.1.2 erklärt wird, wichtig. Durch eine vier Mal bessere

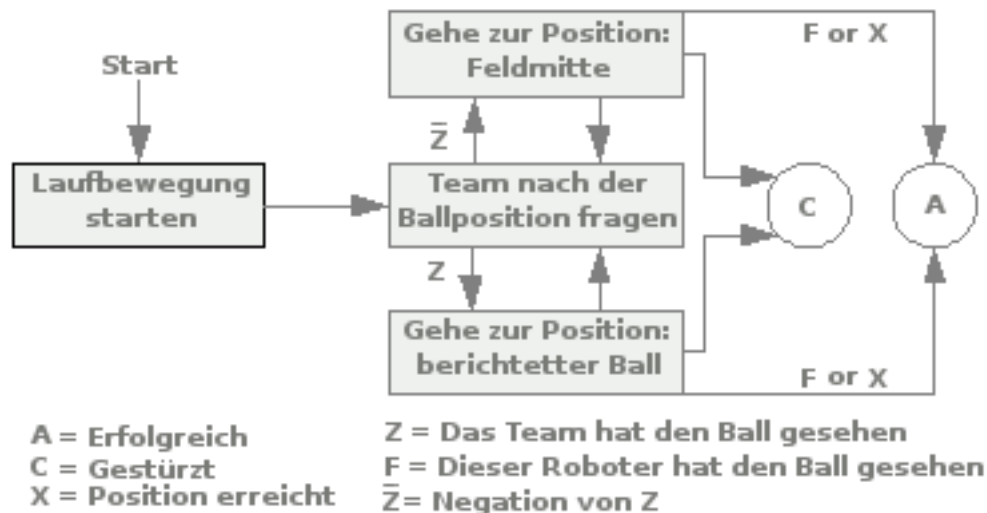


Abbildung 4.4: Das Funktionsdiagramm des Verhaltens: Suche den Ball, mit der Nutzung globaler Positionsdaten.

Auflösung und stärkerer Rechenkapazität, erkennt der Roboter aus der neuen Generation nicht nur entfernte Bälle viel besser, sondern auch die Säulen und Feldlinien. So ist eine Selbstlokalisierung realisiert worden [13]. Sie verleiht dem Roboter die Fähigkeit, neben seiner eigenen globalen Position, auch die aller anderen Objekte, auf dem Feld zu identifizieren. Dank dieser Kenntnis ist es möglich einen Weg, zwischen zwei globalen Koordinaten zu berechnen, auf dem der Roboter entlang laufen kann. Die Entwicklung eines entsprechenden Szenarios *Gehe zur Position (X, Y)* ist Teil der oben genannten Diplomarbeit gewesen.

Anhand der Fähigkeit des Roboters sich auf einem beliebigen Punkt zu positionieren, kann die Entscheidung bei dem Szenario *Suche den Ball* auf das Zentrum des Spielfeldes gefällt werden. Auf diese Weise bleibt der Roboter in einer defensiven Stellung und kann durch die bessere Sichtweite das ganze Spielfeld überblicken.

Das Nutzen der Netzwerkinformationen für die Ballsuche ist eine wichtige Erweiterung, die erst durch die Positionierung möglich geworden ist. Besitzt ein Roboter des Teams die genaue Ballposition, so teilt er diese über die Netzwerkverbindung seinen Mitspielern mit. Anhand des Szenarios zur Positionierung, kann sich jeder Roboter zu der virtuellen Ballposition begeben. Ist eine direkte Sicht zum Ball hergestellt, so terminiert das Szenario mit dem Status *Verhalten Erfolgreich* und die Rolle übernimmt erneut die Entscheidungsgewalt. Die Funktionsweise des Szenarios ist in der Abbildung 4.4 dargestellt.

## Die Laufbewegung

Dieses Bewegungsszenario ist nach dem von H. Moballegh [20] beschriebenen Prinzip innerhalb des FUmoids Projektes realisiert worden. Die Beschreibung bleibt deshalb im Rahmen, der für diese Diplomarbeit relevanten Aspekte.

Es handelt sich hierbei um eine *dynamische Bewegung* bei der, für jede Sequenz des Szenarios die Motorpositionen neu berechnet werden, siehe Abschnitt 3.2.1. Durch die dynamische Eigenschaft können schnell instabile Zustände bei der Bewegung erkannt und entsprechende Stabilisierungsverfahren eingeleitet werden.

Sollte der Roboter zum Beispiel stürzen, so wird ein kontrolliertes Fallen ausgelöst. Auf diese Weise kann die Hardware besser geschützt werden. Bevor das Szenario terminiert, nimmt es den Status *Gestürzt* an und signalisiert so den höheren Ebenen den Vorfall.

Der Roboter besitzt durch seine Konstruktion die Fähigkeit für omnidirektionales Laufen, das im Fall der neuen Robotergeneration in alle Richtungen dieselben Eigenschaften hat. So können gleich hohe Geschwindigkeiten beim Vorwärts-, Rückwärts- und Seitwärtslaufen erreicht werden. Im Gegensatz dazu ist bei der älteren Generation die Laufbewegung nach Vorne deutlich stärker und stabiler, als in den anderen Richtungen. Zusätzlich besitzen beide Robotergenerationen die Fähigkeit zur Rotation um die eigene Achse.

Die verschiedenen Bewegungsformen können kombiniert werden um unterschiedliche Bewegungsabläufe zu erzeugen. Zur Manipulierung der Bewegung wird dem Planungssystem ein Tripel  $\{f, s, r\}$  von Geschwindigkeiten bereit gestellt. Das Tripel beschreibt die Kombination aus *Vorwärts-, Seitwärts-, und Rotationsgeschwindigkeit*. Durch das Vorzeichen der Eingabe kann die Richtung der Bewegung beeinflusst werden, siehe Abbildung 4.5.

Die Laufbewegung wird im Gegensatz zu anderen Bewegungen, durch eine Start- und Stopp-Methode verwaltet. Der Grund dafür liegt in der Sensibilität des Verfahrens und der häufigen Verwendung der Laufbewegung. Die Start-Methode erkennt ob das Laufszenario bereits aktiv ist und veranlasst abhängig davon den Aufruf der Methode *switch-Motion()*, siehe Abschnitt 3.3.1. Die Aufgabe der Stopp-Methode ist es den Roboter schnellstmöglich stabil zum Still stehen zu bringen. Dafür werden alle, für das Laufen wichtige Parameter, auf einen bestimmten Stoppwert gesetzt. Auf diese Weise pendelt sich die Bewegung dynamisch aus.

Jedes Verhaltensszenario manipuliert dynamisch die Laufbewegung durch das oben be-

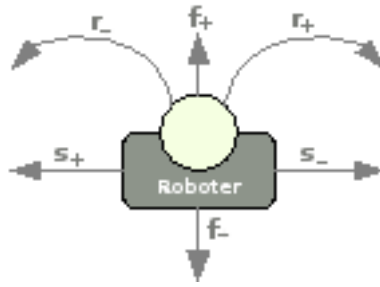


Abbildung 4.5: Der Einfluss der Bewegungsparameter und ihr Vorzeichen auf die Laufrichtung des Roboters.

schriebenen Tripel von Parametern. Dadurch können leicht instabile Zustände entstehen, die vermieden werden sollen. Im Hintergrund steht oft eine falsche Parametrisierung der einzelnen Faktoren. Das Problem der Instabilität ist auch bei der gleichzeitigen Verwendung von der Rotations- und der Seitwärtsbewegung häufig beobachtet worden. Die Ursache ist die gleichzeitige Kontrolle derselben Motoren mit unterschiedlichen Werten. Um die Entstehung instabiler Bewegungen zu vermeiden, sind unterschiedliche Mechanismen, sowohl bei der Laufbewegung als auch in den Verhaltensszenarien eingesetzt worden. Die Problematik besteht darin, das dynamische System so einzugrenzen, dass es stabil bleibt aber nicht zu träge wird. Eine schnelle Reaktionszeit ist beim Fußballspiel unentbehrlich. Beispielsweise muss der Roboter sofort auf unerwartete Hindernisse reagieren können. Andererseits können zu schnelle Bewegungen bei der Vorbereitung der Schussbewegung den Ball unbeabsichtigt treffen, wodurch eine Neupositionierung erforderlich wird. Grundsätzlich unterstützt eine träge Laufbewegung die Genauigkeit der Positionierung, während ein weniger träges Laufen eine kurze Reaktionszeit ermöglicht. Die eingesetzte Lösung erlaubt es jedem Verhaltensszenario einen Vorwärts-, Seitwärts-, Rotations- und Trägheitsparameter zu setzen. Diese Werte werden dann zentral gefiltert, bevor sie zu der Laufbewegung weitergeleitet werden. Die Filterung findet durch die Begrenzung der Vektorlänge statt, wobei der Vektor aus der Vorwärts- und Seitwärtsbewegung gebildet wird. Hierfür wird experimentell eine Vektorlänge ermittelt, die dem Roboter maximale Geschwindigkeit und einen stabilen Bewegungsablauf ermöglicht. Zusätzlich geht der Trägheitsparameter multiplikativ in die Berechnung des Vektors mit ein. Damit die Geschwindigkeiten im erlaubten Bereich bleiben, wird sowohl die Vorwärts-, Seitwärts-, als auch die Rotationsgeschwindigkeit einzeln normalisiert. Erst der normalisierte Bewegungstripel wird zur Laufbewegung weitergereicht.

### 4.1.3 Das Aufstehen

Während der Roboter agiert, kann ein Sturz durch instabile Zustände oder äußere Einflüsse verursacht werden. Es ist die Aufgabe jedes Szenarios diesen Fall abzufangen und entsprechend zu behandeln. Damit der Roboter wieder einsatzbereit ist, muss er die Fähigkeit besitzen sich von alleine wieder aufzurichten. Für die Realisierung dieser Aufgabe ist das Szenario *Aufstehen* implementiert worden. In der folgenden Abbildung 4.6 ist die

Funktionsweise dieses Verhaltensszenarios zu sehen.

Das Szenario ist anhand von statischen Bewegungen umgesetzt worden, diese erfordern

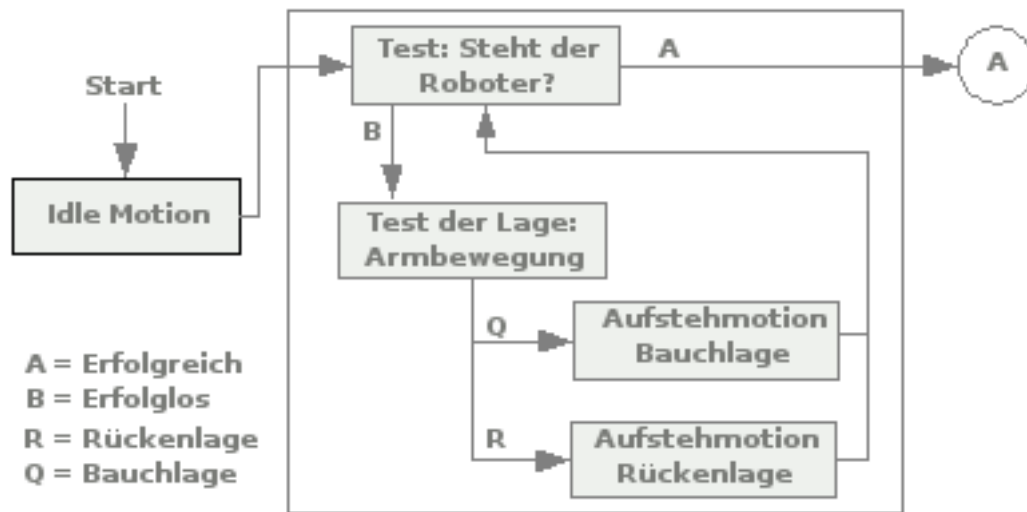


Abbildung 4.6: Das Funktionsdiagramm von dem Szenario: Aufstehen.

eine genaue Kenntnis der Ausgangslage. Beim Aufstehen wird zwischen einer Bauch- und einer Rückenlage des Roboters unterschieden, da dafür verschiedene Bewegungen erforderlich sind.

Die Ausgangslage des Roboters ist bei dem Aufruf des Aufstehszenarios unbekannt. Weiterhin sind keinerlei Sensoren vorhanden, aus denen diese Information direkt gewonnen werden kann. Um die Lage zu bestimmen, muss ein Umweg über die Motorpositionen genommen werden. Hierfür wird der Roboter als erstes in die bekannte *Idle Position*, siehe Abschnitt 4.1.1, bewegt. Aus dieser Position bieten sich, für die Untersuchung der Lage, die Schultermotoren an.

Beide Motoren werden zu einer ausgewählten Zielposition bewegt, wodurch die Arme des Roboters nach vorne gedrückt werden. Liegt der Roboter frontal auf dem Boden, so verhindert der Untergrund die Ausführung der Bewegung. Es handelt sich um eine Rücken- oder Seitenlage des Roboters, wenn beide Motoren vollständig die Zielposition erreicht haben. Im Falle der FUmoids ist eine Unterscheidung nicht notwendig, weil der Roboter anhand derselben Bewegung aus beiden Ausgangssituationen aufstehen kann. Beachtet werden muss, dass die Servomotoren für die Dauer der Bewegung auf eine minimale Steifheit eingestellt werden. Andernfalls können sie durch den oben beschriebenen Test beschädigt werden.

Nur wenn beide Motoren gemeinsam die Zielposition erreichen bzw. nicht erreichen, wird die entsprechende Bewegung für das Aufstehen durchgeführt. Im entgegengesetzten Fall wird der Test wiederholt. Durch die Vermeidung einer falschen Bewegung wird einer Schädigung der Hardware vorgebeugt. Die Ausführung einer falschen Aufstehbewegung gefährdet insbesondere den Kopf des Roboters, wo die Kamera und bei der neuen Generation der Großteil der Elektronik angebracht ist. Ein Defekt im Kopfbereich kann den

Roboter spielunfähig machen.

Die Wahl ungünstiger Motorpositionen für die Armbewegung sowie Vergleichswerte zur Messung der Lage können ein Deadlock verursachen. Die Wahrscheinlichkeit für das Auftreten eines Deadlock's lässt sich jedoch durch ausreichende Tests soweit minimieren, dass er keine praktische Gefahr darstellt.

Nach der Ausführung der Bewegung prüft der Roboter ob er erfolgreich aufgestanden ist. Hierfür wird bei den neuen Robotern das Feedback der Fußsensoren eingesetzt, siehe Abschnitt 1.3.2. Bei der alten Generation ist der oben beschriebene Test auch für zwei Fußmotoren entwickelt und zu diesem Zweck eingesetzt worden.

Das Szenario wird so lange wiederholt bis der Roboter wieder auf seinen eigenen Füßen stehen kann. Ein vorzeitiger Abbruch ist innerhalb der Software nicht vorgesehen, so dass dieses Szenario nur mit dem Status *Verhalten Erfolgreich* terminieren darf.

### **Die Bewegung zum anheben der Arme**

Die Bewegung zum anheben der Arme ist dafür zuständig die oben genannten Schultermotoren in ihre Zielposition zu führen. Hierbei ist lediglich die richtige Wahl der Zielposition wichtig. Sie soll so ausgewählt werden, dass die Motoren möglichst geschont bleiben, sich aber dennoch eine deutliche Grenze für den Test ergibt.

### **Die Aufstehbewegung**

Insgesamt sind zwei Arten von statischen Aufstehbewegungen aufgenommen worden: eins mit der Rückenlage und eins mit der Bauchlage als Ausgangsposition. In beiden Bewegungen spielen die Arme eine zentrale Rolle. Der Roboter stützt sich auf die Arme und hebt den Körper an, um eine hockende Position zu erreichen. Nachdem sich sein Schwerpunkt wieder auf den Fußflächen befindet, muss er zum Aufrichten noch seinen Körper strecken.

Der Roboter benötigt mehrerer Zwischenschritte um die Streckbewegung durchzuführen, da die Beinmotoren große Drehungen unter starker Last ausführen müssen. Die Motoren in den Knien sind dabei am stärksten belastet. Um die Bewegung zu ermöglichen, müssen die Beinmotoren ausreichende Zeit für die Drehbewegung haben.

#### **4.1.4 Sich dem Ball annähern**

Innerhalb des Szenarios *Ball annähern* wurden zwei unterschiedliche Aufgaben umgesetzt. Die erste Aufgabe heißt *Gehe zum Ball* und besteht darin, dass der Roboter von einer Entfernung aus, sich zum Ball hin bewegt. Die zweite Aufgabe *Gehe hinter den Ball* beschreibt eine Positionierung des Roboters auf der vom Ball und gegnerischem Tor gebildeten Geraden. Dabei soll der Roboter in Richtung gegnerisches Tor orientiert sein und sich in der Nähe des Balles befinden. Diese Position wird im folgenden Text als *die Ausrichtung hinter den Ball* bezeichnet.

Für die Umsetzung des Szenarios *Ball annähern* ist ein Algorithmus entwickelt worden, bei dem die erste Aufgabe fließend in die Zweite übergeht. Das Ziel ist, dass der Roboter sich so früh wie möglich auf eine Laufbahn, hinter den Ball bewegt. Im Gegensatz dazu,

kann der Roboter geradlinig auf den Ball zugehen. Bei dem Ball angekommen, rotiert der Roboter so lange um den Ball bis er eine direkte Sicht auf das gegnerische Tor hat. Siehe Abbildung 4.7 für eine grafische Darstellung der beiden Laufbahnen.

Ein fließender Übergang zwischen der Aufgabe *Gehe zum Ball* und *Gehe hinter den Ball*

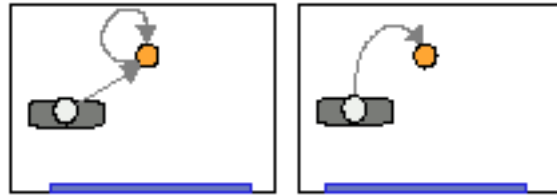


Abbildung 4.7: Links: Darstellung der Laufbahn mit einer klaren Grenze der beiden Aufgaben; Rechts: Darstellung der Laufbahn mit einer fließenden Grenze der beiden Aufgaben.

bringt klare Vorteile mit sich:

- Der Weg hinter den Ball ist kürzer und kann deshalb schneller abgelaufen werden.
- Die Laufbahn ist ein Bogen, auf dem der Roboter mit gleichmäßig hoher Geschwindigkeit gesteuert werden kann. Dies bewirkt einen stabilen Bewegungsablauf und eine schnelle Erfüllung der Aufgabe.
- Der Ball befindet sich erst im letzten Schritt der Bewegung in direkter Nähe des Roboters. So kann verhindert werden, dass der Roboter unbeabsichtigt den Ball bewegt.
- Der Roboter dreht sich schrittweise zu dem gegnerischen Tor um und befindet sich dadurch in einer Angriffsposition. So versperrt er gegnerischen Spielern die Sicht auf das eigene Tor. Zusätzlich stellt er ein Hindernis auf ihrem Weg dar.

Für das Szenario sind mehrere Fähigkeiten der neuen Roboter ausgenutzt worden, wie z.B. die verbesserte Rechenleistung, die Fähigkeit zur Selbstlokalisierung und das zu allen Seiten gleich starke omnidirektionale Laufen. Die Ausgangssituation des Roboters ist durch das erfolgreiche Beenden des Verhaltens *Finde den Ball* gegeben. Aus dem Abschnitt 4.1.1 ist bekannt, dass der Ball nun im Sichtfeld des Roboters zentriert ist, wo der Kopf um den Winkel  $\alpha$  vom Körper weggedreht ist.

Der intuitive nächste Schritt ist es den Winkel  $\alpha$  durch die Drehung des Körpers zu korrigieren, das aber in der Praxis zu lange dauert. Es hat sich als effektiver erwiesen auf den Ball sofort zu zulaufen und gleichzeitig die Kopf- und die Körperdrehung kontinuierlich zu korrigieren. Im Verlauf dieser Bewegung besteht die Gefahr, dass der Roboter den Ball aus seiner Sichtweite verliert. Um dies zu vermeiden, wird der Ball in dem Szenario sequenziell zentriert.

Durch die Kopfdrehung sind alle nachfolgenden Berechnungen, um den Drehwinkel  $\alpha$ , verfälscht. Als Korrektur wird bei der zentralen Berechnung der relativen Position von

Objekten, eine Drehung im euklidischen Raum vorgenommen, siehe Gleichung (4.1). Dort wird der Positionsvektor jedes Objektes aus dem Koordinatensystem der Kamera in das, des Roboterkörpers gedreht.

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} P_{x_{old}} \\ P_{y_{old}} \end{pmatrix} \quad (4.1)$$

Ist  $\alpha = 0$  so bleibt die Position des Objektes unverändert. Auf diese Weise können, trotz des gedrehten Kopfes, korrekte Daten zur Position der Objekte erhalten werden.

Unabhängig von der Kopfdrehung befindet sich der Körper auf dem Weg hinter den Ball. Dafür muss das Szenario eine Sequenz von Bewegungstripel  $\{f, s, r\}$  erzeugen, die diesen Weg beschreiben. Ein Tripel wird dynamisch aus den aktuellen Daten der Umgebung generiert, die in jeder Sequenz kontrolliert werden. So kann das System, durch die Erzeugung eines neuen Bewegungstripels, auf Veränderungen der Umwelt schnell reagieren.

Der Begriff *Positionsfehler* wird als die Differenz zwischen der Position des Roboters und der gewünschten Zielposition, relativ zu einem Objekt, definiert. Hierzu gehört auch die Winkeldifferenz des Objektes bezüglich der Y Achse, siehe Abbildung 4.8. Die Zielposition ist in dem aktuellen Szenario, als ein Punkt  $P(x,y) = (0, 20)$  in der Maßeinheit *cm*, der realen Welt definiert. Sie beschreibt, dass eine zentrale Ausrichtung des Roboters in der Nähe des Balles angestrebt wird.

Die Berechnung des Positionsfehlers zu einem bestimmten Objekt wird durch die Me-

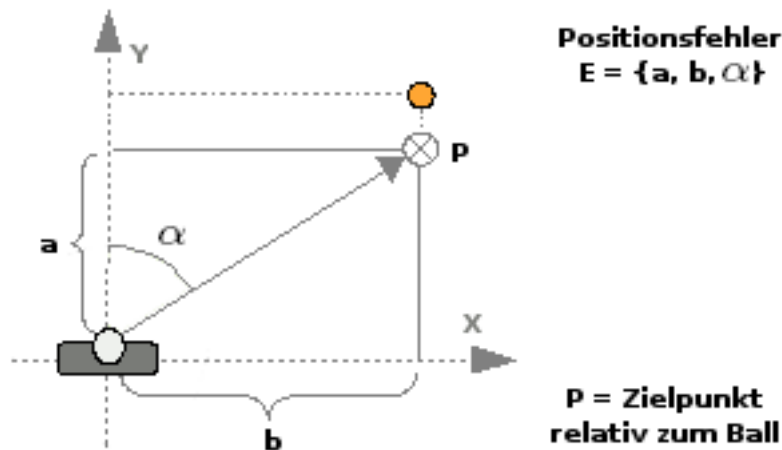


Abbildung 4.8: Die grafische Darstellung des Positionsfehlers zu dem Objekt Ball.

thode `getPositionError(objectPosition)` allen Verhaltensszenarien zur Verfügung gestellt. Der Rückgabewert ist vom Typ *Positionsfehler*, der die oben genannten Fehlerdaten bezüglich des Objektes bündelt.

Die Parameter des Bewegungstripels sind von den Objekten Ball und gegnerisches Tor abhängig. Dabei beeinflusst die Ballposition alleine die Vorwärtsgeschwindigkeit  $f$  und die Seitwärtsgeschwindigkeit  $s$ , während die Rotationsgeschwindigkeit  $r$  aus der Position beider Objekte berechnet wird.

Bei dem Szenario *Ball annähern* wird als erstes der Positionsfehler zum Ball berechnet.

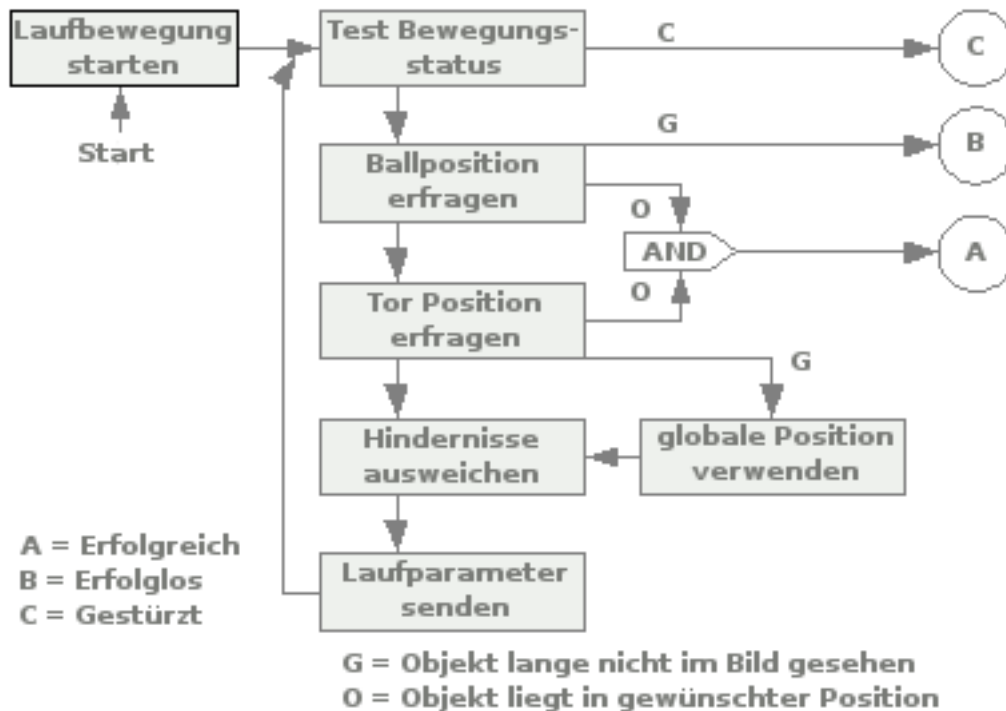


Abbildung 4.9: Das Funktionsdiagramm des Szenarios: Ball annähern.

Der Fehlerwert wird auf der Y Achse der Vorwärtsgeschwindigkeit und auf der X Achse der Seitwärtsgeschwindigkeit zugewiesen. Die Rotationsgeschwindigkeit wird aus der Winkeldifferenz zum Tor und zum Ball berechnet. Das Ziel ist, dass sich der Roboter, das gegnerische Tor und der Ball auf einer Gerade befinden. Dieser Effekt entsteht durch die Berechnung der Rotationsgeschwindigkeit mittels der arithmetischen Summe der beiden Winkeln.

Bevor das Tripel gefiltert wird und der Laufbewegung weitergereicht wird, überprüft das Szenario ob es Hindernisse auf dem Weg des Roboters gibt. Ist dies der Fall, so werden die Geschwindigkeiten neu berechnet. Hierfür wird das *Motor Schema* Modell, das im Abschnitt 2.3.3 beschrieben ist, eingesetzt.

Die Berechnungen laufen sequenziell innerhalb des Szenarios ab. Erst wenn die Aufgabe beendet ist, terminiert das Verfahren mit einem entsprechenden Status. Sollte der Roboter hinfallen, empfängt das Szenario das Feedback der *Bewegungsebene* und terminiert mit dem Status *Gestürzt*. Es ist ebenfalls möglich, dass der Ball durch äußere Einflüsse außerhalb des Sichtfeldes gerät. In diesem Fall wird der Status *Verhalten Erfolglos* gesendet. Das Fehlen des gegnerischen Tores stellt kein Problem dar. Die Position der Tore auf dem Feld ist fest und durch die Selbstlokalisierung permanent bekannt.

Gelingt es dem Roboter die Position hinter den Ball einzunehmen, so wird das Szenario mit *Verhalten Erfolgreich* terminieren.

Dieses Szenario erfüllt zwei Aufgaben gleichzeitig, die wegen der bereits genannten Vor-

teile in einem Szenario umgesetzt worden sind. Der Übergang in das Szenario *Gehe hinter den Ball* bestätigt die Nähe zu dem Ball, da der Roboter voraussichtlich ab hier einen direkten Ballkontakt behält. Wird der Ball durch äußere Einflüsse entfernt, so kommt das Szenario *Gehe zum Ball* erneut zum Einsatz. So können die oberen Ebenen den Ballverlust erkennen. Dieser Wechsel ist für die Planung von großer Bedeutung, da er eine zuverlässige Aussage über den Abstand des Roboters zum Ball trifft.

Um die Informationen über die Entfernung zum Ball an die Hierarchie zu senden, wird

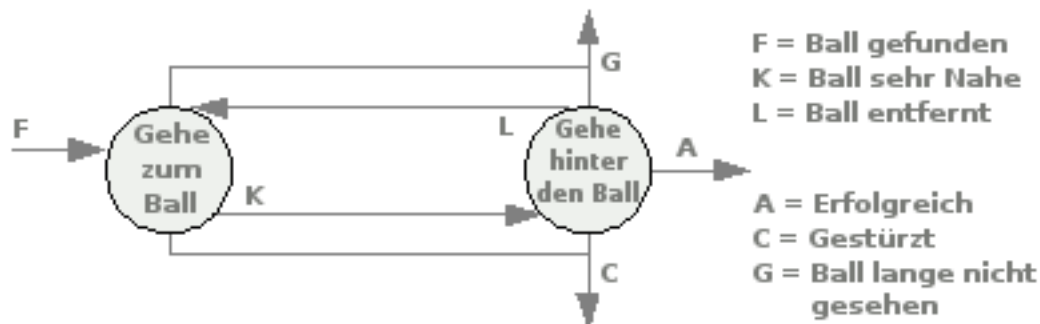


Abbildung 4.10: Die Darstellung der inneren Zustandsübergänge zwischen der Aufgabe: Gehe zum Ball und Gehe hinter den Ball.

innerhalb des Szenarios *Ball annähern* ein Wechsel zwischen den beiden Zuständen durchgeführt. Bei der Konstruktion des internen Zustandsautomaten ist es wichtig, dass das Szenario jederzeit unterbrochen werden kann. Angenommen, der Roboter fällt hin und wird so an der Erfüllung der Aufgabe zum Ball zu gehen, gehindert. Nun muss das Szenario *Ball annähern* sofort abgebrochen werden, damit die Rolle das Aufstehverhalten starten kann. Analog verhält es sich mit dem Zustand *Gehe hinter den Ball*. Die genauen Zustandsübergänge des Automaten sind in der Abbildung 4.10 dargestellt.

#### 4.1.5 Das Dribbeln

Dieses Verhaltensszenario hat die Aufgabe das, vom richtigen Fußballspiel bekannte Dribbeln, zu realisieren. Dabei müssen die Grenzen des Roboters beachtet werden.

Die Rolle ruft das Szenario *Dribbeln* auf, wenn der Roboter hinter den Ball erfolgreich positioniert ist. Nun kann er zusammen mit dem Ball näher an das gegnerische Tor gelangen. Die Fähigkeit zu dribbeln ermöglicht es dem Roboter den Ball in sichere Schussweite zu bringen. Zum Beispiel kann in der Nähe des eigenen Tores ein Schuss, Hindernisse treffen und zurückprallen und so ein Eigentor verursachen.

Die Fähigkeit um Hindernisse herum zu dribbeln, ist in diesem Szenario wichtig. In einer eins gegen eins Situation kann der Roboter den Ball aus der Sackgasse zur Seite dribbeln. Im Vergleich zu einem Seitwärtspass bleibt der Ball beim Dribbeln in der Nähe des Roboters und kann sofort weiterbewegt werden. Zusätzlich ist der Roboter kontinuierlich in Bewegung, was dem Spiel mehr Dynamik verleiht.

Der Aufbau des Programms *Dribbeln* ähnelt dem Szenario *Ball annähern*. Beide Szenarien erzeugen eine Sequenz von Bewegungstripel  $\{f, s, r\}$  die sequenziell aktualisiert

werden. Hierfür wird die Vorwärts- und Seitwärtsgeschwindigkeit aus dem Positionsfehler des Balles berechnet. Die Rotationsgeschwindigkeit ergibt sich aus der Kombination des Winkels vom Roboter zum Ball und vom Roboter zum gegnerischen Tor. Die Zielposition bezüglich des Balles hat sich geändert. Sie befindet sich sowohl horizontal als auch vertikal, mittig zu den Füßen des Roboters. Um diese Position zu erreichen, werden die Füße des Roboters in jeder Sequenz des Bildes als Objekte gesucht. Das Ziel der neuen Position ist es den Ball unter maximaler Kontrolle zu halten und ihn gleichmäßig frontal zu bewegen.

Nachdem der Bewegungstripel berechnet ist, wird er mit der, aus Abschnitt 4.1.2 bekannten Filtermethode, an die Laufbewegung weitergeleitet. Dabei ist die Parametrisierung der einzelnen Komponenten experimentell festgelegt worden.

Eine Bewegung zur Seite wird erst notwendig, wenn Hindernisse direkt vor dem Roboter auftauchen. In diesem Fall haben die FUManoide zwei unterschiedliche Lösungsansätze erstellt.

In dem ersten Ansatz, der bei den alten Robotern eingesetzt worden ist, wird die Rotationsgeschwindigkeit auf  $r = 0$  festgelegt, um so eine eins gegen eins Situation hervorzurufen. Der FUManoide Roboter bewegt den Ball weiterhin nach Vorne und bewirkt, dass dieser vom Gegner abprallt. Nach einigen Versuchen rollt der Ball zur Seite und der Roboter kann ihn mit einigen Seitwärtsschritten einholen. Danach setzt er sein Dribbeln frontal fort.

Der zweite Ansatz ist im FUManoide Projekt für die neue Robotergeneration entwi-

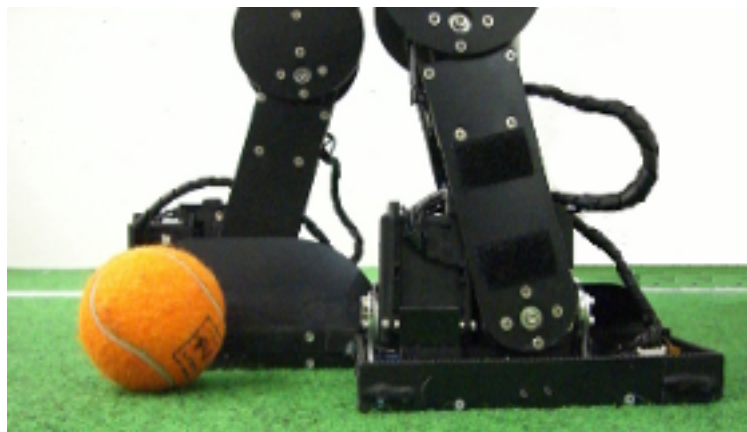


Abbildung 4.11: Die Fußposition beim Orthodribbeln.

ckelt worden und hat den Namen *Orthodribbeln* erhalten. Mit dieser Methode ist die Seitwärtsbewegung des Roboters geschaffen worden, durch die der Roboter den Ball zur Seite bewegen kann. Dabei stellt der Roboter einen Fuß vor dem Anderen, siehe Abbildung 4.11 und bewegt sich gleichzeitig zur Seite. Diese Bewegungsform ist in beide seitliche Richtungen möglich und wird dynamisch in der Laufbewegung durch den Aufruf einer Funktion umgesetzt.

In dem Szenario *Dribbeln* befinden sich zwei Unterszenarios: das *frontale Dribbeln* und

das *Orthodribbeln*, für das Ausweichen von Hindernissen. In der Abbildung 4.12 wird das Funktionsdiagramm des vollständigen Szenarios dargestellt. Hier kann das Schalten zwischen dem normalen Dribbeln und dem Orthodribbeln sowie die gesendeten Statusinformationen betrachtet werden.

Die Spielstrategie entscheidet darüber, ob der Roboter von weitem auf das Tor schießt

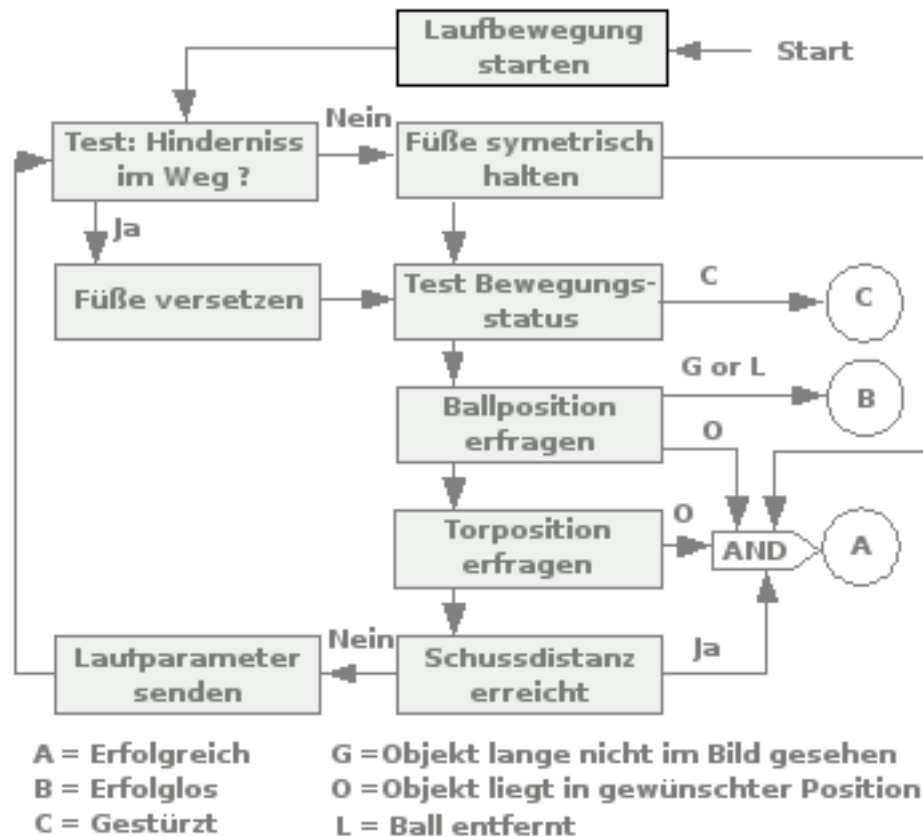


Abbildung 4.12: Die Darstellung der Funktionsweise des Szenarios: Dribbeln.

oder ob er bis über die Torlinie dribbelt. Abhängig vom gegnerischen Team können beide Strategien nützlich sein. Es bietet sich an, dem Dribbelszenario einen Parameter zu übergeben mit der die gewünschte Schussweite angegeben werden kann. Auf diese Weise können viele Strategien angewendet werden.

#### 4.1.6 Das Schießen

Das Verhaltensszenario *Schießen* wird gestartet, wenn der Roboter die gewünschte Schussentfernung erreicht hat und keine Hindernisse vor ihm stehen. In diesem Szenario wird die genaue Schussposition vorbereitet und danach der Schuss ausgeführt.

In der Abbildung 4.13 ist die Funktionsweise des Szenarios graphisch dargestellt. Dabei sind Ähnlichkeiten zu dem Verhalten *Ball annähern* und auch zum *Dribbeln* zu erkennen.

Auch hier wird der Bewegungstripel mit Hilfe derselben Objekte Ball und Tor zusam-

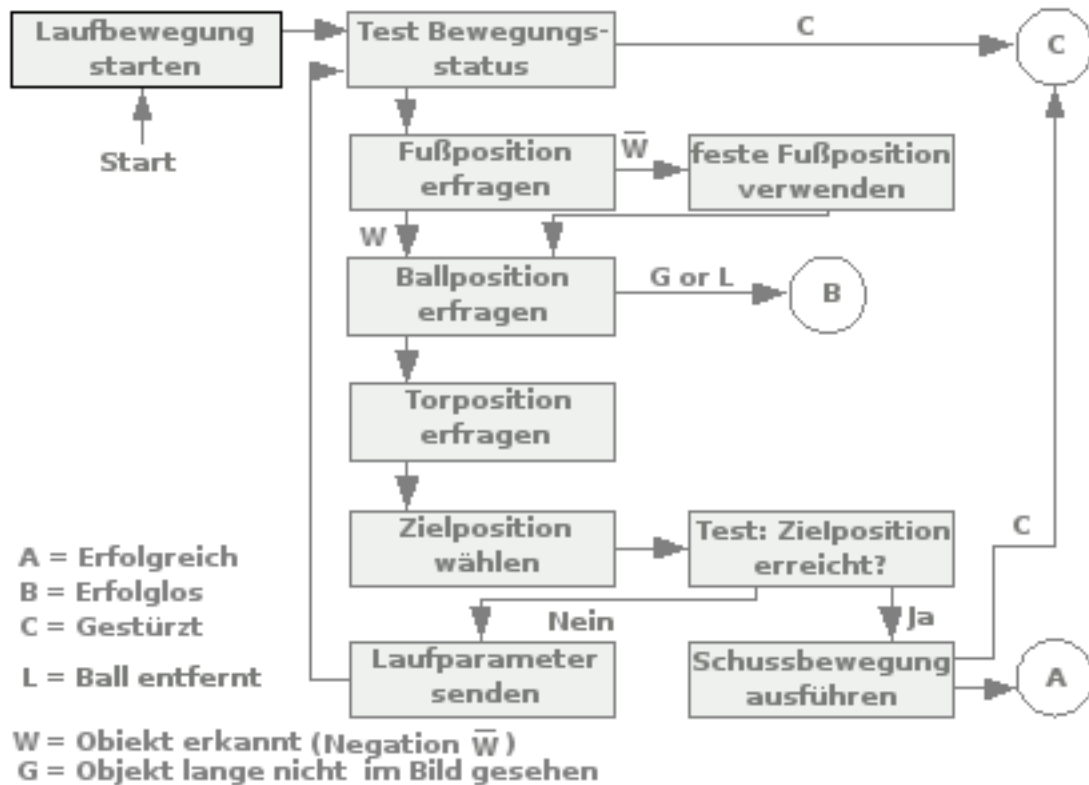


Abbildung 4.13: Die Darstellung der Funktionsweise des Verhaltensszenarios: Schießen.

mengesetzt. Bevor es aber zur Berechnung des Positionsfehlers kommen kann, muss sich der Roboter für eine Zielposition entscheiden. Zur Auswahl stehen die festkodierte Positionen für einen Schuss mit dem linken bzw. rechten Fuß.

Für diese Entscheidung wird die genaue Balllage relativ zu den Füßen des Roboters untersucht. Dabei ist es wichtig, dass die Ballposition zuverlässig ist. Um dies sicherzustellen, wird folgende Filtermethode auf die Messwerte angewandt. Die Position eines Objektes berechnet sich dabei aus den alten und aktuellen Messwerten. Der alte Wert fließt zu zehn Prozent in das Resultat mit ein. Dies genügt um die Messwerte zu stabilisieren. Weil es zu Beginn einer Messung keinen alten Wert gibt, wird die Aufnahme ganz übernommen. Die Position der Füße wird aus den Bildinformationen gewonnen und sequentiell aktualisiert. Im Falle, dass die Füße längere Zeit nicht im Sichtfeld des Roboters sind, wird eine konstante Position für die Berechnung der Balllage verwendet. Abhängig davon, zu welchem Fuß der Ball näher liegt, wird eine entsprechende Zielposition gewählt. Diese Zielposition wird für die Berechnung des Positionsfehlers eingesetzt, um dann die entsprechenden Geschwindigkeiten zu gewinnen.

Es ist wichtig, dass der Roboter die Zielposition so exakt wie möglich einnimmt. Um dies zu unterstützen wird mehrmals eine Bestätigung der Lage gefordert, bevor die Schuss-

bewegung aktiviert wird. Diese Abfrage wird innerhalb der Methoden *isBallKickableRight(ballPosition)* bzw. *isBallKickableLeft(ballPosition)* bearbeitet. Diese Funktionen überprüfen, ob die Differenz zwischen Fuß- und Ballposition innerhalb einer konstanten Grenze liegt. Zu jeder Zielposition wird ein maximaler Fehler festkodiert und als Bezugspunkt verwendet. Diese Konstanten werden experimentell erfasst, so dass sie für jeden Roboter des Teams allgemein gültig sind.

Kann die Überprüfung mehrmals eine positive Antwort liefern, so wird die Schussbewegung aktiviert. Ist dies nicht der Fall, so muss sich der Roboter neu positionieren und das berechnete Bewegungstripel gefiltert werden, siehe Abschnitt 4.1.2. Die gefilterten Werte werden an die Laufbewegung gesendet.

Durch die erneute Positionierung besteht die Gefahr eines Deadlocks. Der Roboter besitzt eine hohe Anzahl an Fehlerquellen, die sich besonders auf kleine Bewegungen stark auswirken. Der Algorithmus muss dennoch schnellstmöglich in der richtigen Position den Roboter stoppen.

Gelingt es dem Roboter die Zielposition zu erreichen, so wird eine *Schussbewegung* gestartet. Bei den FUManooids sind sowohl statische als auch dynamische Schussbewegungen entwickelt worden. Beide Varianten sind in den Spielen eingesetzt worden, wobei die statische Variante bevorzugt worden ist. Der Grund dafür liegt an ihrer größeren Schussweite und Schussstärke.

Das Szenario terminiert nach der Ausführung der Bewegung mit dem Status *Verhalten Erfolgreich*. Ein frühzeitiges Beenden wird durch den Ballverlust, einer zu großen Entfernung zum Ball und dem Sturz des Roboters ausgelöst.

## Die Schussbewegung

Für den statischen Schuss sind diverse Varianten der Bewegung ausprobiert und immer weiter stabilisiert worden. Der Roboter muss auf einem Fuß ausreichend balancieren können, damit der schwebende Fuß eine möglichst starke Schwungbewegung ausführen kann. Nach dieser Bewegung muss der Roboter wieder stabil auf beiden Füßen stehen können. Außerdem ist eine schnelle Ausführung der Bewegung von Vorteil, dies kann jedoch zu Instabilität führen.

Die mechanische Konstruktion des Roboters hat einen großen Einfluss auf die Bewegung. Eine leichte Konstruktion mit großzügiger Fußfläche und einer hohen Anzahl an Freiheitsgraden ist vorteilhaft. Die Stärke und die Geschwindigkeit der eingesetzten Servomotoren sind für die Bewegung ebenfalls ausschlaggebend. Eine statische Schussbewegung ist schwer zu stabilisieren und bedarf viel Wartung, anders ist es bei der Aufstehbewegung. Die dynamische Implementierung der Schussbewegung ist im Rahmen einer Bachelorarbeit umgesetzt worden. Der Ansatz dieser Arbeit besteht in der Modifikation der Laufbewegung. Hierbei wird der Moment genutzt, in dem der eine Fuß des Roboters sich in der Luft befindet, um eine frontale Fußbewegung auszuführen. Der Vorteil der Bewegung besteht in der Fähigkeit des Roboters schnell zu schießen. Der Nachteil ist, dass der resultierende Schuss von geringer Reichweite ist.

Ausschlaggebend für die Wahl der statischen Schussbewegung ist dessen Stärke und die Schussweite von bis zu 6 Meter.

## 4.2 Der Flügelspieler

Der Flügelspieler stellt eine Unterstützung für den Stürmerroboter dar. Alleine hat der Stürmer im Angriff geringe Chancen, gegen drei verteidigende Gegner. Aus diesem Grund muss sich ein weiterer Roboter, der Flügelspieler bei dem Angriff beteiligen. An die neue Rolle werden folgende Erwartungen gestellt:

- Ein Flügelspieler darf den Stürmer bei dem Angriff nicht behindern.
- Er muss die genaue Position des Balles kennen und in seiner Nähe bleiben.
- Ist der Stürmer unfähig seine Aufgabe zu erfüllen, so übernimmt der Flügelspieler schnellstmöglich die Weiterführung des Angriffs.
- Mehrere Roboter sollen im Laufe eines Spiels gleichzeitig diese Rolle übernehmen können.

Damit der Roboter diese Erwartungen erfüllen kann, muss eine sichere Aufstellung relativ zu der Position des Stürmers festgelegt werden. In Frage kommt eine Aufstellung hinter und seitlich vom stürmenden Roboter. Dabei wird die erstere verworfen, da sie nicht ausreichend Sicht auf den Ball bietet. So bleibt eine seitliche Aufstellung übrig, die dynamisch zur rechten bzw. linken Seite des Stürmers eingenommen werden soll.

Zusätzlich muss der Flügelspieler die Fähigkeit zur Ballsuche, zur Positionierung hinter

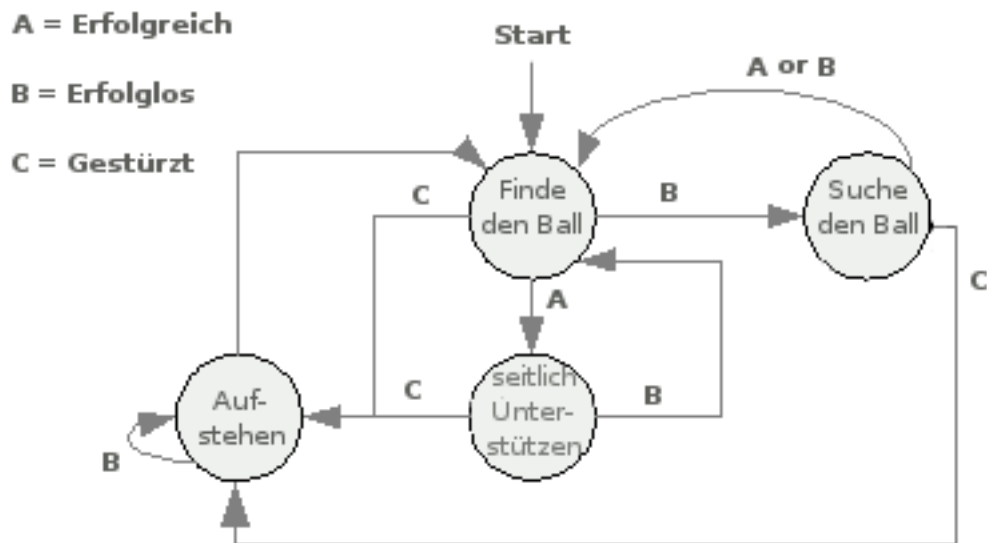


Abbildung 4.14: Das Zustandsdiagramm der Rolle: Flügelspieler. Die Zustände entsprechen den Verhaltensszenarien und die Zustandsübergänge ihrem Feedback.

den Ball und zum Aufstehen mitbringen. Wie die einzelnen Aufgaben in Szenarien zusammengefasst sind und die Zustandsübergänge stattfinden, wird in der Abbildung 4.14

dargestellt.

Der Befehl für die Übernahme des Angriffs unterliegt dem laufenden Strategieszenario. Hierfür findet ein Wechsel von Rollen statt, das nur von der höchsten Ebene der Hierarchie durchgeführt werden kann. Die Beschreibung des Verfahrens findet in Abschnitt 4.4 statt.

Der gleichzeitige Einsatz der Rolle auf mehrere Spieler, stellt von der Struktur der Planungsarchitektur her, kein Hindernis dar. Auf der logischen Ebene muss die Problematik gelöst werden, welcher Spieler sich wann auf welche Position ausrichten darf. Diese wird in dem Szenario *seitlich Unterstützen* auf der Hierarchieebene, Verhalten, realisiert. Weiterhin kann aus dem Zustandsdiagramm der Rolle entnommen werden, dass es kein separates Verhaltensszenario für die Positionierung hinter den Ball gibt. Auch diese Aufgabe ist in das Szenario *seitlich Unterstützen* integriert worden.

Da alle anderen involvierten Szenarien bereits vorgestellt worden sind, bleibt nur noch die Beschreibung des Verfahrens für das *seitliche Unterstützen* übrig.

#### 4.2.1 Das seitliche Unterstützen des Stürmers

Das Szenario des *seitlichen Unterstützens* bildet den Kern der Rolle *Flügelspieler*. Es verleiht dem Roboter die Fähigkeit, sich in einen Zustand passiver Aufmerksamkeit, zu versetzen. Durch seine direkte Ballnähe und die dynamische Anpassung, an die Bewegungen des Balles, ist er jederzeit bereit den Angriff des Stürmers zu übernehmen.

Das Szenario startet mit denselben Funktionen wie das, aus Abschnitt 4.1.4 bekannte Verfahren, zur Annäherung des Balles. Für den Fall das der Roboter noch keine Laufbewegung aktiviert hat, wird nun diese gestartet. Danach erfolgt sequenziell die Untersuchung des Zustandes, die Gewinnung von Bildinformationen und die Berechnung des Bewegungstripels  $\{f, s, r\}$ . Dieses Tripel wird noch in derselben Sequenz gefiltert und an die Laufbewegung weitergereicht.

Um das Bewegungstripel zu erzeugen, wird auch der Positionsfehler zwischen Ball und Zielposition des Roboters berechnet. Hierfür muss sich der Roboter als erstes für eine Position entscheiden. Zur Auswahl stehen, in Angriffsrichtung die Positionen  $P_r$  auf der rechten bzw.  $P_l$  auf der linken Seite, relativ zum Ball. Es handelt sich hierbei um zwei Punkte im Koordinatensystem dessen Ursprung im Zentrum des Roboterkörpers ist. Zusätzlich wird eine Orientierung zum gegnerischen Tor angestrebt.

Bei der Entscheidung auf welche Seite des Stürmers sich ein Roboter ausrichtet, ist zu beachten, dass es gleichzeitig bis zu zwei Roboter mit dieser Rolle auf dem Feld geben kann. Der einzelne Spieler muss ausreichend intelligent sein, um die für ihn günstigste und freie Position einzunehmen. Hierfür wird bei der Aktivierung des Szenarios die relative Ballposition zu dem Roboter untersucht. Abhängig davon, ob der Ball rechts oder links von ihm liegt, wählt der Roboter die zu ihm näherliegende Seite des Stürmers aus. Diese ermittelte Wunschposition muss sich der Roboter von seinen Mitspielern bestätigen lassen, bevor sie zu der endgültigen Zielposition wird. Hierfür werden die ID-Informationen, die er über den virtuellen Identifikationspfad, siehe Abschnitt 3.3.3 erhalten hat, verwendet. Das Verhaltensszenario besitzt für diesen Zweck drei getrennte ID's, die den Zustand der Entscheidung beschreiben. Beim Start des Programms wird die neu-

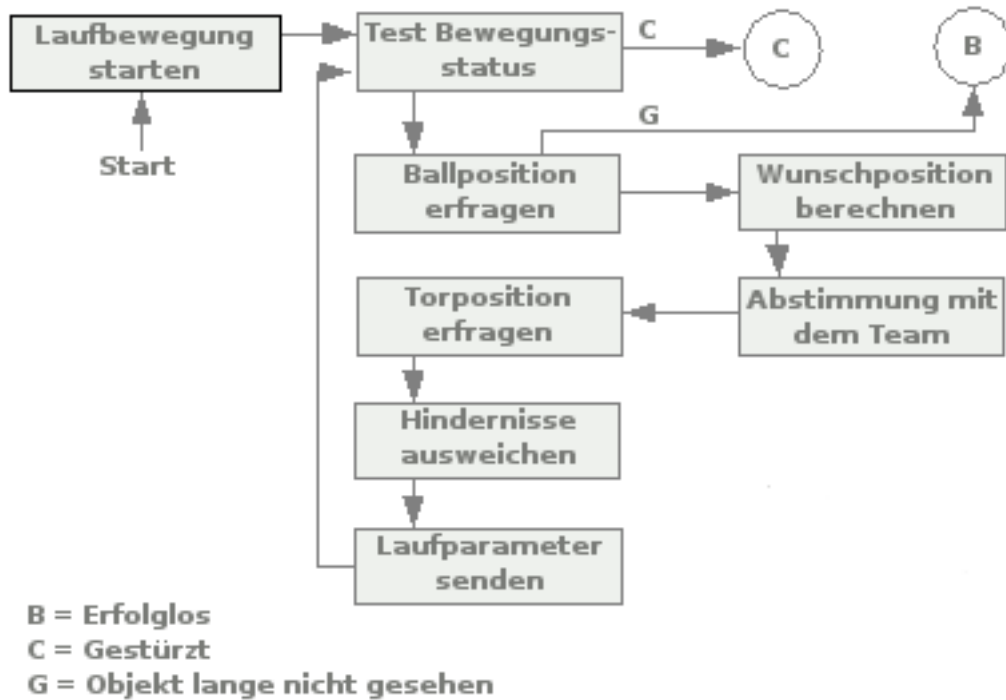


Abbildung 4.15: Das Funktionsdiagramm des Szenarios: Seitlich Unterstützen

trale ID BEHAVIOR\_SUPPORT\_LR vergeben. Sie bleibt erhalten solange bis der Roboter eine feste Entscheidung über seine Zielposition getroffen hat. Danach wird die neutrale ID mit der ID BEHAVIOR\_SUPPORT\_LEFT oder BEHAVIOR\_SUPPORT\_RIGHT entsprechend der Position ersetzt.

Angenommen, die Wunschposition des Roboters A befindet sich auf der rechten Seite des Stürmers, so überprüft das System ob bereits ein Roboter die ID BEHAVIOR\_SUPPORT\_RIGHT als ID-Information gesendet hat. Ist dies der Fall, so muss sich der Roboter A auf die linke Seite des Stürmers stellen. Sein nächstes ID-Signal wird BEHAVIOR\_SUPPORT\_LEFT beinhalten.

Es muss beachtet werden, dass sich zwei Roboter gleichzeitig im neutralen Zustand des Szenarios befinden können. Damit kein Konflikt entsteht, bekommt der Roboter mit höherer Priorität den Vorrang in seiner Entscheidungsfindung. Dabei ist die Priorität der Roboter durch ihre persönliche ID festgelegt. Wenn ein Roboter alle Konkurrenten für seine Wunschposition ausgeschlossen hat, darf er die Position als Ziel übernehmen und die entsprechende ID über den Identifikationspfad senden. Eine Darstellung der erfolgreichen Ausrichtung ist in der Abbildung 4.16 zu finden.

Die aktuelle Implementierung erlaubt es maximal zwei Robotern gleichzeitig dieses Verhalten auszuführen. Die Anzahl der Teilnehmer ist dabei direkt von der Anzahl der möglichen Positionen auf dem Feld abhängig. Folglich ist eine Erweiterung auf mehrere Positionen bzw. Teilnehmer möglich. Im Fall der FUManoide sind zwei Positionen ausrei-

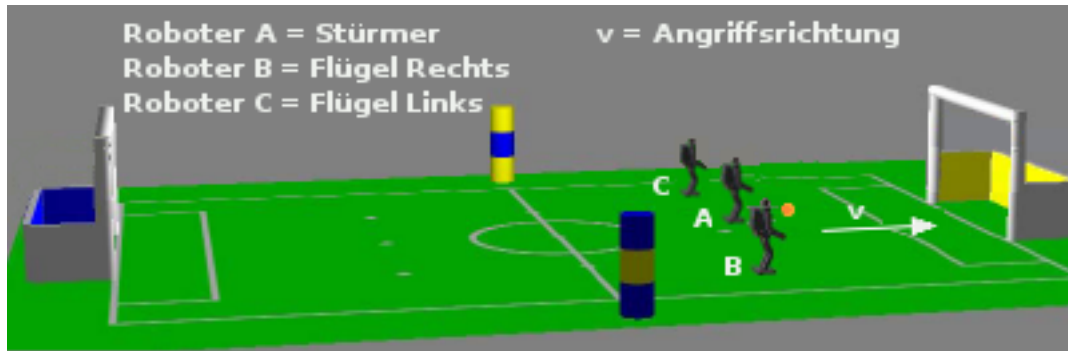


Abbildung 4.16: Die Darstellung der Orientierung und Positionierung des Flügelspielers auf dem Spielfeld.

chend, da insgesamt drei Roboter im Spiel teilnehmen, von denen einer bereits die Rolle *Stürmer* ausführt.

### 4.3 Der Unterstützer

Die Rolle des Unterstützers besteht darin, sich neben dem Stürmerroboter beim Angriff aktiv zu beteiligen. Es gilt folgende Spezifikation zu erfüllen:

- Der Roboter soll verhindern, dass die gegnerischen Spieler den Ball erlangen.
- Er soll den Weg des Stürmers zu dem gegnerischen Tor erleichtern.
- Er muss jederzeit bereit sein die Rolle des Stürmers zu übernehmen.
- Es soll ein Zusammenspiel mit Stürmer und Flügelspieler möglich sein.
- Er darf die Spieler seines Teams nicht behindern.

Diese Rolle ist nur für die Robotergeneration 2008 entworfen und implementiert worden, das impliziert eine andere Entwurfsstrategie als bei den neuen Robotern, da weniger Ressourcen vorhanden sind.

- Die Kamera hat eine niedrige Auflösung von 160x120 Pixel, dadurch wird die Sichtweite des Roboters stark eingeschränkt. Es sind neben den horizontalen auch vertikale Kopfbewegungen eingesetzt worden, um das Abscannen der Umgebung zu ermöglichen.
- Die Menge der zuverlässigen Orientierungspunkte beschränkt sich auf die beiden Tore. Wegen der geringen Rechenkapazität ist keine Linienerkennung realisiert worden. Auch die Erkennung der Säulen am Feltrand ist problematisch gewesen, da sie kleine Objekte in großer Entfernung darstellen.

- Eine Selbstlokalisierung ist nicht vorhanden, da die visuelle Kapazität und die Rechenkapazität des Roboters nicht ausreicht.
- Der Roboter besitzt nur eine Momentaufnahme der Welt. Die Informationen von vorherigen Aufnahmen können nur für kurze Zeit wiederverwendet werden, da sie schnell nicht mehr aktuell sind.
- Die Laufbewegung hat abhängig von der Richtung unterschiedliche Eigenschaften. Das Laufen nach Vorne ist stabil und schnell während das Rückwärts- und Seitwärtslaufen viel langsamer ausgeführt wird.

Bei der Entwicklung der Planung müssen all diese Faktoren mit betrachtet werden. Um seine Umgebung wahrzunehmen bewegt der alte Roboter öfter seinen Kopf als der Neue. Die Ball- und Torinformationen sind schnell nicht mehr aktuell, da keine globalen Daten zur Verfügung stehen und so kann früh ein Zustand der Orientierungslosigkeit eintreten. Dennoch ist die Rolle des *Unterstützers* erfolgreich umgesetzt worden. Es folgte zwar eine Portierung der Rolle auf die neue Roboterplattform, aber keine Anpassung an die neuen Ressourcen.

Die Anforderungen der Rolle *Unterstützer* ist zu einer konkreten Aufgabe zusammengefasst worden. Sie besteht daraus, dass der Roboter dem Stürmer den Weg zu dem gegnerischen Tor frei schafft. Hierfür positioniert sich der Roboter direkt vor dem Stürmer und läuft rückwärts Richtung gegnerisches Tor. Die genaue Ausrichtung des Roboters ist in der Abbildung 4.17 dargestellt.

Der Roboter behält direkten Blickkontakt zu dem Ball und orientiert sich an dem eigen-



Abbildung 4.17: Die Darstellung der Orientierung, der Positionierung und der Laufrichtung des Unterstützers auf dem Spielfeld

nen Tor. Auf diese Weise verdeckt er den Gegnern die Sicht auf den Ball und versperrt zusätzlich den Weg zu ihm. Hierdurch wird dem gegnerischen Torwartroboters das Erfüllen seiner Aufgabe erschwert. Er hat keinen Blickkontakt zu dem Ball und kann keine Abwehrreaktion vorbereiten.

Durch die Position des Stürmers und des Unterstützers befindet sich der Ball, von den zwei wichtigsten Richtungen: vorne und hinten aus, im Besitz des eigenen Teams. Es ist

schwierig für den gegnerischen Roboter im Besitz des Balles zu gelangen, da allein der seitliche Weg frei ist. Dieser Weg bietet die geringsten Erfolgschancen.

Wichtig bei dem Konzept ist, dass der Unterstützer rechtzeitig das Schussvorhaben des Stürmers erkennt. Hat der eigene Stürmer zum Schuss angesetzt, so muss der Unterstützer rechtzeitig den Weg für ihn frei machen. Diese Reaktion wird auf Basis der ID-Informationen umgesetzt, die über den virtuellen Identifikationspfad übermittelt wird. Ähnlich wie beim Flügelspieler wird diese Entscheidung auf der Verhaltensebene getroffen. Hierfür ist das Szenario *frontales Unterstützen* zuständig, welches im Abschnitt 4.3.1 beschrieben wird.

Ähnlich den bereits bekannten Rollen durchläuft auch diese, sowohl ein Szenario der

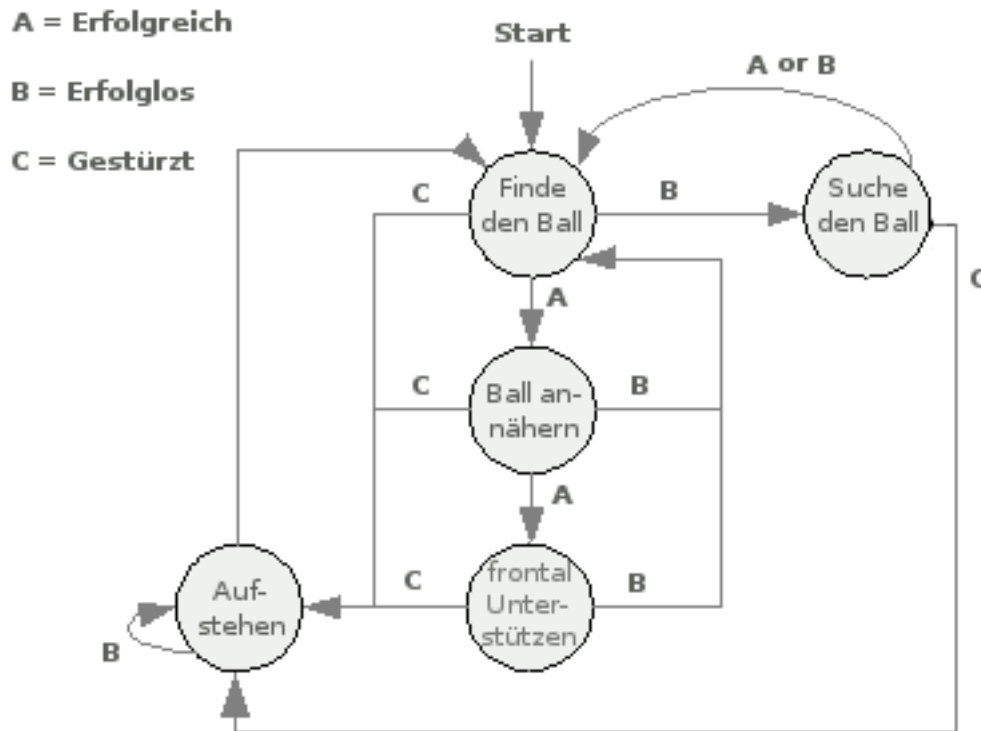


Abbildung 4.18: Das Zustandsdiagramm der Rolle: Unterstützer. Dabei entsprechen die Zustände den Verhaltensszenarien und die Zustandsübergänge deren Status beim terminieren.

Ballsuche als auch ein Szenario, um in Ballnähe zu gelangen. Die in Abschnitt 4.1.4 beschriebene Zusammenfassung der Aufgaben *Gehe zum Ball* und *Gehe hinter den Ball* ist bei den alten Robotern nicht möglich gewesen. Der Roboter bewegt sich geradlinig zum Ball und korrigiert erst im nachfolgenden Verhaltensszenario seine Ausrichtung.

Hat der Roboter seine Position als Unterstützer angenommen, verweilt er in dieser, solange die Strategieebene keine andersartigen Befehle sendet oder der Roboter stürzt. Die Darstellung der genauen Zustände und Zustandsübergänge ist in der Abbildung 4.18 zu sehen.

Bei dem Einsatz der Rolle *Unterstützer* muss darauf geachtet werden, dass die Gegner Hindernisse erkennen und sie umgehen. Das Ausweichen der Gegner, um die beiden Roboter hat den Vorteil, dass der Weg zum Tor frei wird. Bei einem Team, das keinen Hindernissen ausweichen kann, ist diese Rolle kontraproduktiv, denn die Roboter würden sich gegenseitig behindern und zusammenstoßen.

### 4.3.1 Das frontale Unterstützen des Stürmers

Das Szenario *frontales Unterstützen* startet sobald sich der Roboter in direkter Ballnähe befindet. Sie hat die Aufgabe dem Stürmer den Weg ins gegnerische Tor zu bahnen und den Ball für die Dauer des Angriffs geschützt zu halten. Hierfür nimmt der Roboter eine Position gegenüber dem Stürmer ein und läuft rückwärts vor ihm, während dieser den Ball dribbelt. Wie die Abbildung 4.17 zeigt, wird der Ball auf diese Weise zwischen den beiden Robotern gefangen und Richtung gegnerisches Tor bewegt. In der Abbildung 4.19 ist die Funktionsweise des implementierten Szenarios dargestellt.

Der Roboter beobachtet den Ball, das eigene und auch das gegnerische Tor. Da keine

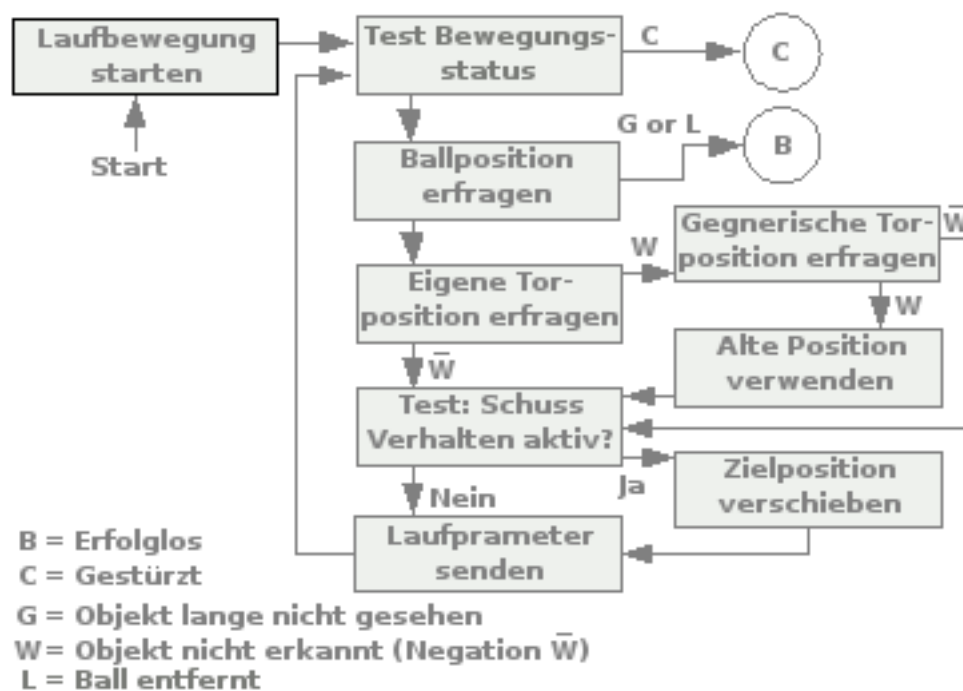


Abbildung 4.19: Das Funktionsdiagramm des Szenarios: frontales Unterstützen.

globalen Daten zur Verfügung stehen, dienen alleine die, aus dem Bild erhaltenen Torkoordinaten, zur Bestimmung der Orientierung. Durch seine Ausrichtung besitzt der Roboter direkten Blickkontakt auf den Ball und das eigene Tor. Es wird eine feste Zielposition angegeben, die sich horizontal genau vor dem Ball und vertikal in einem experimentell erfassten Bereich befindet. Dieser Bereich wird möglichst eng gewählt, so dass der Stür-



verwaltet.

Funktionen gewährleisten indirekt den Zugriff auf die Daten des Objektes. Hierzu gehört die Methode *whoHasBehavior( thisBehaviorID )*. Sie bekommt als Eingabe die ID eines existierenden Verhaltensszenarios und untersucht die Team-Liste nach dem Roboter, welcher das angegebene Szenario ausführt. Kann er diesen ausfindig machen, so wird dessen ID als Rückgabewert übermittelt. In dem Fall, dass die Suche erfolglos ist, wird der Rückgabewert  $-1$  zurückgegeben. Analog zu dieser Funktion ist auch *whoHasRole( thisRoleID )* umgesetzt worden.

Eine Strategie hat eine endliche Menge an Rollen zur Verfügung, die sie unter den drei spielenden Robotern verteilt. Die Spielsituation wird sequenziell analysiert, um eine passende Rollenverteilung zu berechnen. Soll einem Spieler eine Rolle zugewiesen werden, die er bereits ausführt, so verfällt die Zuweisung.

Eine Grundregel bei der Entwicklung ist es, dass ein Stürmer immer auf dem Spielfeld vorhanden sein muss. Dies dient als eine Absicherung dafür, dass immer ein Roboter, für die Leitung eines Angriffes, zur Verfügung steht.

Wenn mehrere Roboter gestürzt sind, hat die Eigenschaft der Strategie, die Rollen zwischen den Spielern zu tauschen, Probleme verursacht. Die Strategie hat durchgehend neue Rollen vergeben, so dass das Verhalten *Aufstehen* unterbrochen worden ist. Dadurch ist es keinem Roboter möglich aufzustehen, so entsteht ein Deadlock. Um dies zu verhindern wird das Aufstehverhalten gesondert behandelt. Ein Strategieszenario darf keine Entscheidung für die gestürzten Roboter treffen, solange er nicht erfolgreich aufgestanden ist.

#### 4.4.1 Ein Stürmer und zwei Flügelspieler

Die Strategie *ein Stürmer und zwei Flügelspieler* hat sich im Laufe der Wettbewerbe als sehr effektiv erwiesen. Es handelt sich hierbei um ein Konzept, bei dem alle Roboter für das offensive Spiel, eingesetzt werden. Dem Szenario wird die Menge von zwei Rollen zur Verfügung gestellt: *der Stürmer* und *der Flügelspieler*.

Ziel dieser Strategie ist, dass sich ein Flügelspieler rechts und einer links vom Stürmer ausrichtet. So kann der Ball von zwei Seiten aus, gesichert werden. Zusätzlich bildet die Aufstellung der Roboter eine Front, die die eigene Spielfeldhälfte verdeckt. Diese Aufstellung ist in der Abbildung 4.16 dargestellt.

Falls eins der Tore nicht ausreichend zu sehen ist, werden einige Roboter stark behindert. Auch die älteren FUMANOIDS Roboter hatten in diesem Fall, mit schweren Orientierungsproblemen zu kämpfen. Dies kann zum vollständigen Verlust der Orientierung führen.

Zum Spielbeginn werden alle drei Spieler mit der Rolle *Stürmer* gestartet. Auf diese Weise haben alle die gleiche Chance, schnell den Ball, zu erreichen. Der Roboter, der als erster am Ball ankommt, darf den Angriff als Stürmer anführen. Die anderen Beiden erhalten die Rolle *Flügelspieler* und teilen sich rechts bzw. links vom Stürmer auf. In dem Fall das mehrere Spieler genau zur gleichen Zeit am Ball ankommen, ist die Priorität des Roboters entscheidend. Je höher die persönliche Roboter ID ist, umso höher ist die Priorität des Roboters.

Die Unterscheidung bezüglich der Ballnähe wird indirekt durch die Aktivierung der Ver-

haltensszenarien gehandhabt. Befindet sich ein Spieler in Ballnähe, so sendet die Rolle *Stürmer* das ID-Signal *Gehe hinter den Ball* aus. Die, als nächstes zu erwartenden Signale, auf der Verhaltensebene sind *Dribbeln* und *Schießen*. Entfernt sich der Roboter aus der Ballnähe, so ändert sich das ID-Signal auf *Gehe zum Ball*. Anhand der gesendeten ID-Informationen können diese Zustände identifiziert werden.

Es ist versucht worden die Entscheidungen, anhand der relativen Balldistanz, zu treffen. Diverse Tests bestätigen, dass die Verwendung dieser Information zu Schwingungen in dem Szenario führt, siehe hierzu Abschnitt 3.1.3. Im Vergleich dazu bieten die Verhaltensszenarien eine zuverlässige Aussage über den Zustand der Roboter.

Die Bedingungen über die Ballnähe werden sequenziell überprüft, um eine Neuverteilung der Rollen zu ermöglichen. Eine Änderung wird in folgenden Situationen eingeleitet:

- Der Ball liegt, weit von dem ausgewählten Stürmer, entfernt.
- Der Stürmer ist unfähig den Angriff fortzusetzen.

Indem kein Roboter des Teams eins der oben aufgezählten ID-Signale bezüglich der Ballnähe sendet, lassen sich beide Fälle identifizieren. Tritt dieser Fall ein, so werden beide Flügelspieler auf die Rolle *Stürmer* geschaltet und zum Ball geschickt. Ab diesem Zeitpunkt wiederholt sich die Entscheidungsfindung zwischen diesen beiden Robotern. In der Abbildung 4.20 wird der Entscheidungsbaum der Strategie dargestellt. Ist der dritte

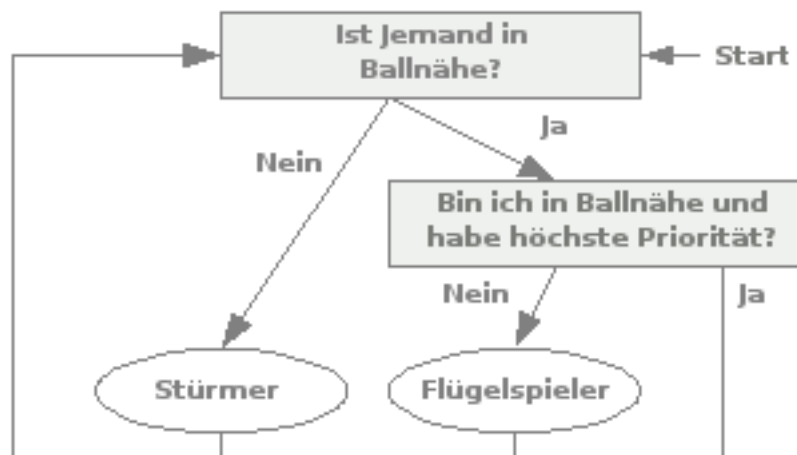


Abbildung 4.20: Der Entscheidungsbaum der Strategie: ein Stürmer und zwei Flügelspieler.

Roboter wieder spielfähig, so ist er mit den anderen Spielern bezüglich der Rollenverteilung gleichgestellt. Es ist möglich, dass die Rolle des *Stürmers* und die eines *Flügelspielers* bereits vergeben sind. In diesem Fall wird dem, ins Spiel zurückkehrenden Roboter, die Rolle des *Flügelspielers* auf der freien Seite zugewiesen.

#### 4.4.2 Ein Stürmer, ein Flügelspieler und ein Unterstützer

Für die Strategie *Ein Stürmer, ein Flügelspieler und ein Unterstützer* wird die Menge der nutzbaren Rollen, um die des *Unterstützers*, erweitert. Die höchste Priorität in der Rollenverteilung hat der *Stürmer*, danach folgt der *Flügelspieler* und erst zum Schluss der *Unterstützer*. Jede Rolle wird in dem Szenario genau einmal vergeben. Folglich kann sich der Flügelspieler auf die, von ihm gewünschte Seite, stellen. In der Abbildung 4.17 ist die resultierende Aufstellung der Roboter dargestellt.

Durch den Einsatz des Unterstützers wird die Strategie offensiver als die oben vorgestellte, siehe Abschnitt 4.4.1. Der Ball wird von dem Stürmer und dem Unterstützer abgeschirmt, so dass nur noch ein seitlicher Zugang möglich ist. Zusätzlich, sichert der dritte Spieler eine der beiden Seitenwege. Innerhalb dieser Formation wird der Ball in Richtung des gegnerischen Tores bewegt.

Die drei Roboter starten mit der Rolle *Stürmer*. Abhängig davon in welcher Reihenfolge sie an den Ball kommen, werden die Rollen mit fallender Priorität vergeben. Treffen zwei Spieler zur selben Zeit ein, so gilt auch hier die Priorität der höheren Roboter ID. Die Funktionsweise des Szenarios ist in der Abbildung 4.21 dargestellt. In dem Fall, dass

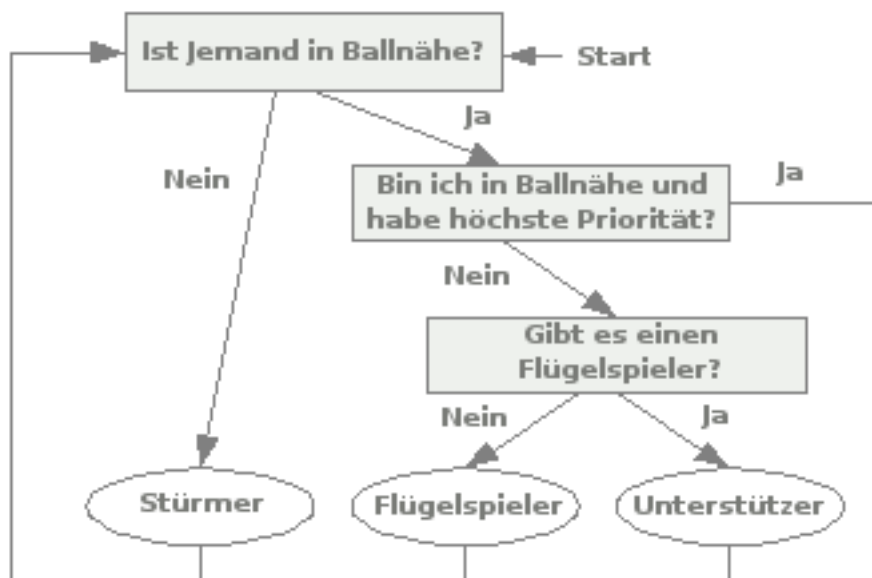


Abbildung 4.21: Der Entscheidungsbaum der Strategie: Stürmer, Flügelspieler und Unterstützer.

der Stürmer den Angriff nicht weiterführen kann, werden alle anderen Roboter zum Ball geschickt. Sie bekommen hierfür, die Rolle *Stürmer* zugewiesen. Durch diese Zuweisung erhöht sich die Wahrscheinlichkeit, dass einer der beiden Spieler schnellstmöglich den Ball erreicht.

### 4.4.3 Die Strategie mit Torwart

In den Regeln der Liga Humanoid KidSize wird der Torwart als ein besonderer Spieler betrachtet. Für ihn gelten im eigenen Strafraum gesonderte Regeln. Beispielsweise darf kein Roboter den Torwart des gegnerischen Teams, in dessen Strafraum, berühren. Weiterhin ist es dem Torwart erlaubt innerhalb dieser Zone den Ball mit den Händen zu berühren. Für alle anderen Spieler ist dies ein Handspiel.

In den Strategien der FUmoids sind bislang alle Roboter gleichberechtigt bei der Rollenzuweisung behandelt worden. Die Rolle *Torwart* darf jedoch nur einem Roboter, für den gesamten Spielverlauf, zugewiesen werden. Dieser Spieler wird vor dem Spielbeginn als Torwart gekennzeichnet. So weiß der Schiedsrichter für welchen Roboter, die gesonderten Regeln gelten.

Die Szenarien der Strategieebene sind diesen Regeln angepasst worden, damit eine Torwartrolle verwendet werden kann. Hierfür ist die Variable *goalKeeper* eingeführt worden, in der die Roboter ID des Torwartroboters gespeichert wird. Beim Starten der Roboter über das Netzwerk, wird ein Roboter aus dem Team als Torwart markiert.

Die Strategie kann anhand dieser ID, den Roboter identifizieren, welchem die Torwartrolle zugewiesen werden soll. Dieser Spieler wird von allen folgenden Rollenwechseln ausgeschlossen.

Jedes Szenario der Strategieebene kann durch die Torwartrolle erweitert werden. Dafür

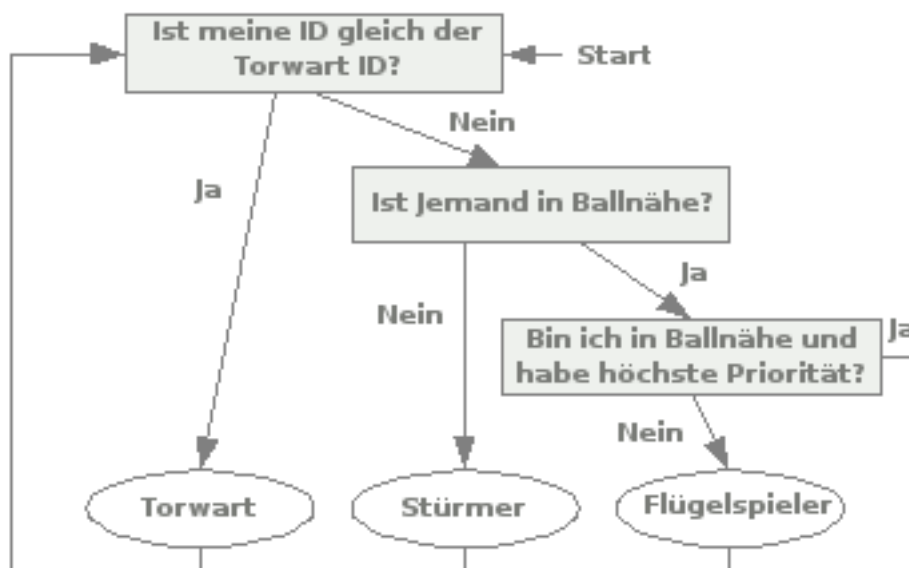


Abbildung 4.22: Der Entscheidungsbaum der Strategie aus Abschnitt 4.4.1 mit hinzugefügter Torwartrolle.

wird ein neuer Entscheidungsbaum mit zwei Teilbäumen erzeugt. Der eine Teilbaum enthält den unveränderten Entscheidungsbaum der alten Strategie. Diesen Entscheidungsweg durchlaufen alle Roboter, die nicht als Torwart ausgezeichnet worden sind. Der zweite

Teilbaum ist alleine für den Torwart reserviert. Durch diese Lösung kann dieselbe Implementierung des Szenarios sowohl für ein Spiel mit, als auch ohne Torwart eingesetzt werden. In der Abbildung 4.22 ist ein solcher Entscheidungsbaum dargestellt.

## 4.5 Die globale Planung

Erst bei der Robotergeneration 2009 ist es, für die FUmanoids möglich gewesen, eine Selbstlokalisierung zu realisieren. Durch diese Softwarekomponente bekommen die Roboter die Fähigkeit, ihre absolute Position auf dem Feld, zu bestimmen. Die früheren Versuche einer Umsetzung scheiterten an den geringen Hardwareressourcen.

Anhand eines internen Weltmodells kann der Roboter durchgehend Bezug, auf die Objekte nehmen, die fester Bestandteil des Spielfeldes sind. Hierzu gehört z.B. das gegnerische Tor, das die Zielorientierung des Roboters in vielen Verhaltensszenarien, darstellt.

Die älteren Roboter benötigen eine direkte Sicht auf alle Objekte, die ihre Bewegung beeinflussen. Hierfür muss der Kopf oft bewegt werden. Beispielsweise wird im Verhalten *Dribbeln* und *Schießen* der Kopf wiederholt hoch und runter bewegt. Anders ist es nicht möglich, ausreichende Informationen über die Ball- und Torposition zu gewinnen. Ohne aktuelle Daten kann die Laufrichtung nicht korrigiert werden und der Spieler kommt von seinem Weg ab.

Die Kopfbewegungen haben mehrere Nachteile: sie verzögern die Erfüllung der Aufgabe und erhöhen die Gefahr eines Ballverlustes. Anhand der Kenntnis über die eigene Position können diese Bewegungen, bei den neuen Robotern, vollständig vernachlässigt werden. Zusätzlich kann auch die absolute Position von Objekten berechnet werden, die im Blickfeld des einzelnen Roboters liegen.

Die Information der globalen Position, der im Bild erkannten Objekte, wird über das Netzwerk dem Rest des Teams mitgeteilt. Auf diese Weise erlangt ein Roboter Kenntnis über die Position aller Gegenstände, die andere Spieler zu einem bestimmten Zeitpunkt, sehen.

Die FUmanoids nutzen diesen Datenaustausch für die Ballsuche. Es ist ausreichend wenn ein Spieler durch seine direkte Sicht auf den Ball, seine globale Position berechnet und an die Anderen berichtet. Spieler, die im Verhalten *Suche den Ball*, siehe Abschnitt 4.1.2 sind, können nun direkt zu dieser Position laufen. Dank diesem Verfahren werden Ballverluste stark minimiert.

Vollständig kann sich die Planung noch nicht auf die globalen Informationen verlassen. Der Grund dafür, sind schwankende Werte, die besonders im Nahbereich zu Objekten, gefährlich werden können. Zum aktuellen Zeitpunkt werden globale Informationen erst dann betrachtet, wenn benötigte Objekte nicht im Sichtfeld des Roboters liegen.

## 5 Zusammenfassung und Ausblick

Das, in dieser Arbeit beschriebene Planungssystem ist erfolgreich realisiert worden. Es ist in mehreren Wettkämpfen eingesetzt worden, auch in den Spielen beim Robocup 2009. Hierbei haben die FUManoids den Titel des Vizeweltmeisters gewonnen.

Die Teilnahme an mehreren Turnieren, wie das Iran Open und German Open, haben eine gute Gelegenheit, für das Testen des Systems, geboten. Es ist eine Vielzahl von Fehlerquellen entdeckt und behoben worden bis der heutige Zustand erreicht werden konnte. Dennoch sind einige Probleme offen geblieben, deren Lösung zukünftig in Erwägung gezogen werden sollte.

Eines davon, bezieht sich auf den Einsatz von statischen Bewegungen. Ihre Aufnahme erfordert ein gutes Verständnis der mechanischen Konstruktion und ihrer kinetischen Eigenschaften. Sie müssen bei jedem Roboter einzeln angepasst werden und verlieren schnell ihre Gültigkeit, direkt proportional zur Abnutzung der Aktoren. Es empfiehlt sich zukünftig nur dynamische Bewegungen einzusetzen. Zwar ist die Entwicklungsdauer länger als bei der statischen Variante, dafür aber fällt die Wartungszeit erheblich kürzer aus.

Eine weitere Optimierung des Planungssystems ist durch die Nutzung des Statuspfades, siehe Abschnitt 3.3.2, zwischen der Rollen- und Strategieebene gegeben. Die Statusinformationen dieser Schicht können dazu genutzt werden, eine Beurteilung über die Spielfähigkeit bzw. Spielunfähigkeit des Roboters abzugeben. Beispielsweise könnte die Kamera des Roboters durch einen Sturz beschädigt werden, so dass das Verhalten *Suche den Ball*, siehe Abschnitt 4.1.2, ständig fehlschlägt. Das Szenario auf der Rollenebene kann erkennen, wenn ein bestimmtes Verhalten wiederholt erfolglos beendet wird. In diesem Fall sendet es eine negative Beurteilung an die höhere Ebene. Das Strategieszzenario besitzt so, die Möglichkeit diesen Roboter zu stoppen und den menschlichen Mitgliedern des Teams ein Alarmsignal zu senden. Diese können den Roboter vom Feld entfernen und den Schaden beheben.

Der Einsatz von Ereignissen stellt eine weitere Möglichkeit zur Verbesserung des Planungssystems dar. Derzeit erfragt jedes Szenario eigenständig, die von ihm benötigte Information. Beispielsweise ist in jedem Verhaltensszenario eine Abfrage integriert, ob der Roboter gestürzt ist. Diese Information erhält sie von der Bewegungsebene und wird an die Rollenebene weitergeleitet. Die Daten durchlaufen annähernd die gesamte Hierarchie, bevor die Aufstehbewegung aktiviert wird. Es wäre sinnvoll, diese und ähnliche Informationen als ein Ereignis zu kapseln. Auf diese Weise kann die Reaktionszeit verkürzt und die Szenarien vereinfacht werden.

Im vorgestellten Planungssystem trifft der einzelne Roboter die Entscheidungen über seinen nächsten Handlungsschritt. Ist eine Entscheidung gefallen, so teilt er sie über das Netzwerk den anderen Spielern mit. Hierbei können Verzögerungen und Konflikte entste-

hen, da jeder einzelne die Daten der Anderen für seine Berechnungen benötigt. Abhilfe könnte eine zentrale Entscheidungseinheit schaffen, welche jedem Roboter eine Rolle zuweist. Hierfür könnte ein Spieler zum Kapitän ernannt werden. Um die Präsenz eines Kapitäns im Spiel zu gewährleisten, wird dieser dynamisch gewählt. So wäre es möglich, dass jeder Roboter die Rollenverteilung für alle anderen Spieler berechnet und sie versendet. In diesem Fall übernehmen die einzelnen Spieler die Planung des Roboters, mit der niedrigsten Roboter ID. Sollte der Kapitän während eines Spiels ausfallen, so würde automatisch die Planung des Roboters, mit der nächst niedrigsten ID, gewählt werden.

Im kommenden Jahr soll das Roboterteam der Freien Universität Berlin erneut, am Robocup in der Humanoid KidSize Liga teilnehmen. Es soll nicht nur der aktuelle Titel verteidigt, sondern auch der Aufstieg zum Weltmeister angestrebt werden. Die aktuelle Projektplanung sieht auch beim Robocup 2010 den Einsatz, des hier vorgestellten Planungssystems, vor. Hierfür ist die Weiterentwicklung einzelner Szenarien und das Integrieren neuer Features vorgesehen. Weiterhin sind für das kommende Jahr neue Hardwarekomponenten wie ein Gyroskop, Drucksensoren und der Umstieg auf eine neue Kamera geplant. Diese Änderungen wirken sich auch auf die Planungskomponente aus, so dass Anpassungen auf dieser Ebene nötig sein werden.

Der Einsatz eines Gyroskops ermöglicht die Messung der Neigung des Roboters. Diese Information kann genutzt werden um die Laufbewegung zu verbessern. Analog verhält es sich mit den Drucksensoren, die die Fußsensoren des Roboters ersetzen und das Erkennen eines Sturzes beschleunigen, sollen.

# Literaturverzeichnis

- [1] R. C. Arkin: *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments*, Dissertation at University of Massachusetts Amherst. Dept. of Computer and Information Science, publisher Georgia Institute of Technology 1987.
- [2] R. C. Arkin: *Behavior-Based Robotics* with a foreword by Michael Arbib, Intelligent Robots and Autonomous Agents series, MIT Press, Cambridge, 1998; 491 pp, ISBN 0-262-01165-4.
- [3] R. C. Arkin and T. Balch: *AuRA: Principles and Practice in Review*, Journal of Experimental and Theoretical Artificial Intelligence 1997, volume 9, pages 175–189.
- [4] T. Balch: *Clay: Integrating Motor Schemas and Reinforcement Learning*, Coll. of comp. tech. report, Ga. Inst. of Tech, 1997.
- [5] S. Behnke, B. Frötschl, R. Rojas: *Using hierarchical dynamical systems to control reactive behavior*. International Joint Conference on Artificial Intelligence, The Third International Workshop on RoboCup, 1999.
- [6] R. A. Brooks: *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation, 1986.
- [7] J. F. Canny: *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award 1987, MIT Press, Cambridge.
- [8] D. W. Eccles, P. T. Groth: *Wolves, bees, and football: Enhancing coordination in sociotechnological problem solving systems through the study of human and animal groups*, June 2006.
- [9] B. Fischer: *Hard- und Software-Entwicklung für ein Stereo-Vision-System in eingebetteter Linux-Umgebung*, Diplomarbeit an der Technischen Universität Berlin 2009.
- [10] R. Guilboud: *Kamerabasierte Selbstlokalisierung humanoider Roboter in der RoboCup Umgebung* Diplomarbeit an der Freien Universität Berlin, Institut für Informatik, 2008.
- [11] H. Jaeger, T. Christaller: *Dual dynamics: Designing behavior systems for autonomous robots*. In S. Fujimura and M. Sugisaka, editors, Proceedings International Symposium on Artificial Life and Robotics, Beppu, Japan, 1997.

- [12] T. H. Labella, M. Dorigo: *Division of Labor in a Group of Robots Inspired by Ants? Foraging Behavior*. ACM Transactions on Autonomous and Adaptive Systems, Vol. 1, No. 1, September 2006.
- [13] T. Langner: *Selbstlokalisierung für humanoide Fußballroboter mittels Mono- und Stereovision* Diplomarbeit an der Freien Universität Berlin, Institut für Informatik, 2009.
- [14] K. H. Low, W. K. Leow, M. H. Ang, Jr.: *A Hybrid Mobile Robot Architecture with Integrated Planning and Control*. Bologna, Italy 2002.
- [15] A. K. Mackworth: *On seeing robots*. Technical Report TR-93-05, Department of Computer Science, University of British Columbia, 1993.
- [16] T. Minato, H. Ishiguro: *Construction and Evaluation of a Model of Natural Human Motion based on Motion Diversity*. March 2008, Amsterdam, The Netherlands.
- [17] H. R. Moballegh: *Concurrent Scenario-Based Planning, a New Approach to Implement Highly Cooperated Multi-Agent Systems* unveröffentlicht.
- [18] H. R. Moballegh, R. Guilbourd, G. Hohl, M. Oertel, R. Rojas: *FUmanoid Team Description 2008*, Institut für Informatik, Arbeitsgruppe Künstliche Intelligenz, Freie Universität Berlin.
- [19] H. R. Moballegh, G. Hohl, T. Landgraf, B. Fischer, T. Langner, T. Fassbender, S. Otte, K. Stoll, A. Tuchscherer, D. Steig, S. Heinrich, S. Mielke, D. Seifert, M. Kukulski, B. Kahlert, R. Rojas: *FUmanoid Team Description 2009*, Institut für Informatik, Arbeitsgruppe Künstliche Intelligenz, Freie Universität Berlin.
- [20] H. R. Moballegh, M. Mohajer, R. Rojas: *Increasing foot clearance in biped walking: Independence of body vibration amplitude from foot clearance*. Freie Universität Berlin, Institut für Informatik, 2008.
- [21] M. Oertel: *Aufbau und Implementierung einer Multithreading Technologie in einer 8 Bit RISC-Architektur Umgebung*. Bachelorarbeit an der Freien Universität Berlin, Institut für Informatik, 2008.
- [22] B. Scassellati: *Theory of Mind for a Humanoid Robot* 2000.
- [23] D. Seifert: *Portierung der FUmanoid-Software*. Studienarbeit an der Freien Universität Berlin, Institut für Informatik, 2009.
- [24] F. Yamaoka, T. Kanda, H. Ishiguro, N. Hagita: *Developing a Model of Robot Behavior to Identify and Appropriately Respond to Implicit Attention-Shifting*. March 2009, La Jolla, California, USA.
- [25] Encyclopedia Britannica <http://www.britannica.com/EBchecked/topic/505818/robot>
- [26] HaViMo [http://robosavvy.com/store/product\\_info.php/products\\_id/349](http://robosavvy.com/store/product_info.php/products_id/349)

- [27] Gumstix <http://www.gumstix.com/>
- [28] Die Lensation GMBH <http://www.lensation.de/>
- [29] OpenEmbedded [http://wiki.openembedded.net/index.php/Main\\_Page](http://wiki.openembedded.net/index.php/Main_Page)
- [30] RoboCup <http://www.robocup.org>
- [31] RoboCup Soccer Humanoid League Rules 2008:  
<http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2008.pdf>  
und 2009:  
<http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2009.pdf>
- [32] Robotis inc. <http://www.robotis.com/zbxe/main>