# Steuerung und Kontrolle von Omnidirektionalen Fußballrobotern

Diplomarbeit der Informatik

vorgelegt von

**Fabian Wiesel**

geboren 11. April, 1978, in Berlin
wiesel@inf.fu-berlin.de
Matrikelnummer: 3514090

am 31. Januar 2006

**Institut für Informatik**
**Freie Universität Berlin**
**Takustr. 9**
**14159 Berlin**

Betreuer: Prof. Dr. Raúl Rojas
Dr. Ing. Achim Liers

## Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Berlin, den 31. Januar 2006

Fabian Wiesel

## Zusammenfassung

Diese Diplomarbeit beschreibt die Steuerungssoftware der Fußballroboter der Freien Universität Berlin, FU-Fighters, welche sowohl in der Small-Size Liga, als auch in der Middle-Size Liga der RoboCup-Initiative antreten. Fußball wird als Testfeld der Künstlichen Intelligenz genutzt, um Lösungen für Agenten zu finden, die kooperativ und kompetitiv mit ungenauen Daten in einer sich ständig ändernden Welt Aufgaben lösen.

Die Steuerungssoftware teilt sich in zwei Komponenten:

a) Die Software auf der Elektronik. Diese dient der Ansteuerung der Motoren und sonstiger Aktoren eines Omniroboters.

b) Ein Programmierrahmen für unserer Multiagentensystem.

Die Elektronik auf dem Roboter setzt die Kommandos des Programmierrahmens in physische Aktionen um. Sie regelt dazu die Bewegung mittels eines PID-Controllers. Desweiteren sichert sie mittels einfacher Logiken die Hardware vor Beschädigung durch falsche Kommandos.

Der Programmierrahmen orientiert sich am verhaltensbasierten Ansatz. Dies bedeutet, dass die verschiedenen Tätigkeiten des Roboters in distinkte Verhaltensweisen unterteilt werden, die jeweils eine eigenes Ziel verfolgen. Durch Interaktion mit der Umgebung und durch das Zusammenspiel der Verhaltensmuster können Probleme in einer sich ständig ändernden Welt gelöst werden.

Die Implementierung orientiert sich an der Subsumption-Architektur, welche die Verhaltensmuster in Module unterteilt und dem Dual-Dynamics Ansatz, welche die Verhaltensmuster mit einer Aktivierungs- und einer Zieldynamik ausstattet.

Die Abhängigkeiten der Module werden in unserem Rahmen explizit durch Konnektoren repräsentiert, wodurch eine parallelisierte Ausführung ermöglicht wird. Die beschriebene Steuersoftware wurde bei der Weltmeisterschaft im Roboterfußball in Osaka, Juli 2005 sowohl für die Small-Size- als auch für die Middle-Size-Roboter eingesetzt.

# Inhaltsverzeichnis


**1 Einführung** **7**
- 1.1 Aufbau der Arbeit . . . 7
- 1.2 Roboterfußball als neue Herausforderung an die KI . . . 7
- 1.3 Die RoboCup-Initiative . . . 8

**2 Grundlagen der reaktiven Steuerung** **11**
- 2.1 Reaktive und Planende Steuerung . . . 11
- 2.2 Reaktive Architekturen . . . 12
  - 2.2.1 Subsumption Architecture . . . 12
  - 2.2.2 Dual-Dynamics Ansatz . . . 13
    - 2.2.2.1 Beispiel . . . 14
  - 2.2.3 Hierarchische Reaktive Verhaltensmuster . . . 15
    - 2.2.3.1 Beispiel . . . 17
- 2.3 Hybride Architekturen . . . 18
  - 2.3.1 Beispiel . . . 18

**3 Low-Level Steuerung** **21**
- 3.1 Regelung des Roboters . . . 22
  - 3.1.1 Messen der Bewegung . . . 22
  - 3.1.2 Beeinflussung der Bewegung . . . 24
  - 3.1.3 Regelung der Bewegung . . . 27
- 3.2 Programmaufbau der Elektroniksoftware . . . 28

**4 High-Level Steuerung** **31**
- 4.1 Zielsetzung . . . 31
- 4.2 Aufbau des alten Verhaltensrahmen . . . 31
- 4.3 Probleme des alten Rahmens . . . 36
- 4.4 Graph von Sensoren als neuer Rahmen . . . 37
- 4.5 Datenabhängigkeitsgraph und Sequentialisierung . . . 39
- 4.6 Verhaltensmuster als Sensoren . . . 40
- 4.7 Aktoren . . . 41
- 4.8 Weitere Arten von Sensoren . . . 42
- 4.9 Konnektoren und Umgebungen . . . 43
- 4.10 Funktionsgruppen . . . 44
  - 4.10.1 Beispiel . . . 44
- 4.11 Graphische Benutzeroberfläche . . . 45

4.12 Nichtsequentielle Ausführung des Verhaltenrahmens . . . . . 47

**5 Fazit und Ausblick** **51**

**Literaturverzeichnis** **53**

# 1 Einführung

## 1.1 Aufbau der Arbeit

Dieses Kapitel bespricht die Motivation für die Entwicklung von fußballspielenden Robotern. In Kapitel 2 gebe ich einen kurzen Überblick über den reaktiven Ansatz zur Steuerung von Agenten und vergleiche ihn mit dem planenden Ansatz. Kapitel 3 behandelt die Software der Elektronik, die zur Ansteuerung des Robotikteams der FU-Berlin, den FU-Fighters, genutzt wird. Kapitel 4 erläutert den Aufbau und Entwicklung eines neuen Verhaltensrahmen für unsere Roboter. Im abschließenden Kapitel fasse ich die Ergebnisse zusammen, und gebe einen Ausblick auf weitere Entwicklungsmöglichkeiten.

## 1.2 Roboterfußball als neue Herausforderung an die KI

Schon früh wurden Spiele in der Künstlichen-Intelligenz-Forschung als Herausforderung gewählt. Spiele bieten eine klar definierte Umgebung mit einem kodifizierten Regelwerk. Dadurch wird die Komplexität des Problems beschränkt. Außerdem wird mittels Sieg und Niederlage oder durch ein Punktesystem ein objektives Maß für die relative Güte der Spieler festgelegt.

Das Schachspiel genießt in der westlichen Welt ein hohes Maß an Ansehen, was die Anforderungen an den Intellekt betrifft. So ist es nicht verwunderlich, dass schon früh und für lange Zeit das Augenmerk sich auf die Erforschung von Schachstrategien richtete[19]. Mit dem Sieg von Deep Blue über Kasparov 1997, jedoch, wurde das Wettrennen um den ersten Sieg eines Schachcomputers über einen menschlichen Schachweltmeister abgeschlossen, wodurch sich sich die damit verbundene Aufmerksamkeit der Öffentlichkeit gelegt hat.

Andere Forscher halten Schach an für sich ungeeignet, um neue Erkenntnisse der Kognitionsprozesse des Menschen zu erlangen, da das Problem zu abstrakt ist [4]. Die Intelligenz liege nicht in der bloßen Leistung, Schach zu spielen, sondern vielmehr in der Fähigkeit, selbständig aus den komplexen Wahrnehmungen das abstrakte Konzept des Spiels zu erkennen und zu begreifen. Man solle daher eher an Problemen arbeiten, die sich in der physischen Welt befinden.

Alan Mackworth schlug 1993 Fußball als Benchmark für die Künstliche Intelligenz vor [17], denn es enthält gegenüber Schach folgende weiteren Herausforderungen:

**Mehrere Agenten** Die Agenten eines Teams müssen miteinander kooperieren und gegen andere Agenten konkurrieren.

**Ständig wechselnde Umgebung** Die Welt ändert sich auch ohne Aktionen des Agenten.

**Unvollständiges Wissen** Der Agent hat eine partielle Sicht der Welt.

**Fehlerbehaftetes Wissen** Die Positionen aller Objekte sind nur mit begrenzter Genauigkeit erfasst.

**Fehlerhaftes Wissen** Einzelne Wahrnehmungen können falsch sein.

**Nichtdeterministische Welt** Die Effekte einer Aktion sind nicht vollständig vorhersagbar.

**Kontinuierliche Welt** Zeit, Raum und Aktionen sind kontinuierlich und Ereignisse finden simultan statt.

## 1.3 Die RoboCup-Initiative

Die RoboCup-Initiative wurde gegründet, um durch internationalen Wettbewerb in standardisierten Problemen die Forschung zu fördern, zu evaluieren und auch einer nicht-wissenschaftlichen Öffentlichkeit zugänglich zu machen [14]. Seit 1997 finden

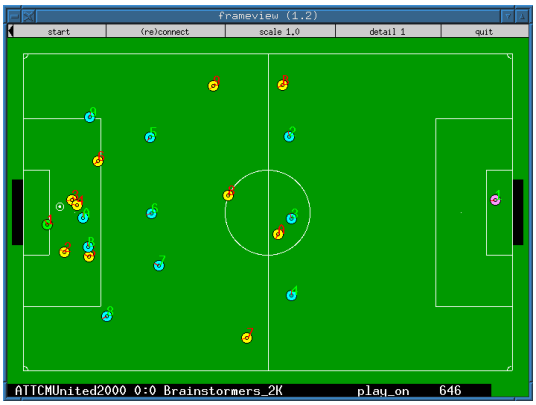


***Abbildung 1.1:*** Simulation-Liga: ATTCMUnited2000 gegen Karlsruhe Brainstormers, Melbourne 2000

jährlich an wechselnden Orten Weltmeisterschaften in verschiedenen Ligen statt. Die erste RoboCup Weltmeisterschaft wurde 1997 mit 41 Teams in drei Ligen bestritten.

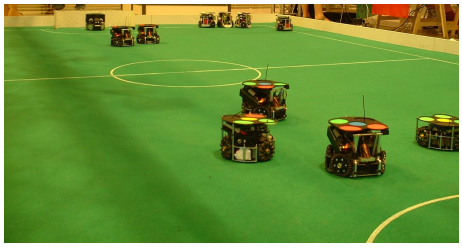

***Abbildung 1.2:*** Roboter der Small-Size Liga (vorne FU-Fighters), Osaka 2005

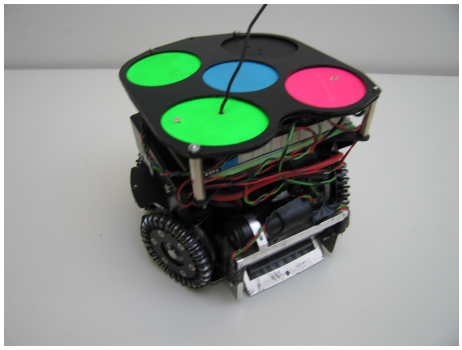

***Abbildung 1.3:*** Small-Size Roboter der FU-Fighters, (2005, 2006)

Die Ligen sind:

- Simulation: 11 unabhängige Agenten pro Team spielen in einer Simulation auf einem Rechner gegeneinander Fußball.
- Small-Size: Fünf Roboter pro Team mit einem maximalen Durchmesser von 18 cm spielen mit einem orangefarbenen Golfball. Jedes Team kontrolliert die Roboter mittels eines zentralen Rechners, der an eine über dem Feld angebrachten Kamera angeschlossen ist. 1997 war das Feld 2,7 mal 1,5 Meter groß. Über die Jahre ist die Feldgröße auf 4,9 mal 3,4 Meter erweitert worden.
- Middle-Size: Die Roboter Middle-Size Liga sind mit eigenen Rechnern und lokaler Wahrnehmung ausgestattet und spielen mit einem orangefarbenen Fußball. 1997 konkurrierten noch vier Roboter mit bis 50 cm Durchmesser in jedem Team auf einem 9 mal 6 Meter großen Feld. Bei der letzten Weltmeisterschaft war die Feldgröße 12 mal 8 Meter und es konnten bis zu sechs Roboter pro Team am Spiel teilnehmen.

M

Durch den entstandenen Wettbewerb zwischen den Teams verbesserten sich die Fähigkeiten der Roboter schnell. Dieser Tatsache wird mit einer jährlich zunehmenden

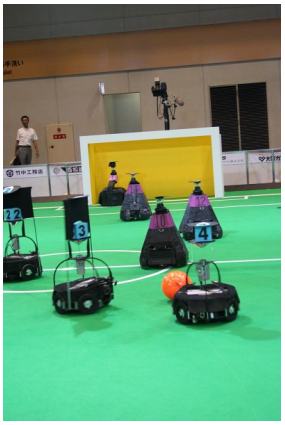

***Abbildung 1.4:*** Wettkampf in der Middle-Size Liga, Osaka 2005

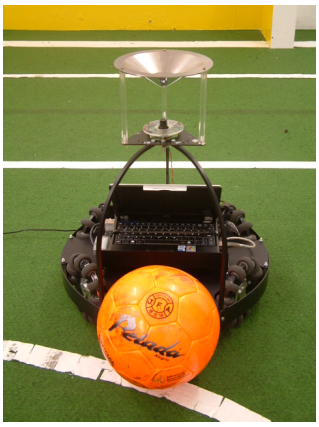

***Abbildung 1.5:*** Middle-Size Roboter der FU-Fighters (2004, 2005)

Erschwerung der Bedingungen Rechnung getragen. So gab es, zum Beispiel, anfangs strenge Anforderungen an die Beleuchtung. Sobald ein Großteil der Teams keine größeren Probleme mit der Beleuchtung hatte, wurden die Anforderungen im darauffolgenden Jahr gelockert. Eine andere Vereinfachung war früher eine Bande am Spielfeldrand. Dadurch konnte der Ball nie ins Aus rollen. Eine bessere Kontrolle des Balls hat dies seit 2002 überflüssig gemacht. Die Verbesserungen in Wahrnehmung und Kontrolle hatten eine graduelle Steigerung der Spielgeschwindigkeit zur Folge, so dass man sich entschied, nach und nach das Spielfeld bis auf die heutigen Dimensionen zu vergrößern.

An der Weltmeisterschaft in Osaka im Jahre 2005 nahmen insgesamt 419 Teams in sieben Ligen teil. Davon verzeichneten die ursprünglichen drei Ligen 111 teilnehmende Teams. Unser Team hat in dieser Meisterschaft in der Small-Size-Liga den ersten und in der Middle-Size-Liga den zweiten Platz belegt. Unsere Team-Beschreibung in der Small-Size ist unter [11] und in der Middle-Size unter [10] nachzulesen.

# 2 Grundlagen der reaktiven Steuerung

## 2.1 Reaktive und Planende Steuerung

Die Steuerung von Robotern lässt sich grob in zwei Ansätze unterteilen: Planende und Reaktive Systeme. Die Ersten bauen auf Planalgorithmen und einer symbolischen Repräsentation der Welt auf. Planalgorithmen suchen eine Folge von symbolischen Aktionen, die in der Repräsentation des Problems einen Lösungsweg darstellen. Diese Folge von Aktionen wird anschließend in der Wirklichkeit umgesetzt. Dieser suchbasierte Ansatz wurde in der Künstlichen Intelligenz zuerst verfolgt und hat beim Schachspiel zu den bekannten Erfolgen geführt.

Das Planen von Bewegungen in einem nicht diskreten Raum ist jedoch recht komplex [6], [7]. Um dieser Komplexität Herr zu werden, wird das Problem approximiert. Dies kann durch Diskretisierung des Raums und der Aktionen geschehen. Eine andere Methode ist die zufällige Wahl von Aktionen oder Wegpunkten aus dem Raum [15]. Die Suche nach einem Lösungsweg über diese Teilmenge des Raumes und der Aktionen lässt sich verhältnismäßig effizient realisieren. In sich ändernden Umgebungen wird zusätzlich versucht, so große Teile der Planung wie möglich wieder zu verwenden. Jedoch ergibt sich hier das prinzipielle Problem, dass je schneller sich die Umgebung ändert, umso mehr Planung zunichte gemacht wird. Damit wird dementsprechend viel Neuplanung erforderlich. Dies ist aber der Reaktionszeit abträglich, welche gerade in sich schnell ändernden Umgebungen sehr wichtig ist.

Eine andere Herangehensweise ist der von R. Brooks [5] propagierte verhaltensbasierte Ansatz. Das Leitmotiv sind Verhaltensmuster, wie sie in der Verhaltensbiologie untersucht werden. Solche Verhaltensmuster können, wie im Falle von Reflexen, einfachste Zusammenhänge zwischen Reiz und Reaktion sein. Auch wenn Verhaltensmuster meist eine bestimmte Aufgabe zugeordnet werden können („task achieving behaviors"), erfüllen sie nur im Zusammenspiel ihre eigentliche Aufgabe: Das Überleben von Individuum und Art zu sichern.

Analog verhält es sich mit den Verhaltensmustern des reaktiven Ansatzes: Jedes Verhaltensmuster ist einem eigenen Modul zugeordnet und dient der Erfüllung einer bestimmten Teilaufgabe. Es wird aufgrund von Reizen ausgelöst und reagiert auf die Wahrnehmung mit der Beeinflussung von Aktoren. Durch das Zusammenwirken der verschiedenen Module wird ein komplexes Verhalten erreicht, das den unterschiedlichen Situationen in einer sich ständig ändernden Umwelt gerecht wird.

Reaktive Systeme betrachten nicht wie Planende eine zukünftige Abfolge von Aktionen und deren Auswirkungen, sondern reagieren aufgrund der Erfahrungen aus der Vergangenheit auf die aktuelle Situation. Die rechnerische Komplexität nimmt

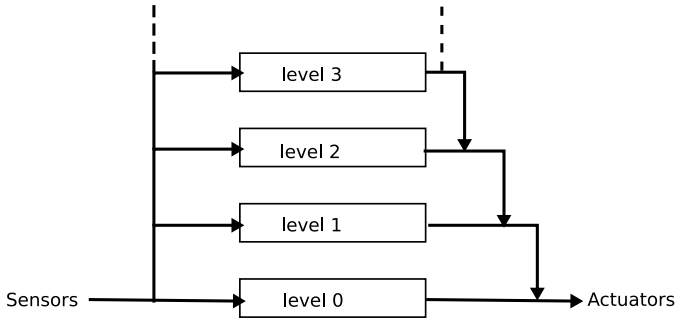


***Abbildung 2.1:*** Hierarchische Struktur von sich ergänzenden Ebenen (nach Brooks [3]). Ebene 0 enthält die simpelsten Verhaltensmuster. Jede weitere Ebene ergänzt die Funktionalität der unteren.

gegenüber planenden Systemen ab, und so fällt auch die Reaktionszeit geringer aus. Reaktive Systeme können aber nicht erkennen, dass eine Aktion, die momentan ungünstig erscheint, längerfristig zum Erfolg führen könnte. Somit ist es nicht immer möglich, mit reaktiven Systemen einen Lösungsweg oder gar den optimalen Lösungsweg zu finden. Rein reaktive oder planende Systeme trifft man daher selten an. Pfadplaner werden in reaktive Systeme eingebaut, um zu verhindern, dass, z.B., ein Roboter in einer Sackgasse landet. Bei planenden Systemen greift man meistens bei der Ansteuerung der Motoren auf Regler zurück, die sich dem reaktiven Ansatz zuordnen lassen, da dort die Reaktionszeit wichtig ist und meist die Rechenleistung begrenzt ist.

## 2.2 Reaktive Architekturen

Für reaktive Verhaltensmuster gibt es verschiedenste Architekturen und Ansätze, die hier nicht alle vorgestellt werden können. Ich bespreche daher nur die Relevantesten. Andere Ansätze umfassen zum Beispiel „Situated Automata“ [13]. Für einen Überblick sei der Leser an „Behavior-Based Robotics“ [1] verwiesen.

### 2.2.1 Subsumption Architecture

Das Gerüst der Subsumption Architektur von Brooks [3] sind einzelne Module. Jedes davon ist ein endlicher Automat, dessen innerer Zustand aufgrund von Eingaben wechselt. Je nach Eingabe und Zustand liefert das Module verschiedene Ausgaben. Die Ausgaben und Eingaben der Module können untereinander oder mit Sensoren oder Aktoren verknüpft werden. Sensoren können zum Beispiel Abstandsmesser oder Kontaktsensoren sein, während beispielsweise die Motoren als Aktoren dienen. Sowohl die Ausgaben von Modulen als auch die Eingaben zu Modulen können von anderen Verhaltensmodulen unterbrochen werden.

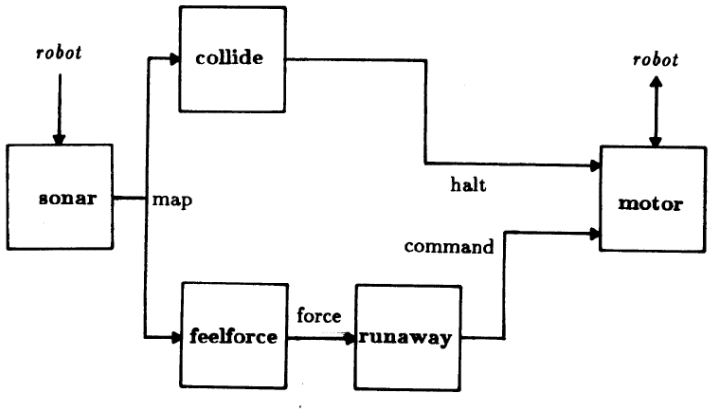


***Abbildung 2.2:*** Beispiel für eine Ebene 0 einer Subsumption Architektur (aus [3]). „Collide" stoppt den Roboter, wenn das Sonar eine zu geringe Entfernung zu einem Hinderniss angibt. „Feelforce" generiert aus den gemessenen Entfernungen eine abstoßende Kraft, welche „Runaway" veranlasst, den Roboter von Hindernissen wegzusteuern.

Bei der Subsumption Architektur werden die Verhaltensmuster auf hierarchische Ebenen verteilt, wobei jede Ebene unabhängig von den höher gelegenden agieren soll (Abbildung 2.1). Die Hierarchie steht für steigende Abstraktion und wachsende Fähigkeiten. Niedrige Ebenen bestehen aus elementaren Verhaltensmodulen, wie zum Beispiel „Notstopp" („Collide"), während sich auf höheren Ebenen komplexere Verhaltensmuster, wie zum Beispiel „Kartographieren", befinden.

Die Ebenen werden nach und nach entwickelt und getestet. Zu Beginn einer neuen Ebene wird der Aufbau der unteren nicht mehr verändert, sondern nur noch durch Hemmung von Ausgängen oder der Unterdrückung von Eingaben und durch Einspeisung anderer Daten beeinflusst. Auf unteren Ebenen stellen Verhaltensmodule eine Grundfunktionalität bereit, die in höheren Ebenen durch verfeinerte Implementierungen ersetzt werden, indem diese die Simpleren durch Unterdrückung aller Eingaben und Ausgaben ausschalten.

Verhaltensmuster höherer Ebenen können Module niedrigerer Ebenen durch Einspeisung eigener Daten für sich ausnutzen. Sie können die Module aber auch indirekt nutzen, indem sie sich auf deren Vorhandensein verlassen. So kann zum Beispiel das Verhaltensmuster „Kartographieren" sich darauf verlassen, dass es beim Herumfahren nicht mit Objekten kollidiert, da auf der unteren Ebene ein „Notstopp" existiert, welches den Roboter vor eine Kollision stoppt.

### 2.2.2 Dual-Dynamics Ansatz

Auch bei dem von H. Jäger und Kollegen [12] vorgestellten Ansatz gibt es Verhaltensmuster, die auf Ebenen verteilt werden. Jedoch haben in diesem Ansatz

nur Verhaltensmustern der untersten Ebene, sogenannte elementare Verhaltensmuster, Zugriff auf Aktoren.

Bei elementaren Verhaltensmuster unterscheidet man zwischen zwei Komponenten:

**Die Aktivierungsfunktion** gibt mit einem Wert zwischen 0 und 1 an, wie stark ein Verhaltensmuster aktiv wird.

**Die Zielfunktion** bestimmt die Änderungswünsche des Verhaltens für die verschiedene Aktoren.

Der Aktor erhält Aktivierungsstärken und Änderungswünsche von verschiedenen Verhaltensmustern. Die Änderungswünsche werden mit der Aktivierungsstärke des dazugehörigen Verhaltensmuster skaliert und auf den aktuellen Wert des Aktors aufsummiert. Somit erzeugen Verhaltensmuster mit einer niedrigen Aktivierungsstärke nur eine geringe Änderung des Aktors.

Verhaltensmuster höherer Ebenen hingegen besitzen keine Zielfunktion mehr, sondern nur noch eine Aktivierungsfunktion. Die Ergebnisse der Aktivierungsfunktionen einer höheren Ebene gehen als Parameter der Zielfunktionen niedriger Ebenen ein.

Während Elementarverhalten Reflexen und Instinkten entsprechen, haben Verhaltensmuster höherer Ebenen ihr Äquivalent in Stimmungen und anderen biologischen endogenen Einflüssen. Sie bestimmen nicht die Ausführung bestimmter Verhaltensmuster, sondern konditionieren lediglich deren Aktivierung und Ausführung. Sie sollen für Feineinstellungen sorgen. Zum Beispiel könnte im Falle des Fußballroboters die Aktivierung des Verhaltensmuster „Ungeduld“ das Verhaltensmuster „Schuss“ dazu veranlassen, früher als sonst, den Ball zu schießen.

#### 2.2.2.1 Beispiel

Um zu verdeutlichen, wie das dynamische Zusammenspiel solcher Verhaltensmuster aussehen kann, betrachten wir den Anlauf eines omnidirektionalen Roboters an einen Ball mit anschließenden Schuss auf ein Tor, wobei ein statisches Hindernis im Weg steht (Abbildung 2.3). Der Einfachheit halber ignorieren wir die Orientierung des Roboters. Unsere hypothetische Implementation besteht aus folgenden Verhaltensmustern:

**Fahr Hinter Ball** steuert auf eine Position zu, die vom Tor aus gesehen hinter dem Ball liegt. Die Geschwindigkeit wird umso geringer, desto näher der Roboter an der gewünschten Position ist. Die Aktivierungsstärke nimmt mit der Entfernung von der gewünschten Position ab.

**Ausweichen** steuert vom nächsten Hindernis weg. Die Aktivierungsstärke hängt von der Entfernung vom nächsten Hindernis und der Aktivierung vom Modul „Ungeduld“ ab. Je näher am Hindernis, desto stärker. Je ungeduldiger, desto weniger stark.

**Anlauf** steuert mit dem Ball auf das Tor zu. Das Verhaltensmuster wird umso stärker aktiv, desto näher der Roboter auf der Linie Tor-Ball liegt. Anlauf wird nicht aktiv, wenn der Roboter sich zwischen Ball und Tor befindet.

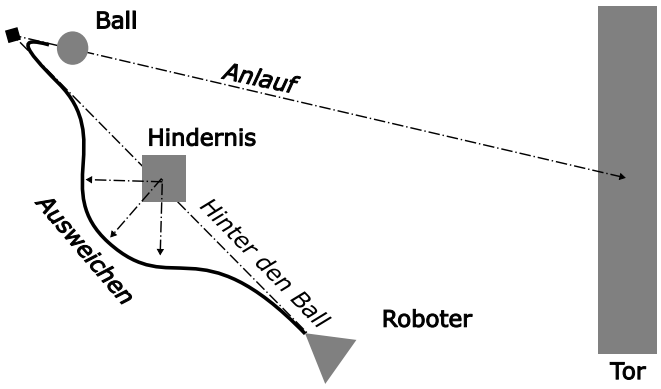


***Abbildung 2.3:*** Ausführung eines Schusses mittels Dual-Dynamics

**Schuss** löst den Schuss aus. Wird aktiv, wenn der Roboter nah genug am Ball ist und die Schussbahn ausreichend frei ist, wobei ein umso geringerer Sicherheitsabstand des Ball zu den Hindernissen gefordert wird, je größer die „Ungeduld“ ist.

**Ungeduld** ist kein Elementarverhalten. Die Aktivität hängt davon ab, wie lange der Roboter nicht am Ball war, dabei aber genügend Sicherheitsabstand zu anderen Robotern gewahrt hat und kein Schuss gefallen ist.

Am Anfang ist nur das Verhaltensmuster „Fahr Hinter Ball“ aktiv und steuert auf die Position hinter dem Ball zu. Da das Hindernis im Weg ist, steuert das Verhaltensmuster auch auf das Hindernis zu. Je näher der Roboter dem Hindernis kommt, desto mehr wird „Ausweichen“ aktiv, was bewirkt, dass der Roboter das Hindernis umfährt. Je näher der Roboter der Zielposition kommt, desto mehr bremst der Roboter ab. Ist der Roboter schon nah hinter dem Ball, so nähert sich der Roboter auch der Linie Ball-Tor, und daher nimmt dabei die Aktivität von „Anlauf“ zu. Das bewirkt, dass der Roboter sich immer mehr in Richtung Ball und Tor bewegt. Etwaige Abweichungen in der Position von der Ball-Tor-Linie werden durch „Fahr Hinter Ball“ ständig korrigiert. Ist der Ball nah genug am Roboter, so wird der Schuss ausgelöst.

„Ungeduld“ bewirkt durch die Abschwächung von „Ausweichen“, dass sich der gewahrte Abstand mit der Zeit verringert, so dass der Roboter bessere Chancen hat, den Ball zu bekommen. Dabei erhöht sich aber auch die Wahrscheinlichkeit einer Kollision. Ebenso wird mit der „Ungeduld“ die Schussfreudigkeit erhöht, was insgesamt zu einem aggressiveren Verhalten führt.

### 2.2.3 Hierarchische Reaktive Verhaltensmuster

Angelehnt an den Dual-Dynamics Ansatz wird auch beim ersten Verhaltensrahmen der FU-Fighters von 1999 [2] zwischen Aktivierungs- und Zielfunktionen unterschieden und die Verhaltensmuster sind in Ebenen gegliedert. Die Ebenen werden getaktet ausgeführt, wobei jede höhere Ebene mit einer geringeren Taktfrequenz ausgeführt

werden kann. Sensoren höherer Ebene fassen die Sensorwerte niedrigerer Ebenen zu langsamer wechselnden Wahrnehmungen zusammen.

Verhaltensmuster höherer Ebenen nehmen jedoch nicht nur durch ihre Aktivierungsfunktion Einfluss auf Verhaltensmuster niedrigerer Ebenen, sondern verfügen selbst auch über eine Zielfunktion, über die sie Aktoren beeinflussen können, wobei die Bedeutung der Aktoren erweitert wird. Sie entsprechen nicht mehr nur physischen Aktoren, sondern liefern auch Wahrnehmungen für Sensoren niedrigerer Ebenen. Der Wert des Aktors bestimmt sich wie im Dual-Dynamics Ansatz. Ein Aktorwert höherer Ebene wird jedoch nicht physisch umgesetzt, sondern in einem Sensor niedrigerer Ebene gespeichert. Dadurch ist es möglich, dass komplexere Verhaltensmuster, ähnlich wie bei der Subsumption-Architektur, primitivere Verhaltensmuster direkt nutzen können, indem sie ihnen ein Ziel vorgeben.

Die Parameter der Aktivierungsfunktion eines Verhaltens beschränken sich auf die aktuellen Sensorwerte, die letzten Aktivierungswerte derselben Ebene und die aktuellen Aktivierungswerte der nächsthöheren Ebene. Die Zielwerte der

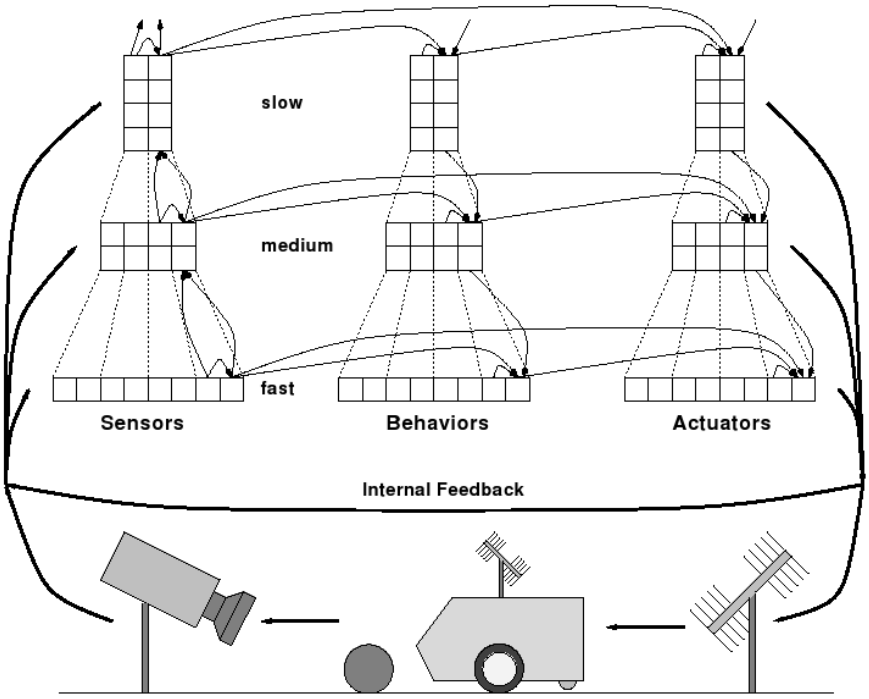


***Abbildung 2.4:*** Sensoren, Aktoren und Verhaltensmuster sind in Ebenen angeordnet. Verhaltensmuster greifen auf Sensoren und Aktoren der Ebene zu

Verhaltensmuster bestimmen sich nur aus den aktuellen Sensorwerten der Ebene und den Aktivierungswerten der nächsthöheren Ebene.

S. Lenser [16] orientiert sich an diesem Rahmen, verzichtet hierbei jedoch auf Ebenen. Anstelle des gewichteten Mittels der setzenden Rahmen findet eine andere Behandlung

konkurrierender Wünsche statt. Diese werden in eine Entweder-Oder-Beziehung gesetzt. Die Aktivierungswerte werden nicht mehr als Gewichtung betrachtet, sondern als erwartete Belohnung bewertet. Ein Entscheider versucht unter der Menge der Verhaltensmuster diejenigen auszuwählen, die nicht miteinander um einen Aktor konkurrieren und dabei die Summe der Belohnungen/Aktivierungswerte zu maximieren. Da das Auffinden des Maximums rechnerisch zu komplex ist, wird eine Heuristik verwendet.

#### 2.2.3.1 Beispiel

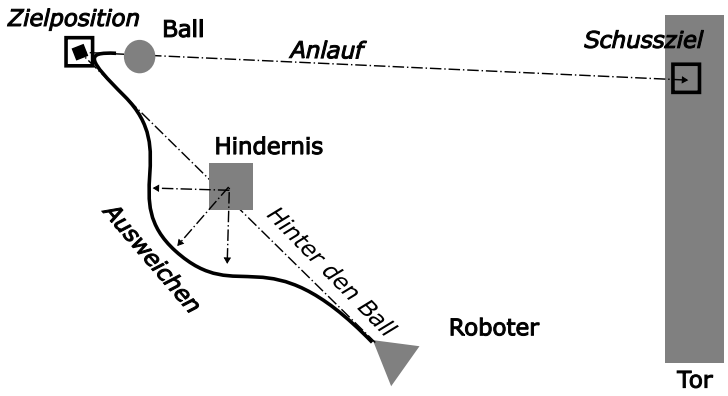


***Abbildung 2.5:*** Ausführung eines Schusses mittels Hierarchischen Verhaltensmuster

Wir gehen wieder von derselben Ausgangssituation wie im Beispiel 2.2.2.1 aus (Abbildung 2.5). Hier ließe sich das Problem mit den folgenden Verhaltensmustern realisieren:

**Fahr zum Ort** steuert eine gegebene Position mit einer gewünschten Geschwindigkeit an.

**Fahr Hinter Ball** wählt als Zielposition eine Position, die vom Schussziel aus gesehen hinter dem Ball liegt. Die Aktivierungsstärke nimmt mit der Entfernung von der gewünschten Position ab.

**Ausweichen** steuert vom nächsten Hindernis weg. Die Aktivierungsstärke hängt von der Entfernung vom nächsten Hindernis ab. Je näher am Hindernis, desto stärker.

**Anlauf** wählt das Schussziel als Zielposition. Wird umso stärker aktiv, je mehr der Roboter auf der Linie Schussziel-Ball liegt.

**Schuss** löst den Schuss aus. Wird aktiv, wenn der Roboter nah genug am Ball ist und die Schussbahn ausreichend frei ist.

**Wähle Schussziel** wählt unter der Berücksichtigung der Hindernisse die beste Position im Tor als Schussziel.

Der Ablauf ist vergleichbar mit dem Beispiel in Absatz 2.2.2.1. Jedoch anstelle das Schussziel fest in „Fahr Hinter Ball“ und „Schuss“ vorzugeben, wird es von anderen Verhaltensmustern (im Beispiel „Wähle Schussziel“) bestimmt. So wäre es auch möglich einen anderen Roboter als Schussziel vorzugeben und so einen Pass zu realisieren. Ein weiterer Unterschied ist, dass die Ansteuerung eines Ortes aus „Fahr Hinter Ball“ extrahiert wurde und in ein eigenes Verhaltensmuster „Fahr zum Ort“ gekapselt wurde. Dadurch steht die Funktionalität auch anderen Verhaltensmustern, in unserem Beispiel „Anlauf“, zur Verfügung. Im Falle von Dual-Dynamics müsste die ähnliche Funktionalität in beiden Verhaltensmuster implementiert werden.

## 2.3 Hybride Architekturen

Hybride Architekturen sollen den reaktiven und den planenden Ansatz vereinen. Eine Variante ist die Unterteilung in drei Schichten:

**Fähigkeiten-Schicht:** Die Menge von Verhaltensmuster sind Fähigkeiten, die die Detailarbeit schnell und reaktiv erledigen

**Sequenzer-Schicht:** Der Sequenzer entscheidet, welche Verhaltensmuster aktiviert werden und versorgt sie mit Parametern. Sollte die Aktion nicht zur erwarteten Situation führen, muss der Sequenzer darauf reagieren.

**Planer-Schicht:** Der Planer durchsucht den Zustandsraum, um einen oder mehrere mögliche Lösungswege für den Sequenzer zu finden.

Erann Gat gibt in „On Three Layer Architectures“ einen Überblick über die verschiedensten Implementierungen dieser Variante [9].

### 2.3.1 Beispiel

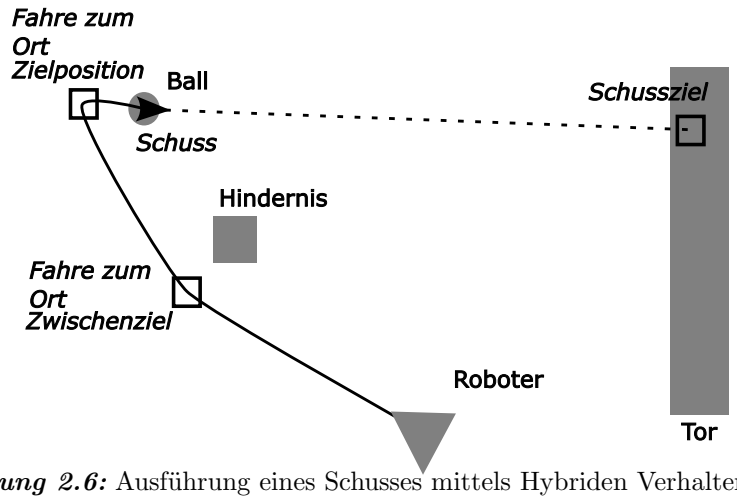


***Abbildung 2.6:*** Ausführung eines Schusses mittels Hybriden Verhaltensmuster

In diesem Beispiel ist die Fähigkeiten-Schicht mit den Verhaltensmustern „Fahr zum Ort“, „Schuss“ und „Ausweichen“ ausgestattet. Ziel des Spiels ist es, ein Tor zu schießen, und somit ist das auch das Ziel des Planers. Fangen wir nun mit dem Ziel an und suchen rückwärts einen Lösungsweg.

Damit der Ball ins Tor kommt, muss sich hinter dem Ball ein Roboter befinden und dieser „Schuss“ mit dem Tor als Ziel ausführen, kurz: („Schuss“, Tor). Damit hinter dem Ball ein Roboter steht, muss der Roboter hinter den Ball fahren („Fahr zum Ort“, hinter den Ball). Die Aktion kann aber nicht direkt ausgeführt werden, da ein Hindernis im Weg ist. Da neben dem Hindernis Platz ist, muss der Roboter zuerst dorthin fahren („Fahr zum Ort“, neben Hindernis).

Der Sequenzer erhält vom Planer die umgekehrte Folge der eben genannten Aktionen und Parameter und führt nun diese Befehle nacheinander aus (Abbildung 2.6). Der Sequenzer überprüft fortlaufend, ob die Aktionen noch möglich sind. Bewegt sich zum Beispiel das Hindernis zu nah an den geplanten Weg, so muss der Sequenzer eine Neuplanung veranlassen. Während der Planung kann der Sequenzer „Ausweichen“ aktivieren, und so eine Kollision vermeiden.

Der Planer kann auch schon Alternativen für den Sequenzer bereitgestellt haben, wie zum Beispiel einen Wegpunkt auf der anderen Seite des Hindernis. Dem Sequenzer wird somit die Möglichkeit gegeben, auf eine unvorhergesehene Situation zu reagieren.

# 3 Low-Level Steuerung

Wie auch immer die Steuerungsarchitektur geartet sein mag, so müssen bei Robotern die Befehle mechanisch umgesetzt werden. Zur Ansteuerung der physischen Aktoren werden für gewöhnlich Mikrocontroller verwendet. Die Elektronik der FU-Fighters basiert auf einem Mikrocontroller der HCS12-Baureihe der Firma Freescale. Es handelt sich dabei um einen mit 8MHz getakteten Prozessor mit 12KByte RAM. Zwar ist die Leistungsfähigkeit dadurch entsprechend begrenzt, dafür sind an der Elektronik Aktoren und Sensoren direkt angeschlossen, wodurch Wahrnehmung und Ansteuerung nur eine geringe Verzögerung haben.

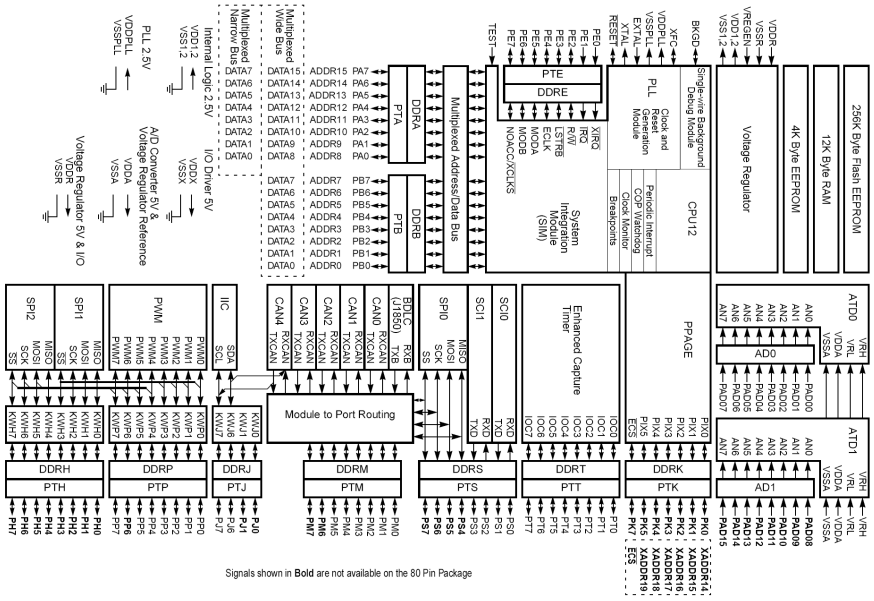


***Abbildung 3.1:*** Schema des HCS12DG256 Mikrocontrollers (aus [8])

Es bietet sich daher an, so viel von der Steuerung wie möglich auf den Mikrocontroller zu verlagern. Aufgrund der geringen Rechenleistung und begrenzten Sensorik wird hierbei der reaktive Ansatz verfolgt.

Die Elektronik dient der Umsetzung der durch die serielle Schnittstelle gesendeten Befehle. Dafür werden die angeschlossenen Aktoren angesteuert und die Aktionen mittels der Sensoren geprüft. Im Falle der Small-Size wird das serielle Signal über Funk übertragen, deswegen müssen die Daten vor Fehlern gesichert werden. Die Befehle

umfassen Fahrgeschwindigkeiten in X,Y-Richtung, Drehgeschwindigkeit, Aktivierung der Dribbelwalze oder einer der maximal zwei Schüsse mit Angabe einer Schussstärke.

An die Elektronik sind folgende Sensoren angeschlossen:

- 8 Tickzähler (Je einer pro Motor und Drehrichtung)
- 1 Lichtschranke
- 1 Gyroskop (nur Small-Size)

Folgende Aktoren stehen zur Verfügung:

- 4 H-Brücken (PWM-gesteuert) für die Ansteuerung von Gleichstrommotoren
- 1 Schuss (2 bei Small-Size-Robotern)
- 1 Rücksetz-Signal für USB (nur Middle-Size)
- 1 Dribbelwalze (nur Small-Size)

## 3.1 Regelung des Roboters

Unter einer Regelung versteht man den Vorgang der fortlaufende Beeinflussung einer gemessenen Größe, genannt Regelgröße oder auch Ist-Wert. Die Beeinflussung erfolgt aufgrund des Vergleichs der Regelgröße mit einer gegebenen Größe, genannt Führungswert oder auch Soll-Wert.

In unserem Fall sind die als Kommando gegebenen Fahr- und Drehgeschwindigkeiten die Führungsgrößen. Wie können wir aber die Regelgröße messen und wie können wir diese beeinflussen?

Hierzu müssen wir das physikalisches Modell des Roboters betrachten.

### 3.1.1 Messen der Bewegung

Die Antriebseinheit unseres Roboters besteht aus 4 omnidirektionalen Rädern, die alle tangential auf einem Kreis um den Schwerpunkt angeordnet sind, wie es in Abbildung 3.3 zu sehen ist. Die Räder sind omnidirektional, da an ihnen kleine Räder angebracht sind, wodurch man mit ihnen nur Kraft entlang der Rollrichtung des Rades ausüben kann (Abbildung 3.4). Um diese Kraft zu erzeugen, sind Gleichstrommotoren an den Achsen angebracht. Die Motoren sind mit Tickgebern ausgestattet, deren Signale auf der Elektronik registriert und mitgezählt werden. Die Anzahl der Ticks in einem Zeitintervall entspricht der mittleren Geschwindigkeit entlang eines Rades.

Uns interessiert jedoch die Gesamtbewegung des Roboters. Um den Zusammenhang zu erfassen, drehen wir die Betrachtungsweise erst einmal um und bestimmen, welche

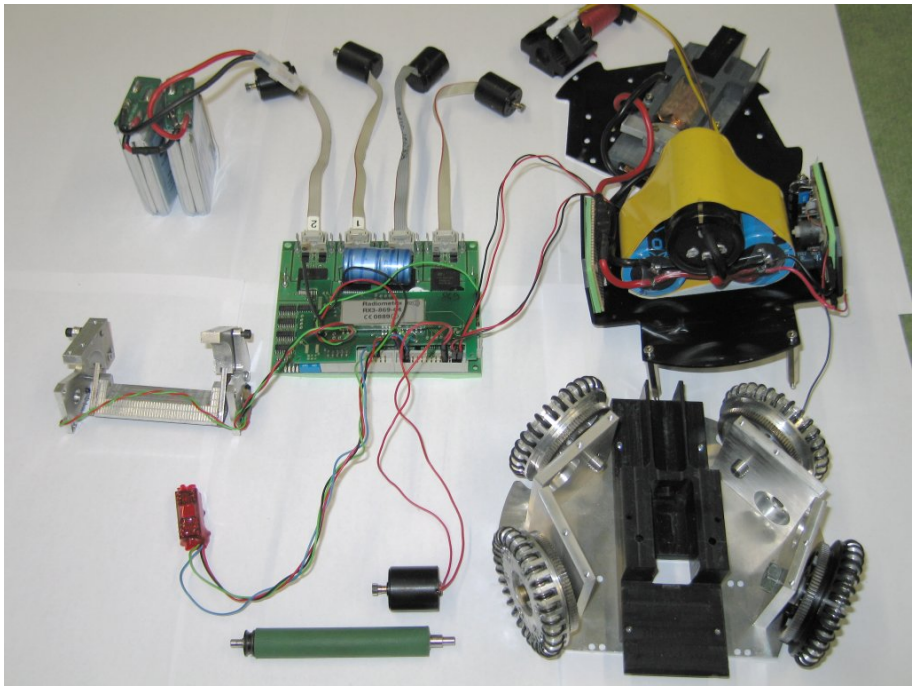

***Abbildung 3.2:*** SmallSize-Roboter zerlegt. In der Mitte befindet sich die Elektronik mit Funkmodul an der alle Sensoren und Aktoren angeschlossen sind, darüber die Antriebsmotoren und rechts oben im Bild die elektromagnetische Schussaparate. Rechts von der Elektronik die Kondensatoren, Schalter und Ladeelektronik für die Schussaparate. Unten rechts im Bild ist das Fahrgestell und unten ist die Dribbelwalze und der dazugehörige Motor zu sehen. Das rote Objekt links unten ist das Gyroskop. Links der Elektronik liegt der Vorbau des Roboters mit eingebauter Lichtschranke.

Geschwindigkeiten sich an den Rädern durch eine beliebige Bewegung über die Ebene ergeben müssen.

Bei einer infinitesimalen Bewegung können Translation und Drehung unabhängig voneinander betrachtet werden. Eine Drehung ist einfach bestimmt, da alle Räder den gleichen Abstand $R$ zum Schwerpunkt haben. Drehen wir uns um einen Winkel $\partial\phi$, so stellen wir an den Rädern eine Bewegung von $R\partial\phi$ fest.

Bei der Translation betrachten wir, wie sich ein einzelnes Rad bei einer beliebigen Ausrichtung gegenüber der Fahrtrichtung verhält. Am Einfachsten sind die Extrema: Ist das Rad parallel zur Fahrtrichtung, so dreht sich das Rad entsprechend der Bewegung, steht das Rad senkrecht zur Fahrtrichtung, so findet sämtliche Bewegung über die kleinen Rädchen statt, und wir stellen keine Drehung des Rades fest. Aus diesen Extremfällen und der Graphik 3.5 wird ersichtlich, dass am Rad nur die Projektion der Translation auf dessen Rollrichtungsvektor festgestellt wird.

Platzieren wir die Räder entsprechend Abbildung 3.3, so ergeben sich aus den verschiedenen Ausrichtungen der Räder verschiedene Winkel zur Seitwärts- und

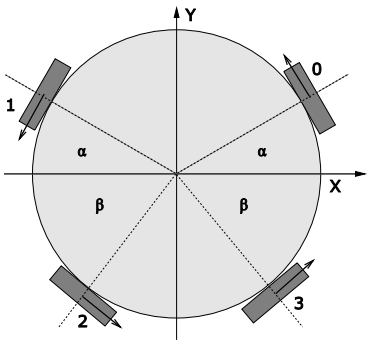


***Abbildung 3.3:*** Omniroboter: Radanordnung

Vorwärtsbewegung. Kombinieren wir die Projektionen der Bewegungen in X- und Y-Richtungen auf die Richtungsvektoren der Räder und fügen sie mit der Rotation zusammen, so ergibt sich für die Bewegung an den Rädern das lineare Gleichungssystem:

$$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} -\sin\alpha & \cos\alpha & 1 \\ -\sin\alpha & -\cos\alpha & 1 \\ \sin\beta & -\cos\beta & 1 \\ \sin\beta & \cos\beta & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ R\frac{\partial\phi}{\partial t} \end{pmatrix} \tag{3.1}$$

Dieses Lineare Gleichungssystem ist überbestimmt. Zur Umkehrung der Gleichung kann die Pseudoinverse als Lösung herangezogen werden, und somit mittels der Tickzahl die Regelgröße gemessen werden. Bei den Small-Size Robotern lässt sich die Drehgeschwindigkeit mit dem Gyroskop auch direkt messen.

### 3.1.2 Beeinflussung der Bewegung

Wir vereinfachen das Modell, und nehmen an, dass die Kraft des Motors direkt und ohne Verlust auf dem Boden ankommt und dort nur in der Ebene wirkt. Wir vernachlässigen Schlupf, Gewichtstransfer durch erhöhten Schwerpunkt und dergleichen. Da der Roboter ein starrer Körper ist, addieren sich die Kräfte vektoriell zu einer resultierenden Kraft in der Ebene. Das Kreuzprodukt zwischen der Antriebskraft eines Rades und des Vektors vom Schwerpunkt zur Radaufhängung ergibt ein Drehmoment senkrecht zur Ebene des Roboters (Abbildung 3.6). Die Summe der Drehmomente ergibt das Gesamtdrehmoment, das auf den Roboter wirkt. Für die Beträge der Kräfte ergibt sich

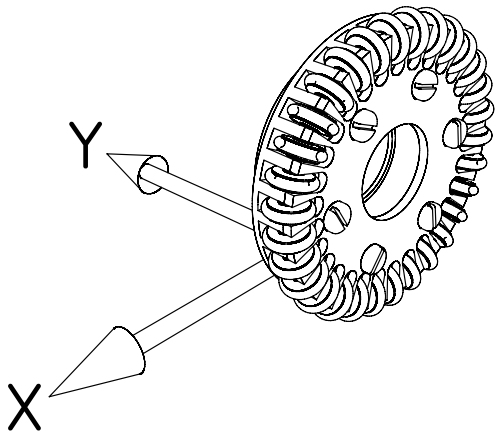


***Abbildung 3.4:*** Omnirad: Freirollend nach $Y$, Antrieb entlang $X$

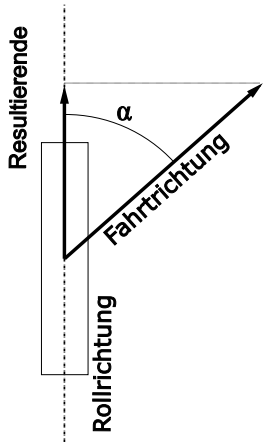


***Abbildung 3.5:*** Skizze der registrierten Bewegung bei einer von der Rollrichtung $X$ um $\alpha$ abweichenden Bewegung

die Formel 3.2.

$$\begin{pmatrix} F_x \\ F_y \\ M \end{pmatrix} = \begin{pmatrix} -\sin\alpha & -\sin\alpha & \sin\beta & \sin\beta \\ \cos\alpha & -\cos\alpha & -\cos\beta & \cos\beta \\ R & R & R & R \end{pmatrix} \begin{pmatrix} F_0 & F_1 & F_2 & F_3 \end{pmatrix}^T \tag{3.2}$$

Das lineare Gleichungssystem 3.2 gibt an, welche Kräfte auf den Roboter wirken, wenn wir verschiedene Einzelkräfte an den Motoren erzeugen. Da wir aber Einfluss auf die Fahr- und Drehgeschwindigkeiten nehmen wollen, interessiert uns wieder der umgekehrte Fall: Welche Kräfte müssen wir von den Motoren fordern, um den Roboter zu bremsen oder zu beschleunigen?

Nun ist die Umkehrung des linearen Gleichungssystems unterbestimmt, eine Lösung ist somit nicht eindeutig. Dies ist leicht nachvollziehbar, da es theoretisch keinen Unterschied macht, ob bei der Vorwärtsbeschleunigung die Kraft gleichverteilt aus allen vier Rädern oder nur aus den beiden hinteren kommt oder ob sogar die vorderen Motoren aktiv gegen die hinteren arbeiten. Praktisch gesehen ist zumindest der letzte Fall nicht wünschenswert.

Alle Fälle, wo die Motoren nur gegeneinander arbeiten, ohne eine resultierende Kraft zu erzeugen, bilden den Kern des linearen Gleichungssystems.

Die Pseudoinverse liefert die eindeutige Menge der Lösungen, die vom Kern maximal entfernt sind. Daher bietet es sich an, diese zur Bestimmung einer Lösung zu wählen.

Wie ergeben sich die Kräfte an den Motoren? Eine genaue Erläuterung der physikalischen Vorgänge wird hier nicht gegeben. Dazu sei der interessierte Leser an die entsprechende Literatur verwiesen.

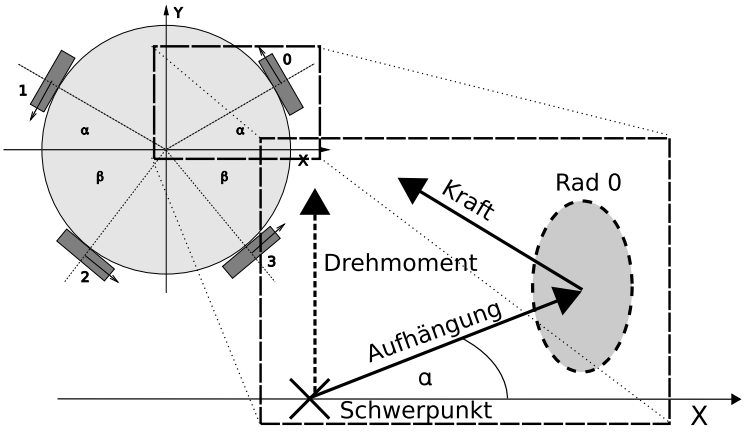


***Abbildung 3.6:*** Die Kraft eines Rads erzeugt ein Drehmoment, welches Senkrecht zu Aufhängung und der Kraft steht

Wichtig für die Regelung ist jedoch, dass proportional zu der Spannung am Motor ein Drehmoment entsteht. Diese Spannung ergibt sich aus der Summe der Eigeninduktion und der angelegten Spannung. Der Betrag der Eigeninduktion ist proportional zur momentanen Drehgeschwindigkeit, arbeitet jedoch entgegen der Spannung, die die Drehung erzeugt. Die angelegte Spannung selbst wird durch die Schaltung der H-Brücken bestimmt.

Vereinfacht sind H-Brücken Schalter, mittels derer die Akkuspannung in beliebiger Richtung gepolt an den Motor angelegt werden kann. Weiterhin kann der Motor kurzgeschlossen oder der Stromkreis vollständig unterbrochen werden.

Ist der Motor kurzgeschlossen, so liegen $0V$ an, und die Eigeninduktion erzeugt ein bremsendes Drehmoment. Ist der Stromkreis unterbrochen, so dreht der Motor frei. Liegt die Akkuspannung an, so entsteht ein dementsprechendes Drehmoment.

Die H-Brücken werden mittels Puls-Weiten-Modulation (PWM) periodisch für eine gewisse Zeit angeschaltet. Dadurch liegt in dieser Zeit das durch die Polung und die momentane Drehgeschwindigkeit bestimmte maximal mögliche Drehmoment an. Aufgrund der hohen Grundfrequenz des PWM-Signals wirkt dies, als ob ein dem Verhältnis von aktiver Zeit zur Gesamtperiode der PWM (kurz: Tastverhältnis) entsprechendender Teil des Maximaldrehmoments anliegt.

Um also eine bestimmte Kraft in eine bestimmte Richtung zu erzeugen, müssen wir sie nach Formel 3.2 auf die Räder verteilen. Wir bestimmen dann anhand der momentanen Drehgeschwindigkeit des Rades die induzierte Spannung und summieren sie mit der anzulegenden Spannung. Dieser Wert ist proportional zur maximal möglichen Kraft, die am Rad erzeugt werden kann. Das Verhältnis von gewünschter Kraft zur momentan maximalen Kraft ergibt das Tastverhältnis der PWM. Wollen wir bremsen, so gibt es zwei mögliche Varianten, die Bremskraft zu erzeugen: mittels Gegenspannung oder

mittels Eigeninduktion. Ist die Kraft der Eigeninduktion ausreichend, so kann mit dieser die Bremskraft energiesparend erzeugt werden.

### 3.1.3 Regelung der Bewegung

Wir bestimmen den Ist-Wert der Regelgrößen indirekt, indem wir zuerst die Ticks in einem Regelintervall messen, was den Drehgeschwindigkeiten der Räder entspricht. Den Ist-Wert erhalten wir indem wir diese Geschwindigkeiten mittels der Pseudoinversen von Formel 3.1 in eine Bewegung umrechnen. Diese Bewegung lässt sich nach den Newtonschen Axiomen beeinflussen. Die dazu notwendigen Kräfte können wir entsprechend der Pseudoinversen von Formel 3.2 auf die Räder verteilen und durch eine einfache Rechnung in ein PWM-Signal umgesetzten.

Welche Stellwerte, also Kräfte, müssen wir aber wählen, um die gewünschten Sollwerte zu erreichen? Würde das oben genannte Modell perfekt sein, so ließe sich das einfach ausrechnen. Nun ist aber das Modell stark vereinfacht, und außerdem hängt das Ergebnis von verschiedenen äußeren und dynamischen Einflüssen ab. Zu diesen Einflüssen zählen zum Beispiel Batteriespannung, Zustand der Motoren und Getriebe oder Bauunterschiede zwischen den Robotern.

Daher muss geregelt werden. Die Regelungstechnik bietet hierzu verschiedene Regelalgorithmen, wovon die gebräuchlichsten PID-Regler sind. PID-Regler steht für Proportional-Integral-Differential-Regler, welcher durch die folgende Formel beschrieben werden kann:

$$u(t) = k_p e(t) + k_i \int_0^t e(x)dx + k_d e'(t) \tag{3.3}$$

In der Formel steht $e$ für Abweichung des Ist- vom Sollwert und $u$ für den Stellwert. Die Größen $k_p$, $k_i$ und $k_d$ sind Parameter des Reglers.

Diese kontinuierliche Gleichung wird durch die entsprechende diskretisierte Formel ersetzt.

$$u_t = k_p e_t + k_i \sum_{j=0}^{t} e_j + k_d \frac{e_t - e_{t-1}}{\Delta t} \tag{3.4}$$

An Stelle des Differenzenquotienten tritt jedoch in dieser Implementation eine geglättete Variante, da sonst das Rauschen in der Messgröße zu stark verstärkt würde.

Zusätzlich wird die Änderung des Sollwertes auf ein erreichbares Maß begrenzt: Ist die geforderte Beschleunigung größer als die experimentell beobachtete Höchstbeschleunigung, so wird der Sollwert nicht sofort auf die gewünschte Geschwindigkeit geändert, sondern in jedem Regelschritt nur um die maximale Beschleunigung erhöht, bis der Sollwert mit dem vom Befehl geforderten übereinstimmt. Wird der Sollwert auf einen unerreichbaren Wert gesetzt, dann summieren sich die Differenzen von Erreichbaren und Geforderten im Integralterm. Dies bewirkt jedoch nicht, dass der Sollwert schneller erreicht wird, da der Proportionalterm bereits so ausgelegt ist, ihn so schnell wie möglich zu erreichen. Dieser aufgebaute Integralterm kann sich erst wieder abbauen, wenn das Ziel erreicht wurde und bewirkt somit

ein Überschießen des Ist-Werts über den Soll-Wert. Ein weiterer Grund ist, dass die Beschleunigung größtenteils nicht durch die Kraft der Motoren begrenzt wird, sondern durch die Haftung der Räder auf dem Boden. Die Forderung einer höheren Beschleunigung hätte zur Folge, dass mehr Kraft eingesetzt wird, als die Haftreibung erlaubt, was zu durchdrehenden Rädern führt.

Die Parameter des PID-Reglers wurden nach der Heuristik des CMU-PID-Tutorials [18] bestimmt: Erhöhe Proportionalteil $k_p$ bis die gewünschte Beschleunigung erreicht wird. Um das Überschießen zu mindern, erhöhe den Differentialfaktor $k_d$. Gibt es eine bleibende Regeldifferenz, so beseitige diese durch die Erhöhung des Integralfaktors $k_i$. Überschießt dadurch der Ist-Wert, so kompensiere dies durch einen erhöhten Differentialfaktor $k_d$.

## 3.2 Programmaufbau der Elektroniksoftware

Das Programm ist primär Interrupt getrieben, wobei die Interrupt-Routine in einen unteren und einen oberen Teil (Upper-Half/Lower-Half) unterteilt ist, wie es bei Treibern in Betriebssystemen üblich ist. Der untere Teil erledigt meist nur die nötigste Arbeit, wie Bestätigung des Interrupts, Sicherung der Daten und verschiebt den komplexeren Teil außerhalb des Interruptkontexts. Die Bearbeitung dieses Teils geschieht in der Hauptschleife des Programms. Sie wartet auf einen beliebigen Interrupt und überprüft dann, ob die obere Hälfte einer Routine zur Ausführung markiert ist. Ist eine Routine markiert, so wird diese von der Hauptschleife ausgeführt.

Folgende Interrupts werden wie folgt behandelt:

- Serieller Empfang: lese das Zeichen von serieller Schnittstelle und gebe es an den finiten Automaten zur Decodierung. Ist das Paket vollständig, so wird die obere Hälfte der Empfangsroutine zur Ausführung markiert.
- Serieller Versand: falls Daten im Puffer vorhanden sind, versende das nächste Zeichen.
- Lichtschranke: Schussbehandlung
- Echtzeittakt: wird jede Millisekunde ausgelöst und erledigt folgende Aufgaben:
  - Schussbehandlung
  - Alle 4 ms: lese die Tickzähler aus und markiere die Regelung zur Ausführung.

Schussbehandlung bedeutet: Falls die Lichtschranke durchbrochen ist, wird entsprechend der gewünschten Schussstärke der Schuss für eine bestimmte Zeitdauer ausgelöst. Längere Auslösezeiten führen zu stärkeren Schüssen. Da die Schussstärke empfindlich auf die Auslösedauer reagiert, wird die Schussbehandlung zu Beginn jedes Taktes ausgeführt, damit die Auslösezeiten möglichst genau eingehalten werden. Die Auslösezeit kann sowohl Vielfache als auch Bruchteile des Taktes umfassen. Für die Vielfachen wird jeder Takt runtergezählt, während für die Bruchteile aktiv gewartet wird. Das aktive Warten

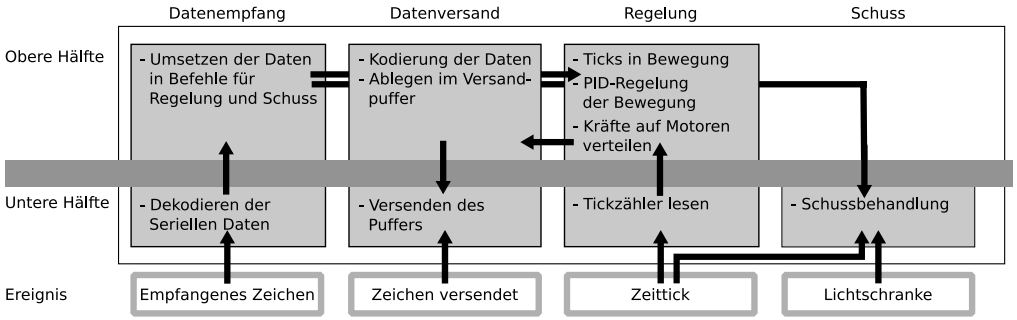


***Abbildung 3.7:*** Schema der Zusammenhänge der Software der Elektronik

kann die Regelung oder die Kommunikation beeinträchtigen. Da während der wenigen Millisekunden, in denen der Schuss ausgeführt wird, der Roboter schwer zu kontrollieren ist, spielt diese Tatsache jedoch keine große Rolle. Nach jedem Schuss wird eine weitere Auslösung des Schusses für eine gewisse Zeit gesperrt, damit Energie für einen neuen Schuss aufgeladen werden kann und die Schussmechanik nicht zu stark belastet wird. In der oberen Hälfte der Regelung wird die oben genannte Umrechnung der Ticks in eine räumliche Bewegung durchgeführt. Anschließend regelt der PID-Regler die Bewegung, wobei die Beschleunigung begrenzt wird. Schließlich werden die Regelwerte zurück auf die Motoren verteilt und in die PWM-Signale der Motoren umgerechnet. Damit bei Ausfall der Kommunikation der Roboter nicht blind durch die Gegend fährt, wird nach einer gewissen Zeit der Roboter gestoppt, falls keine Befehle mehr ankommen. Im Falle der Middle-Size wurde beobachtet, dass die USB-Seriell-Umwandler unter Belastung ausfallen können. Daher wird im Falle eines Ausfalls der Kommunikation periodisch ein Rücksetz-Signal an den Umwandler gesendet. Die obere Hälfte des Datenempfangs umfasst die Umsetzung des Pakets in Befehle für Regelung, Schuss und Dribbelwalze.

Wurde eine Regelung durchgeführt, so wandelt der Datenversand die Messung in eine geeignete Kodierung um und speichert sie in einem Puffer zum Versand. Die untere Hälfte der Versandroutine leitet die Daten des Puffers Zeichen für Zeichen an die serielle Schnittstelle weiter. Im Falle der Middle-Size werden so die Daten als weitere Sensordaten der High-Level Verarbeitung zur Verfügung gestellt.

# 4 High-Level Steuerung

Die in Absatz 2.2.3 beschriebene Struktur wurde seit 1999 implementiert und diente in den folgenden Jahren als Basis für die Weiterentwicklung des Verhaltens der Roboter. Es wurden Verhaltensmuster für verschiedene Generationen von Robotern realisiert und diente sowohl zur zentrale Kontrolle von mehreren Agenten als auch zur dezentralen Steuerung einzelner Roboter. Hierbei nahm die Anzahl der implementierten Verhaltensmuster als auch die Komplexität des Zusammenspiels eben dieser zu. Dadurch traten Schwachpunkte der Architektur und deren Implementierung zu Tage, welche durch ein neuen Rahmen in dieser Arbeit behoben werden sollen.

## 4.1 Zielsetzung

Aufgrund der Ausrichtung auf den Wettbewerb lag der Schwerpunkt bei der Entwicklung des neuen Rahmens darauf, ein System zu entwickeln, das zu jedem Zeitpunkt möglichst konkurrenzfähig ist. Die Schwachpunkte des alten Rahmens sollten bereinigt werden. Gleichzeitig musste sichergestellt sein, dass zu keinem Zeitpunkt ein qualitativer Rückschritt erfolgte. Dies bedeutete, dass sich die alten Realisierungen von Verhaltensmuster einfach übernehmen lassen sollten, da die darin kodierte Erfahrung sich nicht einfach reproduzieren lassen.

Zur Überprüfung der Entwicklung sollten das alte und das neue System koexistieren, um die neue Implementation mit der alten vergleichen zu können.

Da der Verhaltensrahmen mittlerweile auch auf Notebooks, statt auf einer leistungsstarken Workstation, eingesetzt wurde, war es wünschenswert, den Betriebsmittelbedarf zu verringern.

## 4.2 Aufbau des alten Verhaltensrahmen

Zwar wurde in Absatz 2.2.3 schon das Konzept des Verhaltensrahmen erläutert, jedoch sind über die Jahre ein paar Änderungen vorgenommen worden, die nicht an anderer Stelle publiziert wurden. Außerdem liegen einige Probleme mehr im Detail der Implementierung, daher soll der alte Rahmen an dieser Stelle genauer erläutert werden.

Spiel, Team und Roboter werden durch entsprechende Klassen und Abhängigkeiten modelliert. Die Klasse Spiel hat zwei Teams, jedes Team eine beliebige Anzahl an Robotern. Spiel, Team und Roboter sind jeweils Agenten, die eine hierarchische Verhaltensstruktur besitzen, wie sie in Abbildung 4.1 dargestellt ist. Die Anzahl der Ebenen wurde pro Agent auf drei fixiert.

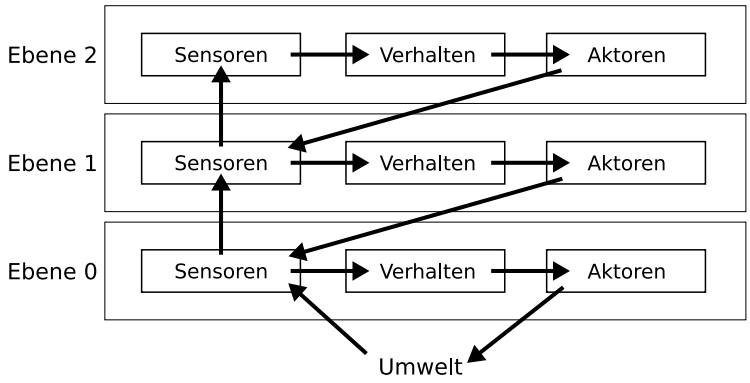


***Abbildung 4.1:*** Zusammenhang der drei Ebenen eines Agenten

Die unterste Ebene eines Teams erhält Zugriff auf die obersten Ebenen der Roboter und des Spiels, wodurch die Ebenen des Teams oberhalb der von Spiel und Roboter anzusiedeln sind (siehe Abbildung 4.2). Jeder Agent besitzt drei Exemplare der abstrakten Klasse Ebene. Bei einer Implementation eines Agenten werden diese Ebenen durch entsprechende Konkrete ersetzt (Beispiel: Abbildung 4.3).

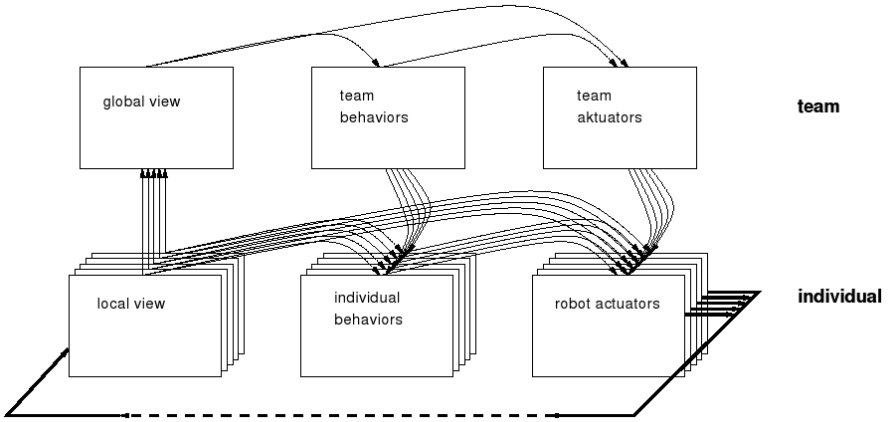


***Abbildung 4.2:*** Verhältnis von Teamebenen zu Roboterebenen (aus [2])

Eine konkrete Ebene hält Sensoren, Verhaltensmuster und Aktoren als öffentliche Klassenbestandteile und registriert sie bei ihren Managern. Eine Ebene hat Zugriff auf eine konkrete Implementierung einer niedrigeren Ebene und stellt durch sie die Abhängigkeiten zwischen den Sensoren der unteren Ebene und ihren eigenen Sensoren und Aktoren her. Jedem Verhaltensmuster wird ein Verweis auf ein konkretes Exemplar einer Ebene übergeben, wodurch dieses Zugriff auf Sensoren und Aktoren der Ebene erhält. Somit ist die Existenz und der Typ der Sensoren und Aktoren durch die statische Prüfung des Compilers gesichert. Der SensorManager übernimmt die Aggregierung

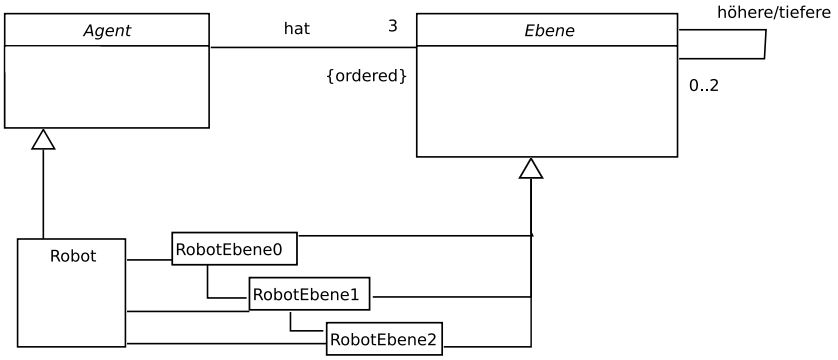


***Abbildung 4.3:*** Die abstrakten Ebenen des Agenten werden in der Implementation des Roboters durch konkrete Implementationen ersetzt.

der Sensorwerte der niedrigeren Ebene, der VerhaltenManager die Ausführung der Verhaltensmustern und der AktorManager das Setzen der mit den Aktoren assoziierten Sensoren mit den entsprechenden Werten der Aktoren.

Sensoren speichern in einem Takt Werte eines Datentyps, die dadurch auch im Nachhinein von Verhaltensmustern abgerufen werden können. Die Berechnung ist nicht wie im Konzept beim Zugriff auf den aktuellen Sensorwert begrenzt, sondern hat Zugriff auf alle bisher berechneten Werte. Dafür werden die Möglichkeiten der Abhängigkeit der Aktivierungs- und Zielfunktionen an anderer Stelle weiter eingegrenzt. So berechnet sich die Zielfunktion nur noch aus den Sensoren der selben Ebene und nicht mehr aus den Aktivierungswerten höherer Ebenen.

Die Aktivierungsstärke wird in zwei Schritten berechnet. Im ersten Schritt kann sich die Aktivierungsfunktion aus allen Sensoren der Ebene des Verhaltens berechnen. In der zweiten Stufe findet die Hemmung durch andere Verhaltensmuster statt. Dem VerhaltenManager können Hemmungen zwischen verschiedenen Verhaltensmustern einer Ebene gemeldet werden. Die Werte der Aktivierungsfunktionen, die ein Verhaltensmuster hemmen, werden summiert. Die Summe wird auf eins begrenzt. Das Produkt des Wertes der Aktivierungsfunktion mit (1-„die Summe“) ergibt die Aktivierungsstärke des Verhaltens. Wenn auch in der Verhaltensforschung Fälle bekannt sind, in denen sich zwei Verhaltensmuster gegenseitig hemmen, und dadurch ein drittes zu Tage tritt, so ist dieser Fall vom Entwurf her nicht gewünscht. Daher wird vom VerhaltenManager eine kreisfreihe Hemmstruktur erzwungen.

Sensoren erhalten ihre Werte auf drei unterschiedliche Arten: Bei der ersten Möglichkeit wird der Mittelwert der Sensorwerte eines anderen Sensors in einem gegebenen Zeitfenster ermittelt. Dieser Sensor muss auf einer niedrigeren Ebene angesiedelt sein.

Die zweite Möglichkeit nutzt die Aktoren einer direkt über der eigenen liegenden Ebene. Die Verhaltensmuster dieser Ebene können deren Aktoren einen Wunschwert mitteilen. Diese Aktoren bilden nun das mit der Aktivierungsstärke der Verhaltensmuster

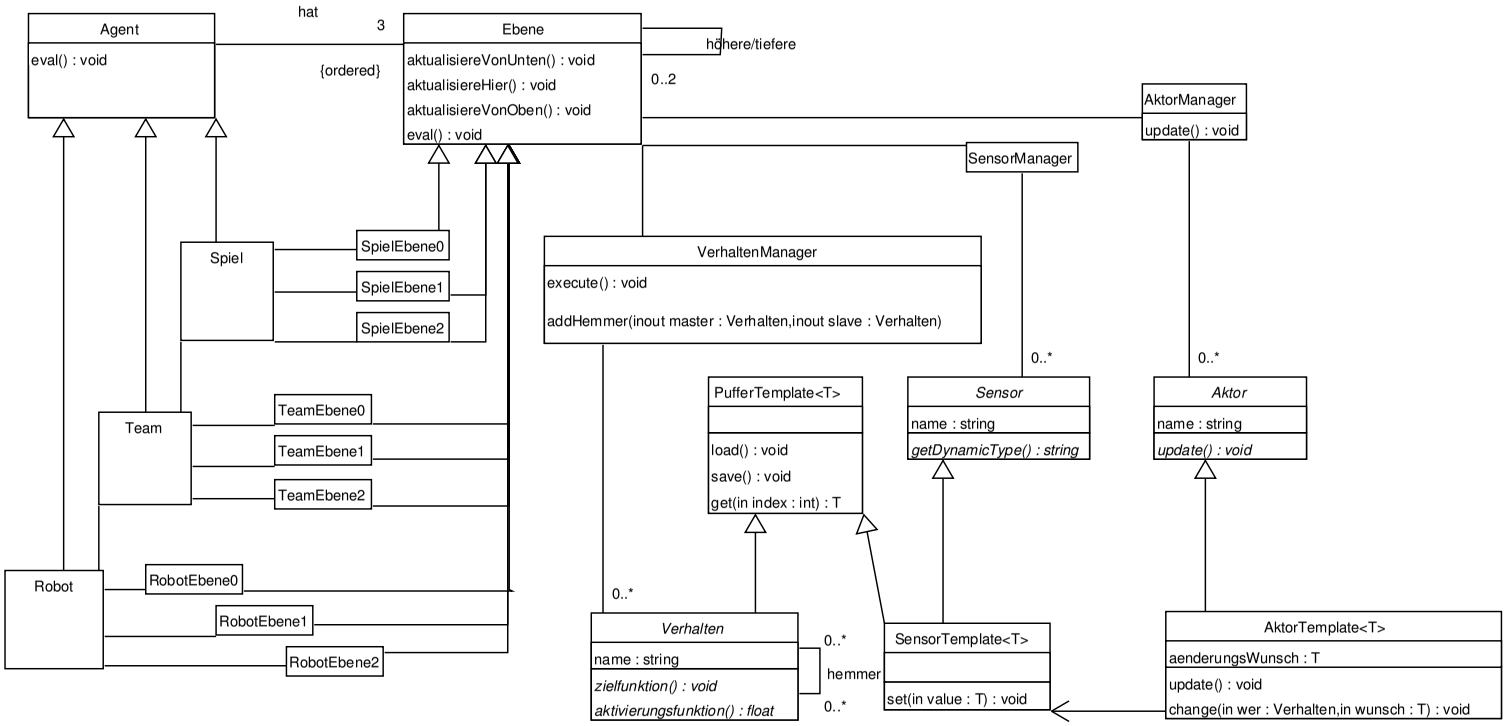


***Abbildung 4.4:*** UML-Diagramm des alten Rahmens der FU-Fighters 2004. Relation zwischen Ebenen und Verhaltensmustern, Sensoren und Aktoren wurden der Übersichtlichkeit halber weggelassen.

gewichtete Mittel der Änderungswünsche und modifizieren dementsprechend den mit ihnen assoziierten Sensor einer Ebene niedriger.

Bei der letzten Variante werden innerhalb eines Codestücks der Ebene beliebige Berechnungen durchgeführt und an dieser Stelle der Sensor gesetzt. Diese Berechnungen dürfen auf Sensoren einer Ebene unterhalb der aktuellen zugreifen.

Die Abhängigkeiten sind somit limitiert auf Sensoren, die sich aus Sensoren der unteren Ebene errechnen, und Sensoren, die von einer oberen Ebene durch Aktoren gesetzt werden und Verhaltensmuster, die auf alle Sensoren und Aktoren der Ebenen der Agenten einer Höhe zugreifen (Abbildung 4.1).

Die verschiedenen Ebenen sollen mit steigender Höhe den steigenden Abstraktionsgraden entsprechen. Daraus resultierte die Idee, die höheren Ebenen mit geringerer Häufigkeit auszuführen, da abstraktere Entscheidungen längerfristig von Bedeutung sein sollten. Auf Grund dessen nimmt die Mittlung eines Sensorwerts bei deren Berechnung eine solche herausragende Stellung ein: Ein Sensor höherer Ebene sollte stabilere Werte halten. Aufgrund der geringeren Ausführungsfrequenz konnten dort auch komplexere Berechnungen durchgeführt werden, da dies sich nicht so gravierend auf die Laufzeit auswirkt. Aus der Sicht einer Ebene ergibt sich daraus folgende Ausführungsreihenfolge:

1. Hochreichen der Sensoren
   Zuerst werden in der Ebene die Sensoren berechnet, die sich aus den Sensoren der unteren Ebene ergeben. Dies geschieht entweder durch eine Berechnung auf der Ebene selbst oder durch Mittlung von Sensoren durch den SensorManager.

2. Ausführung der nächsthöheren Ebenen
   Gibt es eine nächsthöhere Ebene, so wird diese ausgeführt. Dadurch werden die restlichen Sensoren der aktuellen Ebene durch deren Aktoren gesetzt.

3. Aktivierungsfunktion der Verhaltensmuster
   Alle Sensoren der Ebene sind nun berechnet. Der VerhaltenManager lässt die Verhaltensmuster anhand der verschiedene Sensorwerte ihre Aktivierungsfunktion errechnen, und berechnet an Hand derer und der Hemmungen die verschiedenen Aktivierungsstärken.

4. Ausführung der aktiven Verhaltensmuster
   Ist ein Verhaltensmuster aktiv (Aktivierungsstärke $> 0$), wird die Zielfunktion ausgeführt. Aus den gegebenen Sensorwerten der Ebene errechnen wir den aktiven Verhaltensmustern neue Stellwerte für die Aktoren und geben sie den Aktoren bekannt. Die Aktoren bilden das durch die Aktivierungsstärke gewichtete Mittel dieser Stellwerte. Gibt es keine Stellwerte für einen Aktor, so wird der alte Wert beibehalten.

5. Herunterreichen der Aktorenwerte an die untere Ebene:
   Die Werte der Aktoren werden vom AktorManager an die mit den Aktoren assoziierte Sensoren der unteren Ebene weitergegeben.

## 4.3 Probleme des alten Rahmens

Im alten Rahmen sind Abstraktionsgrad und Reaktionszeit durch die Ebenen miteinander gekoppelt. Wollen wir in einem Verhaltensmuster die Fähigkeiten anderer Verhaltensmuster durch Aktoren nutzen, so muss dieses Verhaltensmuster in einer höheren Ebene liegen als alle diese Verhaltensmuster. Dies bedeutet, dass es dementsprechend seltener aufgerufen wird und mit gemittelten Sensoren arbeiten muss, was sich nachteilig beides auf die Reaktionszeit auswirkt. Die Motivation für die Verringerung der Aufrufsfrequenz ist die eingesparte Rechenleistung. Bei der vorhandenen Implementierung war die Einsparung zu vernachlässigen. Daher wurde auf diese Funktionalität verzichtet.

Nun kann man noch die Reaktionszeit des Verhaltens verringern, indem das Verhaltensmuster auf einen ungemittelten Sensor zugreift. Korrekterweise müsste dazu ein Sensor in der Ebene erzeugt werden, der sich aus der Mittlung eines Sensors niedrigerer Ebene errechnet, wobei die Mittlung ausgeschaltet wird. Was bedeutet, dass der Sensor einfach dupliziert wird. Da in allen Berechnungen theoretisch auf jeden Punkt in der Vergangenheit eines Sensors zugegriffen werden kann, muss die gesamte Geschichte gespeichert werden. Um die Echtzeitfähigkeit zu gewährleisten, wird am Anfang für jeden Sensor ein Puffer angelegt, der über die gesamte Spielzeit genügen muss. Somit ist die Duplizierung nicht nur stilistisch problematisch, sondern ist auch mit einem entsprechenden Speicherbedarf verbunden. Da jedes Verhaltensmuster Zugriff auf seine zugehörige Ebene hat, und diese Ebene wiederum Zugriff auf die nächstniedrigere hat, kann man auch die strikte Trennung zwischen den Ebenen umgehen und direkt auf den Sensor einer niedrigeren Ebene zugreifen. Dies widerspricht jedoch dem Konzept der Abstraktion durch die Schichten.

Ähnlich geartet ist das Problem bei den Aktoren: Ein gewünschter Aktor kann in einer niedrigeren Ebene als das neue Verhaltensmuster liegen. Das Verhaltensmuster selbst ist aber durch die Nutzung eines anderen Aktors auf dieser Ebene fixiert. Nun könnte man wieder den Aktor in der höheren Ebene duplizieren. Oder man setzt sich wieder über die strikte Trennung hinweg und greift einfach direkt auf den Aktor zu.

In beiden Fällen wurde zumeist der Direktzugriff aufgrund des geringeren Programmier- und Speicheraufwand gewählt.

Da ein Verhaltensmodul auf Sensoren und Aktoren als Komponenten einer konkreten Implementierung einer Ebene zugreift, ist es eng mit der Ebene gekoppelt: Verhaltensmodule können nur Klassenbestandteil von Exemplaren einer bestimmten oder von ihnen abgeleiteten Ebenen werden.

Das Problem verschärft sich dadurch, dass auch der Zugriff auf Sensoren und Aktoren anderer Roboter durch Zugriff auf Komponenten der Ebenen erfolgt. Da die Prüfung statisch durch den Compiler geschieht, müssen bereits die Basisklassen diese Sensoren und Aktoren als Komponenten enthalten, wenn ein Verhaltensmuster auch nur potentiell auf sie zugreift.

Ein weiterer Nachteil des Rahmens ist, dass die Berechnungen von Sensoren bis auf eine Mittlung von Werten aus der unteren Ebene nicht durch den Rahmen strukturiert

werden, sondern in die Ebenen codiert werden. Somit sind diese Berechnungen auf die Ebene und ihre Ableitungen begrenzt. Auch alle anderen Berechnungen, die sich nicht durch den Rahmen repräsentieren ließen, wurden als Funktionen der Ebene implementiert und waren damit an diese gebunden.

All dies machte eine Neukonzeption des Rahmens wünschenswert.

## 4.4 Graph von Sensoren als neuer Rahmen

Was muss nun beibehalten werden, damit nicht alles neu geschrieben werden muss? Wir brauchen weiterhin Ebenen-Funktionen, Sensoren, Aktoren und Verhaltensmustern mit Aktivierungs- und Ausführungsfunktionen, um das alte Verhaltensmuster nachbilden zu können.

Damit nicht weite Teile der Graphischen Benutzeroberfläche (GBO) neu geschrieben werden müssen, wäre es vorteilhaft, wenn aus ihrer Sicht die Schnittstelle weitestgehend unverändert bleibt. Die GBO greift über die Ebenen auf SensorManager und VerhaltenManager zu. Auf Ebenen können wir verzichten, da sie aus Sicht der GBO nur mittelbar dem Zugriff auf Sensoren und Verhaltensmuster dienen. Sensoren und Verhaltensmuster und ihre entsprechenden Manager scheinen notwendig. Agenten sind auch weiterhin eine sinnvolle Repräsentation des Spiels, der Teams und der Roboter. Damit sowohl der alte als auch der neue Rahmen funktioniert, leiten wir die neuen Sensoren von der Basis „Sensor“ des alten Rahmens ab.

Konzentrieren wir uns nun zuerst auf die Sensoren, denn Aktoren und Verhaltensmuster lassen sich ebenfalls als Sensoren darstellen, wie ich in Abschnitt 4.6 und 4.7 erläutern werde. Somit treffen die folgenden Erläuterungen auch auf sie zu.

Wenn wir Verhaltensmuster und Ebenen entkoppeln wollen, müssen wir auf andere Weise auf Sensoren zugreifen. Da die GBO über SensorManager auf Sensoren zugreift, bietet es sich an, hierbei denselben Weg zu gehen. Der SensorManager liefert den zu einem Namen registrierten abstrakten Sensor. Sollte zu dem Namen ein Sensor existieren, müssen wir noch überprüfen, ob der gefundene Sensor vom geforderten Typ ist. Die Auflösung des Namens und die Überprüfung des Typs kapseln wir in Konnektoren, die die Verbindung herstellen und als Stellvertreter der Sensoren dienen. Dies soll einmal vor Beginn der Ausführung des Verhaltens geschehen, da somit eine falsche oder fehlende Sensorverbindung deterministisch entdeckt wird, und die damit verbundene Arbeit im zeitkritischen Teil vermieden wird. Jeder weitere Aufruf ist dann eine einfache Indirektion über einen getypten Zeiger. Wie die Verbindung genau hergestellt wird, ist in Abschnitt 4.9 erläutert.

Eine weiteres Element des Konzepts für den neue Rahmen ist, dass jeder Sensor sich zu jedem Zeitpunkt aus anderen Sensoren selbst berechnen soll, anstelle von außen gesetzt zu werden. Man leitet hierzu einen neuen Sensor von der Basisklasse ab, und implementiert die Berechnungsfunktion. Der Zugriff erfolgt durch die oben genannten Konnektoren. Jedem Konnektor wird der Name des Sensors und der relative Zeitraum, auf den zugegriffen werden soll, übergeben. Da der Zugriff auf die Sensoren mittels der

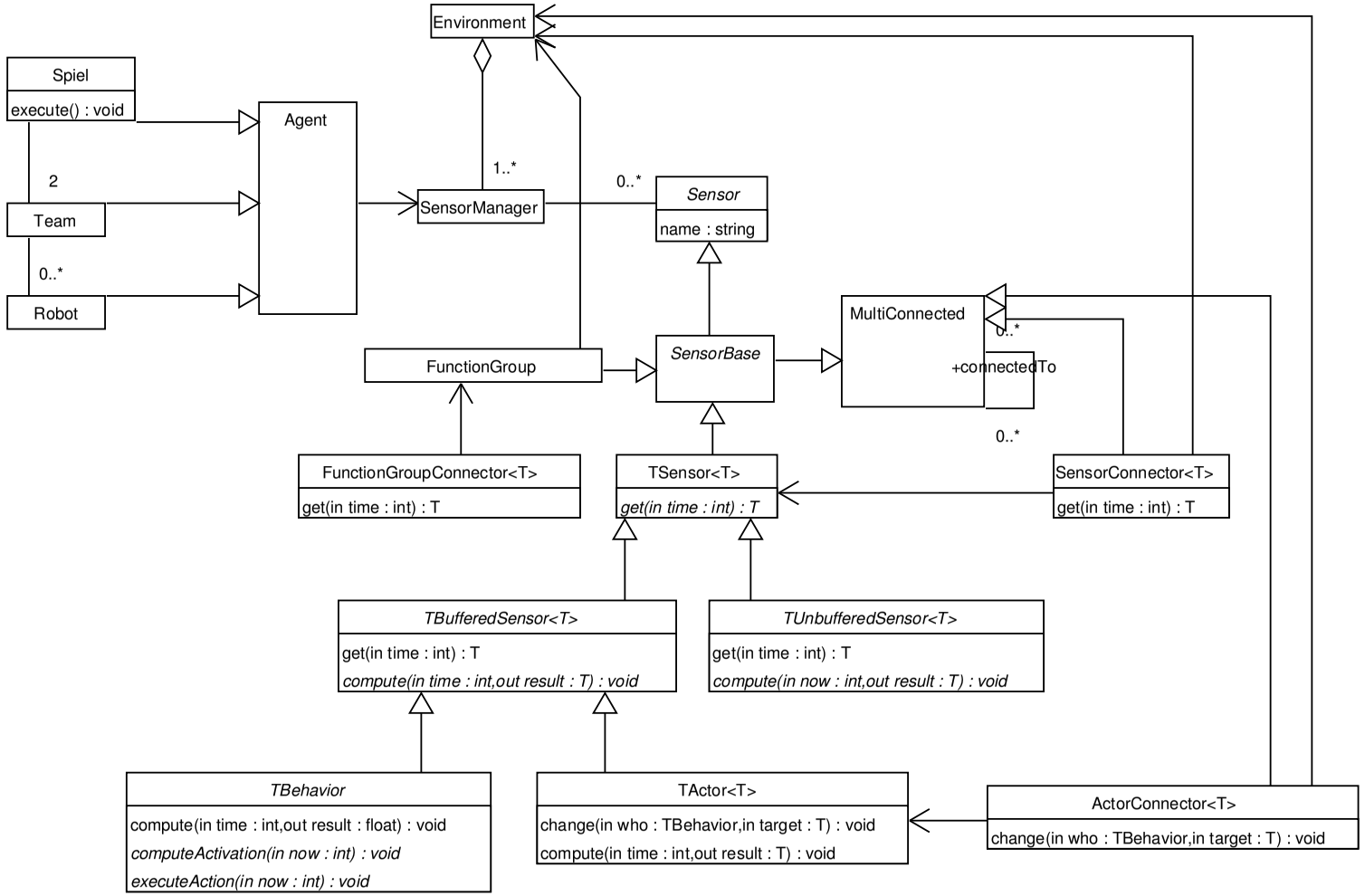


***Abbildung 4.5:*** Neues Rahmen der FU-Fighters. Die Agenten inkarnieren die von SensorBase abgeleiteten Sensoren, Verhaltensmuster, Aktoren und Funktionsgruppen und registrieren sie bei ihrem SensorManager. Nicht dargestellt wird die Relation der abgeleiteten Sensoren zu den Konnektoren. Der Abhängigkeitsgraph wird durch Exemplare der Klasse MultiConnected dargestellt.

Konnektoren erfolgt, können diese auch sicherstellen, dass der Zugriff innerhalb des vom Programmierer angegebenen zeitlichen Rahmen liegt. Es kann jedoch nicht überprüft werden, ob die Angabe des Zeitrahmens die striktest mögliche ist.

Durch diese Angaben ist klar, welcher Sensorwert sich aus welchen anderen Sensorwerten berechnen kann, womit die Datenabhängigkeiten explizit repräsentiert sind.

Würde man nun jeden Sensorwert bei jedem Zugriff neu berechnen und ein fester Anteil der Sensoren auf die Vergangenheit anderer Sensoren zugreifen, so würde die Rechenzeit schnell ins exponentielle ausufern. Schon daher ist es sinnvoll, Sensoren einzuführen, die ihre Berechnungen puffern. Wir unterteilen damit Sensoren in gepufferte und ungepufferte Sensoren.

Gepufferte Sensoren werden in jedem Zeitschritt berechnet, damit jede Stelle des Puffers einen gültigen Wert enthält. Somit wird jeder Wert genau einmal berechnet (Eager Evaluation). Da die Konnektoren den Zugriff auf die Vergangenheit eines Sensors zeitlich begrenzen, lässt sich die notwendige Puffergröße einschränken.

Ungepufferte Sensoren hingegen speichern die Werte nicht, sondern werden immer bei Bedarf berechnet (Lazy Evaluation). Damit ist bei ungepufferten Sensoren nicht gesichert, in welcher zeitlichen Reihenfolge die Berechnungen erfolgen. So kann es zum Beispiel sein, dass ein ungepufferter Sensor erst den Sensorwert zum Zeitpunkt $t$ berechnen soll und erst danach zum Zeitpunkt $t - 1$, also im Prinzip in umgekehrter Reihenfolge. Da die Reihenfolge, in der die Werte berechnet werden, beliebig ist, ergibt es nur wenig Sinn durch einen internen Zustand Werte von einer Berechnung zur nächsten zu übertragen. Vielmehr bietet dies eine schwer indentifizierbare Fehlerquelle, da das Fehlverhalten nur zufällig auftreten kann. Somit wird für einen ungepufferten Sensor gefordert, dass sich jede Rechnung nur aus Werten von anderen Sensoren zusammen setzt. Um dies zu erzwingen, wird die Berechnungsfunktion als konstant deklariert. Die Deklaration einer Funktion als konstant bewirkt in C++, dass in dieser nur nicht-modifizierende Operationen der Klasse aufgerufen werden können. Um in der Klasse einen Zustand zu speichern, muss aber Klasse modifiziert werden. Dies wird aber durch den Compiler abgefangen.

Gepufferte Sensoren hingegen werden für jeden Zeitschritt sequentiell berechnet. Daher ergibt es auch Sinn, auf einen internen Zustand einer letzten Berechnung zurückzugreifen. Die Berechnungsfunktion wird somit als nicht-konstant deklariert.

Der Programmierer muss nun die Vor-und Nachteile gegeneinander abwiegen und von der gewünschten Basisklasse ableiten. Hat der Sensor keinen internen Zustand, so lässt sich durch Wechsel der Basisklasse und Modifizieren der Deklaration der Berechnungsfunktion beliebig zwischen Gepufferten und Ungepufferten wechseln.

## 4.5 Datenabhängigkeitsgraph und Sequentialisierung

In welcher Reihenfolge sollen nun die Sensorwerte berechnet werden? Damit überhaupt eine Reihenfolge existieren kann, muss es sich bei dem Datenabhänigkeitsgraph um einen gerichteten azyklischen Graphen handeln. Somit liefert jede topologische Sortierung des

Graphen eine mögliche Sequentialisierung der Berechnungen. Nun könnte man fordern, dass bereits im Graphen der Sensoren keine zyklischen Abhängigkeiten existieren, nur ist diese Forderung strenger, als die des alten Rahmens und könnte Probleme bei der Übernahme der Verhaltensmuster zur Folge haben. Dies widerspricht also der Zielsetzung der Kompatibilität. Daher betrachten wir stattdessen die zeitliche Abhängigkeit der Daten, wie sie durch die Konnektoren repräsentiert wird, und überprüfen, ob der Sensor im gewünschten Bereich überhaupt berechnet werden muss. Handelt es sich also um einen gepufferten Sensor, so ist er für vergangenen Zeitpunkte bereits berechnet und eine Datenabhängkeit existiert nur in der Berechnung des aktuellsten Wertes.

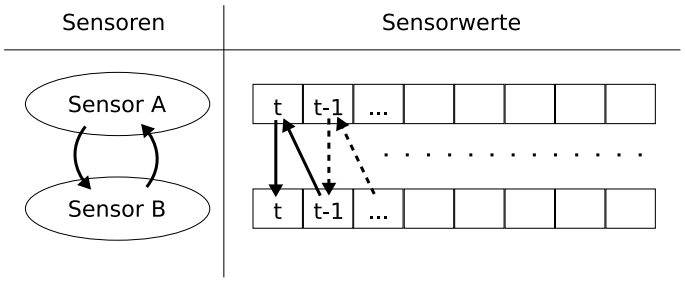


***Abbildung 4.6:*** Datenfluß zweier abhängiger Sensoren, für die eine sequentielle Berechnungsreihenfolge existiert

Betrachten wir als obige Abbildung 4.6 als Beispiel: Sensor $A$ und $B$ sind gepufferte Sensoren, und Sensor $A$ zum Zeitpunkt $t$ berechnet sich aus Sensor $B$ zum Zeitpunkt $t-1$ und Sensor $B$ aus Sensor $A$ zum Zeitpunkt $t$. Wir haben also einen Kreis in der Abhängigkeiten der Sensoren. Da jedoch der Sensor $B$ ein gepufferter Sensor ist, wurde er zum Zeitpunkt $t-1$ bereits berechnet und hält einen gültigen Wert. Damit löst sich in diesem Fall die Datenabhängigkeit auf.

## 4.6 Verhaltensmuster als Sensoren

Wie kann man nun mit Sensoren Verhaltensmuster repräsentieren? Die Behauptung ist, dass ein Verhaltensmuster sich als Spezialisierung von gepufferten Fließkommazahl-Sensoren mit einer Aktivierungsfunktion und Zielfunktion darstellen lässt, wobei der Wert des Sensors der Aktivierungswert ist. In der Berechnung des Sensorwerts wird zuerst mit Hilfe der Aktivierungsfunktion dieser Wert bestimmt, ist dieser größer Null, so wird die Zielfunktion ausgeführt.

Im alten Rahmen berechnet sich der Aktivierungswert aus den hemmenden Verhaltensmuster und der Aktivierungsfunktion, die sich wiederum aus den Sensoren der Ebene berechnet. Im neuen Rahmen kann ein Sensor sich nun aus beliebigen Sensoren berechnen, solange kein Kreis entsteht. Somit kann jede Aktivierungsfunktion auch im neuen Rahmen implementiert werden. Desweiteren kann für die Berechnung

der Hemmung auch auf den Aktivierungswert anderer Verhaltensmuster zugegriffen werden, da dies einfache Sensorwerte sind. Da schon im alten Rahmen Kreisfreiheit in der Hemmstruktur erzwungen wurde, ist die Forderung der Kreisfreiheit der Sensorwerte keine weitere Einschränkung und die Kompatibilität ist gewährleistet.

Wie sieht es aber mit der Zielfunktion aus? Die Zielfunktion berechnet sich aus den Sensoren der Ebene und setzt die Aktoren der Ebene. Da ein Verhalten ein Sensor ist, kann es Zugriff auf beliebige Sensoren erhalten. Somit können wir auch die Zielfunktion realisieren, wenn wir auf irgendeine Weise Zugriff auf die Aktoren erlangen. Dies muss an dieser Stelle vorausgesetzt werden, und wird bei der Realisierung der Aktoren in Absatz 4.7 erläutert.

Nun wäre es zwar möglich, jedes Verhaltensmuster als Sensor selbst zu implementieren, jedoch ist dies unnötig kompliziert. Eine Verkomplizierung liegt zum Beispiel in der Art, wie die hemmenden Verhaltenmuster festgelegt werden müssten und wie die Hemmung berechnet wird. Sensoren bestimmen durch die Wahl der Konnektoren selbst, aus welchen Sensoren sie sich berechnen. Jedoch wird bei hemmenden Verhaltensmustern für gewöhnlich zuerst das gehemmte Verhaltensmuster geschrieben und das Hemmende entsteht erst im Nachhinein. Es scheint aber nicht sonderlich elegant, zu fordern, dass bei jedem neu erzeugten Verhaltensmuster alle nun von ihm gehemmten Verhaltensmuster modifiziert werden müssen. Außerdem müsste so das Hemmen jedes Mal reimplementiert werden. Das spricht dafür, eine gemeinsame Basisklasse für Verhaltensmuster zu erzeugen, so dass nur noch die Aktivierungsfunktion und Zielfunktion selbst implementiert werden müssen. Diese werden dann von der Berechnungsfunktion aufgerufen.

Bei der Verhaltensklasse lässt sich nun jeder beliebiger Fließkommazahl-Sensor, wozu auch Verhaltensmuster zählen, als Hemmer registrieren. Das Verhaltensmuster erzeugt nun für die hemmenden Sensoren Stellvertreter (Inhibitor) und speichert sie ab. Ein Inhibitor stellt sich im Graphen als eine eingehende Verbindung dar und dadurch sind Abhängigkeiten wieder explizit repräsentiert und werden vom Rahmen auf Kreisfreiheit überprüft. Wie im alten Rahmen wird nun mittels der gespeicherter Inhibitoren die Hemmung in der Berechnungsfunktion berechnet.

Dadurch, dass Verhaltensmuster Fließkommazahl-Sensoren sind, ist auch die Darstellung der Verhaltensmuster durch die GBO erledigt. Sie werden genauso wie Fließkommazahl-Sensoren dargestellt. Wir können daher aus Sicht der GBO beim neuen Rahmen auf den VerhaltenManager verzichten. Nachteil hierbei ist, dass auf der GBO nicht mehr zwischen Verhaltensmuster und Sensoren unterschieden wird.

## 4.7 Aktoren

Aktoren sind Sensoren, die sich aus den verschiedenen Stellwerten und Aktivierungsstärken, der mit ihnen verbundenen Verhaltensmustern errechnen. Aktive Verhaltensmuster rufen beim Aktor eine Stellfunktion auf. Dies wird als Wunsch vom Aktor registriert. Diese Wünsche werden in der Berechnung des Sensorwerts ausgewertet.

Als Sensoren sind sie beim SensorManager registriert und lassen sich einfach mittels eines Sensor-Konnektors auslesen und die Stellwerte können in der GBO dargestellt werden. Nun führen wir noch Aktor-Konnektoren ein, die als Stellvertreter eines Aktors dienen, wodurch Verhaltensmuster verändernd auf die Stellfunktion zugreifen können.

Wie auch bei den Hemmungen werden hier die Abhängigkeiten zwischen Verhaltensmustern und Aktoren in umgekehrter Reihenfolge hergestellt, da nicht der Aktor bestimmt, durch welche Verhaltensmuster sich der Wert errechnet, sondern die eingehenden Verhaltensmuster.

Im Normalfall berechnet sich ein Aktor wie im alten Rahmen durch das gewichtete Mittel der anliegenden Wünsche. Wenn keine Wünsche anliegen, so wird im Gegensatz zum alten Rahmen nicht der alte Wert beibehalten, sondern der Aktor wird auf einen Standardwert zurück gesetzt. Dies soll dazu führen, dass dem Programmierer schneller auffällt, wenn er in einem Verhaltensmuster bei einer Fallunterscheidung vergisst, einen Aktor zu setzen.

Nun ist die Mittlung nicht immer sinnvoll oder möglich. So ist zum Beispiel der Mittelwert einer Aufzählung nicht definiert. Aktoren können die Änderungswünsche in ihrer Berechnungsfunktion in beliebiger Weise zu einem Ergebnis zusammenfassen. Ein Variante ist der TMinimumActor, welcher das Minimum der eingehenden Wünsche wählt. Dieser Aktor wird zum Beispiel bei der Bestimmung der maximal erlaubten Fahrgeschwindigkeit eingesetzt. Für Aufzählungen existiert ein Sensor, der die Wünsche nach einem Wahlverfahren zusammenfasst: Das Element mit den meisten Stimmen (summierten Aktivierungswerten) wird gewählt.

## 4.8 Weitere Arten von Sensoren

Ein Sensor besitzt zu einem Zeitpunkt nur ein Datum. Daher werden nun häufig die Ergebnisse einer Berechnung zu einem komplexen Datentyp zusammengefasst. Die GBO oder auch andere Leser sind aber eventuell nur an Teilen des Ergebnisses interessiert. In diesem Fall kann ein SensorAccessorSensor eingesetzt werden. Jede Abfrage eines Wertes wird auf den Aufruf einer Zugriffsfunktion (Accessor) des zeitgleichen Datums eines anderen Sensors zurückgeführt. Damit braucht der AccessorSensor keine eigene Pufferung, sondern stützt sich dabei vollständig auf den zugrunde liegenden Sensor.

Offen bleibt, wie die Daten in den Rahmen gelangen. Da diese Daten die Basis aller Berechnungen sind, müssen sie in gepufferten Sensoren gespeichert werden. Hierzu stehen verschiedene Sensoren zur Verfügung, die alle darauf basieren, auf verschiedene Weise in der Berechnungsfunktion aus einem anderen Speicherbereich die Daten in die Puffer zu kopieren.

In einer Implementation wird einfach ein Klassenbestandteil direkt kopiert, in einer anderen Variante wird mittels einer Zugriffsfunktion der Klasse der Wert kopiert. Dies sind die Methoden, nach denen die Daten den Elementen im alten Rahmen zur Verfügung gestellt wurden. Mittels dieser Sensoren lassen sich die Daten des alten Rahmens in den neue Rahmen einbinden.

Eine neue Variante ist der DoubleBufferedSensor. Die Klasse soll als Bestandteil einer Klasse dienen und kann wie ein Verweis auf das Datum behandelt werden. Alle Änderungen erfolgen dann auf einer „Hintergrund“-Variable. Diese wird atomar mit denen von allen anderen „Hintergrund“-Variable mit ihren entsprechenden „Vordergrund“-Variablen zu Beginn jedes Zeitschrittes ausgetauscht. Damit wird sichergestellt, dass die Daten der Sensoren konsistent bleiben.

## 4.9 Konnektoren und Umgebungen

Zu Beginn werden alle Sensoren des Spiel, der Teams und der Roboter erzeugt und bei ihren jeweiligen SensorManagern registriert. Die Reihenfolge der Erzeugung der Sensoren darf hierbei zwangsweise keine Rolle spielen, da es zyklische Abhängigkeiten zwischen den Sensoren geben kann, wie in Absatz 4.5 bereits erläutert wurde.

Im Spiel sind alle SensorManager zur GameEnvironment zusammengefasst. Da jeder Sensor bei einem dieser SensorManager registriert ist, kann sich ein Konnektor mit einer GameEnvironment jeden Sensor holen, vorrausgesetzt, ihm ist bekannt, welcher SensorManager diesen bereit hält. Will man auf einen Sensor im Spiel oder alle Sensoren mit denselben Namen in allen Robotern oder allen Teams zugreifen, so ist dies mit einer GameEnvironment bereits möglich.

Üblicherweise will man jedoch als Sensor eines Agenten auf einen anderen Sensor desselben Agenten zugreifen. Dazu sind die Informationen der GameEnvironment im Allgemeinen nicht ausreichend und werden somit im Spiel für die beide Teams um die jeweilige Identifikation des Teams zur TeamEnvironment, und im Team nochmals um die des Roboter zur RobotEnvironment erweitert. Dies geschieht durch Vererbung. Bei jeder Ableitung erhöht sich die Anzahl der einsetzbaren Konnektoren. Im Falle der TeamEnvironment ist es möglich, nach Sensoren „meines Teams” oder „aller Roboter meines Teams” zu fragen, bei der RobotEnvironment zusätzlich auch nach „meinem Roboter”.

Konnektoren dienen auch als Stellvertreter. Somit ist die Schnittstelle eines Aktor-Konnektors eine andere als die eines Sensor-Konnektors. Da aber ein Aktor auch ein Sensor ist, ist es immer noch möglich, auf ihn mittels eines Sensor-Konnektors wie auf einen Sensor zuzugreifen.

Gelingt es einem Konnektor nicht, den gewünschten Sensor zu finden, so gibt es verschiedene sinnvolle Reaktionen. Die strengste Variante erzwingt das Vorhandensein des gewünschten Sensors: Wird beim Verbinden der Sensor nicht gefunden, wird dies als fataler Fehler gewertet, und das Programm terminiert, nachdem der Benutzer benachrichtigt wurde. Dies wäre äquivalent zum alten Rahmen, wo das Verhaltensmuster durch den Typ der Ebenen das Vorhandensein der Sensoren forciert. Dies hat aber zur Folge, dass sich das Teamverhalten entweder nicht zusammen mit den alten Robotern einsetzten lässt, oder dass der Sensor nachträglich in den alten Robotertyp eingefügt werden muss, wenn neue Robotertypen neue Aktoren/Fähigkeiten oder Sensoren erhalten und ein Teamverhalten auf diese zugreift. Beim alten Rahmen hat dies zum

letzteren geführt, da eine Interoperabilität zwischen den Robotern erhalten bleiben sollte. Die Trennung zwischen den Agenten ist also nicht sauber.

Alternativ lässt sich auch festlegen, dass ein beliebiger Standardwert zurückgegeben wird, oder, im Falle eines Aktors, dass der Stellwert ignoriert wird, wenn ein Sensor nicht existiert. Die Entscheidung, welche Behandlung die richtige ist, hängt vom lokalen Kontext ab, und kann dort durch die geeignete Wahl des Konnektors getroffen werden.

Durch die Art der verwendeten Konnektoren in einem Sensor wird festgelegt, welche Art von Umgebung für den Sensor selbst erforderlich ist und somit auch dessen Zugehörigkeit.

Beispiel: Wollen wir in einem Sensor auf einen Sensor in „meinem Roboter" zugreifen, so nutzen wir einen „Mein Roboter"-Konnektor. Dieser erfordert eine RobotEnvironment. Der Konnektor erhält aber die Umgebung von diesem Sensor selbst, somit ist es für diesen selbst auch nötig, Zugriff auf eine RobotEnvironment zu erhalten. Solch ein Sensor kann nur in einem Roboter erzeugt werden, und gehört daher zu ihm.

## 4.10 Funktionsgruppen

Ein letzter offener Punkt ist, wie die beliebigen Funktionen der Ebenen realisiert werden können. Hierzu werden Funktionsgruppen verwendet. Funktionsgruppen sind ebenfalls von SensorBase abgeleitet. Der Name der Basis ist primär historisch bedingt, denn eine Funktionsgruppe ist strikt gesehen kein Sensor, da sie zu keinem Zeitpunkt einen bestimmten Wert hat, sondern weiter parametrisiert wird. Stattdessen ist eine Funktiongruppe eine Klasse mit beliebiger Schnittstelle, die sich aus verschiedenen Sensoren und gegebenen Parametern berechnet. Ein Konnektor zu einer Funktionsgruppe kann kein Stellvertreter mehr sein, da sie nicht jede beliebige Schnittstelle nachbilden kann. Ein Funktionsgruppen-Konnektor liefert einen Verweis auf die Klasse selbst zurück. Dadurch, dass die Funktionsgruppe von SensorBase abgeleitet ist, kann sie, wie Sensoren, mit Konnektoren auf Sensoren (und andere Funktionsgruppen) des Rahmens zugreifen, und ihre Abhängigkeiten werden modelliert.

### 4.10.1 Beispiel

Um das Zusammenspiel der Komponenten zu verdeutlichen, betrachten wir die Beispielaufgabe aus Kapitel 2: Der Roboter soll den Ball ins Tor spielen. Auf dem Weg zum Ball steht ihm jedoch ein Hindernis im Weg. Da der Rahmen auf dem Prinzip der hierarchischen reaktiven Verhaltensmuster aufbaut ist, ließe sich das Problem genauso wie im dazugehörigen Beispiel im Absatz 2.2.3.1 lösen. Wir wandeln die Realisierung etwas ab, um die Möglichkeiten des neuen Rahmens aufzuzeigen.

In Abbildung 4.7 sind alle beteiligten Elemente dargestellt. Die Kästen Spiel, Hindernis und Roboter sind die SensorManager mit ihren angemeldeten Einheiten. Die Verbindungen im Graphen entsprechen den Konnektoren. Da Sensoren sich selbst berechnen können, verzichten wir auf das Verhaltensmuster „Wähle Schussziel", welches

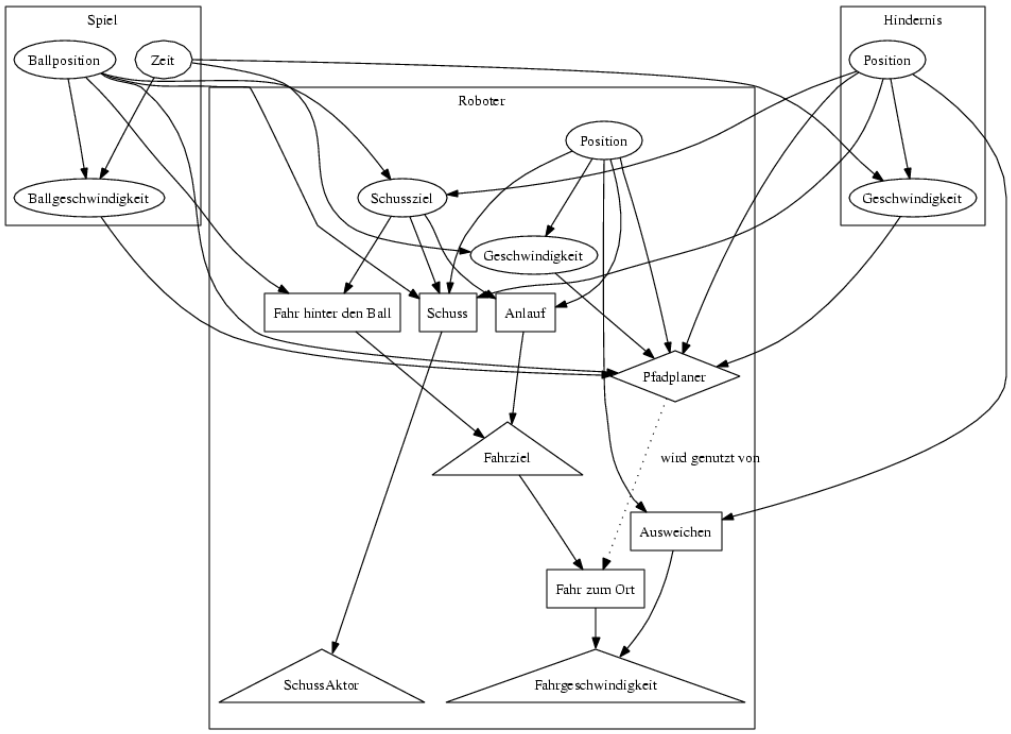


***Abbildung 4.7:*** Sensoren und Verhaltensmuster zur Realisierung des Problems: Sensoren sind rund umrandet, Aktoren dreieckig und Verhaltensmuster rechteckig. Pfeilrichtungen stellen den Datenfluss dar.

das Schussziel durch einen Aktor setzt. Stattdessen lassen wir den Sensor „Schussziel“ sich selbst berechnen. Eine weitere Abweichung ist die Nutzung eines Pfadplaners durch „Fahr zum Ort“. Dieses Verhaltensmuster überprüft nun, ob der Weg von der aktuellen Position zu dem gegebenen Fahrziel frei ist. Wenn nicht, so greift es auf den Pfadplaner zurück, um eine Ausweichposition zu bestimmen. Wie in Abbildung 4.8 zu sehen ist, generiert der Pfadplaner einen Weg um das Hindernis herum. Auf der Trajektorie wird ein Punkt gewählt, der anstelle des Fahrziels angefahren wird. Ist der Weg zum Ziel frei, so unterscheidet sich der Ablauf nicht weiter vom Beispiel der hierarchischen reaktiven Verhaltensmuster.

## 4.11 Graphische Benutzeroberfläche

Um die Fehlersuche zu vereinfachen, lassen sich die verschiedenen Sensorwerte zu den vergangenen Zeitpunkten in der GBO visualisieren. Die GBO kann über die Agenten und Sensormanager auf die verschiedenen Sensoren zugreifen.

In Abbildung 4.10 ist die Oberfläche zur Selektion des Sensors zu sehen. In der Tabelle befinden sich in einer Zeile drei Listenfelder. Im Ersten kann der Benutzer den

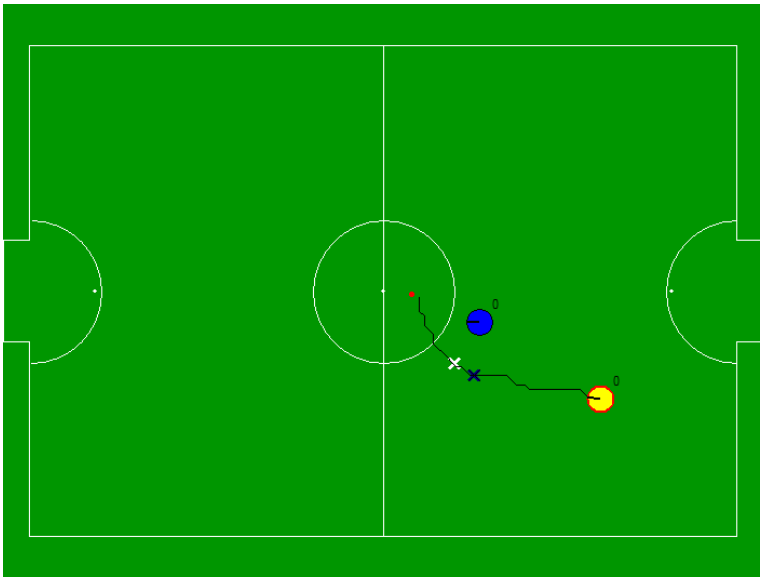

***Abbildung 4.8:*** Ausführung eines Schusses im neuen Rahmen: Aufgrund des Hindernisses wird ein grober Pfad zum Ziel (Ball) geplant. Auf dem Pfad wird ein anfahrbarer Punkt gewählt und stattdessen angefahren.

Agententyp auf Spiel, Team 0 oder Team 1 einschränken. Im Falle eines Teams kann im nächsten Listenfeld zwischen dem Team selbst oder einem seiner Roboter selektiert werden. Durch die beiden Listen ist der SensorManager bestimmt. Mittels der dritten Liste kann der Benutzer sich aus den Sensoren des SensorManagers einen auswählen. Durch den Typ der Daten, die ein Sensor speichert, bestimmt sich der Visualisierer. Dieser bietet dem Benutzer verschiedene graphische und textuelle Darstellungsmodi an. Die Graphischen sind in der GBO durch die Buchstaben in der Tabelle repräsentiert. Entsprechend des gewählten Modus werden die Daten des Sensors entweder als Kurve im unteren Teil des Fenster (siehe Abbildung 4.9) oder zweidimensional auf dem Feld visualisiert. So kann zum Beispiel ein Vektor, wie in Abbildung 4.9, als Kreuz wiedergegeben werden. Alternativ kann auch eine Komponente des Vektors oder dessen Betrag als Kurve veranschaulicht werden.

Die textuelle Darstellung erfolgt in der selben Tabelle wie die Auswahl des Sensors. Bei Anklicken des Textes werden dem Benutzer verschiedene textuelle Darstellungsmodi zu Auswahl gegeben. Winkel lassen sich zum Beispiel sowohl im Bogenmaß als auch in Grad, aber auch als normierter Vektor angeben.

Unterhalb der Tabelle zur Auswahl der Sensoren befindet sich zum Überblick eine Darstellung der gerade aktiven Verhaltensmuster eines gewählten Roboters.

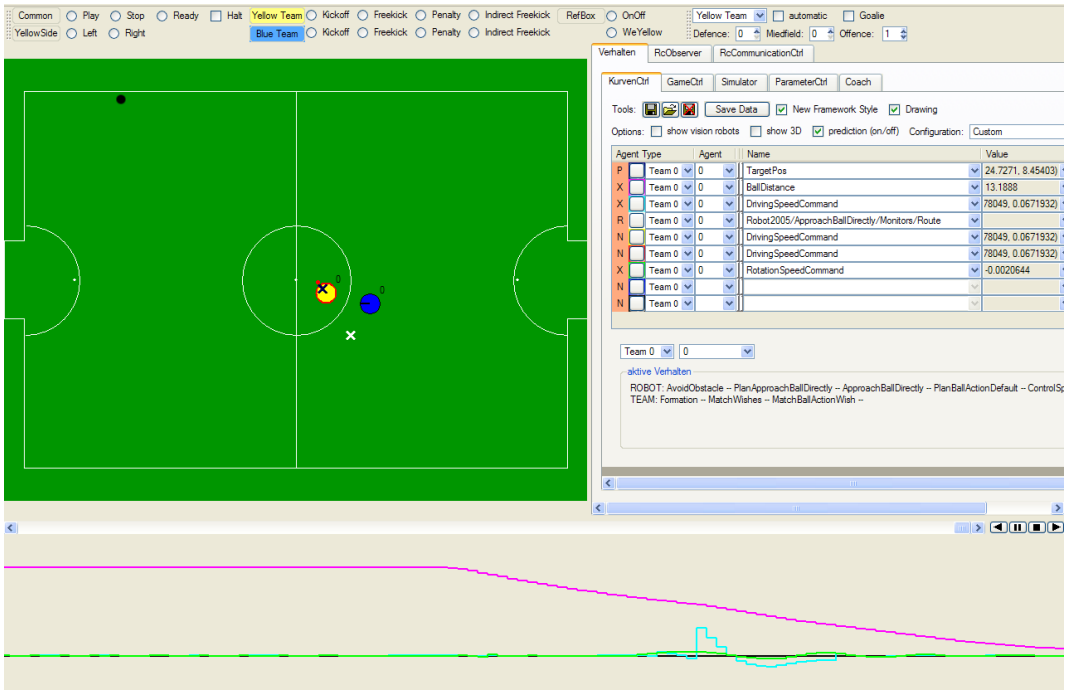


***Abbildung 4.9:*** Graphische Benutzeroberfläche des Verhaltenrahmens: Die obere Leiste dient zur Darstellung und Einstellung des Spielmodus (Anstoß, Freistoß, usw.). Rechts darunter befindet sich eine Oberfläche zur Wahl der Darstellung der Sensoren. Auf ihr werden die Sensorwerte textuell dargestellt. Links davon ist eine Repräsentation des Feldes mit dessen Objekten. Dort können verschiedene Sensoren dargestellt werden (z.B. wie das Kreuz auf dem Feld). Unten ist eine Fläche zur Darstellung von Sensorenwerte über die Zeit in Form einer Kurve.

## 4.12 Nichtsequentielle Ausführung des Verhaltenrahmens

Da es meist fünf nahezu unabhängig agierende Agenten gibt, ist zu erwarten, dass sich die Ausführung gut parallelisieren lässt. Wie in Absatz 4.5 erwähnt, ist jede topoligische Sortierung eine mögliche Ausführungsreihenfolge. Gibt es verschiedene mögliche Reihenfolgen, so ist das Problem nichtsequentiell und man kann annehmen, dass hier ein gewisses Maß an Parallelität vorhanden ist. Wie kann diese Parallelität aussehen? Ist eine Menge gepufferter Sensoren zum aktuellen Zeitpunkt bereits berechnet, dann können Sensoren, die nur auf diese Menge zugreifen, simultan auswertet werden. Dies liegt daran, dass das reine Auslesen der gepufferte Sensoren keine Nebenwirkungen hat. Praktisch ist dies bei allen gepufferten Sensoren der Fall. Nicht unbedingt jedoch bei Verhaltensmustern, denn diese greifen auch schreibend auf andere Sensoren (Aktoren) zu. Dieser Zugriff auf Aktoren muss also auf Mehrfädigkeit vorbereitet sein. Ebenso kann es zu Problemen beim Auslesen ungepufferter Sensoren kommen, da diese erst bei Bedarf berechnet werden. Aber auch diese können ohne größeren Aufwand auf Mehrfädigkeit vorbereitet werden.

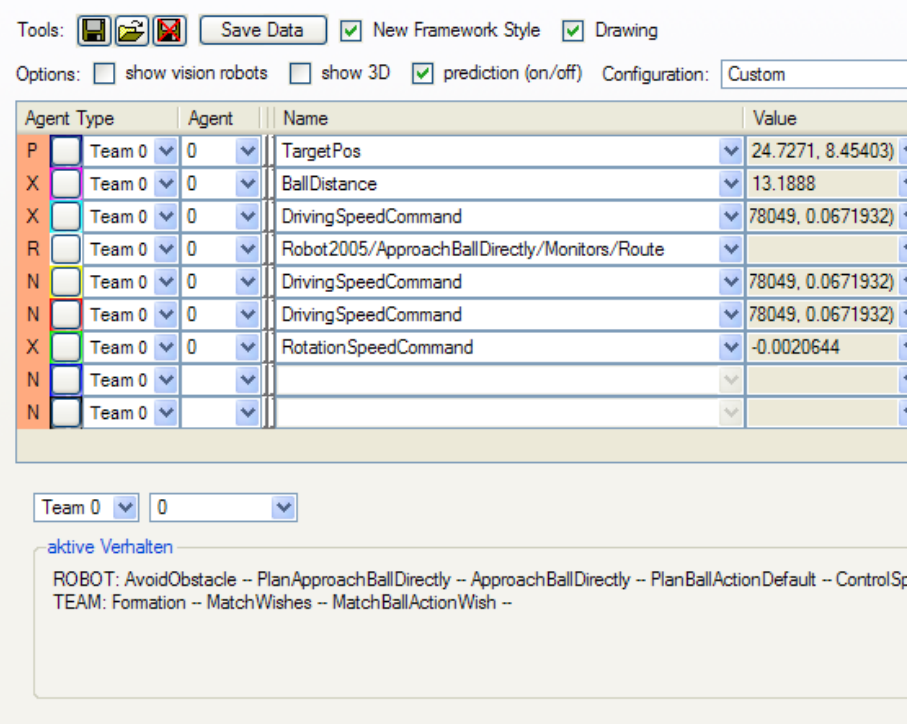


***Abbildung 4.10:*** Tabelle zur Auswahl der darzustellenden Sensoren. Der Buchstabe steht für Darstellungsart des Sensors. Farbig umrandete Kästchen bestimmen die Farbwahl. Es folgen Agenttyp, Agentennummer und Sensorname zur Wahl des Sensors. Zuletzt ist die textuelle Darstellung des momentanen Sensorwerts zu sehen.

Schließlich bleiben noch Funktionsgruppen aus. Diese lassen sich auf Grund ihrer Beliebigkeit nicht ohne weiteres allgemeingültig auf Mehrfädigkeit vorbereiten. Deswegen muss für sie ein gegenseitiger Ausschluss gefordert werden: Eine Gruppe von Sensoren, die unmittelbar oder mittelbar durch einen ungepufferten Sensor oder eine andere Funktionsgruppe auf dasselbe Exemplar eine Funktionsgruppe zugreift, darf nicht gleichzeitig berechnet werden.

Wir betrachten eine Menge von Sensoren als eine Aufgabe und beginnen damit, dass jeder Sensor eine Aufgabe ist. Definieren wir den Graphen über die Aufgaben in der Art, dass genau dann, wenn eine gerichtete Kante zwischen den Sensoren zweier Aufgaben existiert, auch eine Kante zwischen den beiden Aufgaben existiert. Wenn zwei Aufgaben nicht gleichzeitig ausgeführt werden dürfen, oder wenn es eine zirkuläre Abhängigkeit zwischen Aufgaben gibt, fasst man sie zu einer Aufgabe zusammen. Beachten wir diese Regeln, so erhalten wir am Ende einen kreisfreien Graphen der Aufgaben. Jede dieser Aufgaben induziert auf dem Graphen aller Sensoren einen Untergraphen. Dieser ist kreisfrei, und somit lässt sich die Aufgabe selbst sequentialisieren.

Die simpelste Variante diese Aufgaben zu erledigen, ist eine Art parallelisierten topologischen Sortierung: Wir halten eine Arbeitsliste mit allen Aufgaben, deren

Abhängigkeiten erfüllt sind und merken uns, wie viele Aufgaben noch ausstehen. Zu Anfang steht die Gesamtzahl aller Aufgaben aus und wir starten mit einer Liste aller Aufgaben, die zu keinem Zeitpunkt von anderen Aufgaben abhängen.

Solange es noch ausstehende Aufgaben gibt, entnimmt jeder Faden der Arbeitsliste eine Aufgabe und arbeitet sie ab. Anschließend verringert der Faden die Anzahl der ausstehenden Abhängigkeiten der Aufgaben, die von der momentanen Aufgabe abhängen um eins. Stehen bei einer Aufgabe keine Abhängigkeiten mehr aus, so kann diese an die Arbeitsliste angehängt werden.

Solange die Arbeitsliste leer ist, müssen wir auf eine neue Aufgabe warten.

In der Ausführung wird aktiv gewartet, da die Berechnung des Verhaltensrahmens höchste Priorität genießt, und so eine geringe Latenzzeit gewährleistet werden kann. Alternativ könnte die Rechenzeit lediglich für das Zeichnen der GBO oder Betriebssystemtätigkeit genutzt werden. Die Anzahl der Fäden muss hierbei aber auf die Anzahl der Prozessoren abgestimmt sein.

Der dadurch erzielte Geschwindigkeitsgewinn beläuft sich jedoch lediglich auf 18%. Dies kann auf folgende Gründe zurückgeführt werden:

a) Die rechnerisch aufwendigsten Aufgaben sind als Funktiongruppen realisiert und hängen eng zusammen. Durch die Zusammenfassung von Aufgaben wird der rechnerisch intensivste Teil nicht parallelisierbar.

b) Die parallelisierbaren Aufgaben sind sehr simple Berechnungen. Der Aufwand bei der Synchronisation im Zugriff auf die Arbeitsliste steigt in die Größenordnung dieser Aufgaben.

Hierbei muss bedacht werden, dass die Parallelisierung erst nachträglich getestet wurde. Die Implementierung der Funktionsgruppen wurden nicht mit bedacht auf Parallelisierbarkeit geschrieben.

# 5 Fazit und Ausblick

Die hier beschriebene Elektroniksoftware und der Verhaltensrahmen wurden zur Weltmeisterschaft in Osaka in der Small-Size- und der Middle-Size-Liga erfolgreich eingesetzt.

Problematisch haben sich bei der Regelung auf der Elektronik zwei Dinge erwiesen, die im Modell nicht weiter beachtet wurden: Schlupf und Motorenverschleiß. Schlupf wirkt sich negativ auf die Kontrolle aus, da die von der Elektronik wahrgenommene Bewegung von der tatsächlichen abweicht. Man könnte vermuten, dass durch die Überbestimmtheit des Gleichungsystems 3.1 sich eine solche Abweichung detektieren lässt. Bedauerlicherweise waren die Ergebnisse im allgemeinen Fall jedoch nicht eindeutig genug. Durch die Hinzunahme eines weiteren Sensors, des Gyroskops, besteht jedoch die Möglichkeit einer besseren Detektion des Schlupfs. Offen bleibt auch, wie am besten auf die Detektion des Schlupfs reagiert werden sollte. Durch die Überlastung der Motoren trat massiver Verschleiß an Motoren durch Überhitzung auf. Der Verschleiß ließe sich minimieren, wenn die Elektronik die thermische Belastung der Motoren modelliert und diese vor Überlastung schützt.

Der Verhaltensrahmen bietet durch die explizite Repräsentation der Datenabhängigkeiten dem Programmierer die Möglichkeit, sich durch eine graphische Darstellung der Abhängigkeiten selbst Klarheit über die Zusammenhänge der Module zu verschaffen. Durch die Kapselung der Berechnung in Sensoren kann das Verhalten besser strukturiert werden. Dadurch wird das System wieder verständlicher und leichter handhabbar. Desweiteren bietet der Datenabhängigkeitsgraph vom Prinzip her gute Möglichkeiten zur Parallelisierung. Der momentane Scheduler bietet die Möglichkeit, die Aufgabe mit den meisten Elementen als Abhängigkeitsgraph der Sensoren darzustellen. Hierdurch wird dem Entwickler ein Mittel an die Hand gegeben, schlecht parallelisierbare Komponenten zu identifizieren. Eine weitere Möglichkeit ist die Entwicklung eines anderen Schedulers. So könnte bei der Verteilung der Aufgaben Rücksicht auf die Caches der Prozessoren genommen werden. Eine andere Variante ist, man versucht den Mehraufwand des Schedulings zu verringern, indem man sehr simple Aufgaben zusammenfasst. Für die verschiedene Arten des Schedulings kann man auf den weiten Fundus der Methoden in Compilerbau und Rechnerarchitektur zurückgreifen.

Ein Nachteil des neuen Rahmens ist, dass durch den Verzicht auf Ebenen auch die Sichtbarkeitsgrenzen wegfallen. So kann nun jeder Agent auf jeden Sensor zugreifen. Auch wäre es interessant die Sensoren zur besseren Übersicht gruppieren zu können. Beides ließe sich mittels mehrerer SensorManager pro Agent realisieren. So könnte man einen Agenten mit einen öffentlichen und beliebig vielen nicht-öffentlichen SensorManagern ausstatten.

Ein anderer Punkt ist die Integrierung vom planenden Ansatz. Auch wenn das implementierte Verhalten nicht rein reaktiv ist, so dominiert doch die reaktive Komponente. Eine Konzept wäre, die Verhalten zur Planung zu nutzen, indem vom Planer den Verhaltensmustern Daten von hypothetische Situationen als Eingabe gegeben würden. Die Aktivierungsfunktion entscheidet nicht mehr, ob ein Verhalten aktiv werden soll oder nicht, sondern wird nur noch als Empfehlung für den Planer betrachtet. Dieser variiert jedoch diese Empfehlung und wertet das Verhalten dann entsprechend der Variation die Zielfunktionen aus. Dadurch wird eine Folgesituation generiert. Jeder Übergang ist durch die Aktivierungsstärken der Verhalten bestimmt. Der Planer generiert so eine Voraussicht. Ist dadurch absehbar, dass eine Sequenz zu besseren Ergebnissen führt, als das die rein reaktive Abfolge generieren würde, so wird diese Sequenz ausgeführt.

Die in dieser Arbeit beschriebene Steuerungsarchitektur wird für die neue Generation von FU-Fighters Roboter weiterhin verwendet und bei der Weltmeisterschaft 2006 in Bremen eingesetzt.

# Literaturverzeichnis


[1] Ronald C. Arkin. *Behavior-Based Robotics*. Bradford Books. MIT-Press, Cambridge, MA, USA, London, GB, 1998. 12

[2] S. Behnke, B. Frötschl, and R. Rojas. Using hierarchical dynamical systems to control reactive behavior. In *Proc. IJCAI '99 - International Joint Conference on Artificial Intelligence, The Third International Workshop on RoboCup*, pages 28–33, 1999. 15, 32

[3] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986. 12, 13

[4] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, June 1990. 7

[5] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *'Proceedings, IJCAI-91', Sydney, Australia.*, pages 569–595, San Mateo, CA, USA, 1991. Morgan Kaufmann publishers Inc. 11

[6] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, FOCS'87 (Los Angeles, CA, October 12-14, 1987)*, pages 49–60. IEEE, IEEE, 1987. 11

[7] J. F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award 1987, MIT Press, Cambridge, 1987, 1987. 11

[8] Freescale Semiconductor, Inc. *MC9S12DP256B Device User Guide V02.15*, January 2005. 21

[9] Erann Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial intelligence and mobile robots: case studies of successful robot systems*, pages 195–210, Cambridge, MA, USA, 1998. MIT Press. 18

[10] Ketill Gunnarsson, Wolf Lindstrot, Ra´ul Rojas, and Fabian Wiesel. Fu-fighters team description 2005. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *Proceedings of The 9th RoboCup International Symposium*, Osaka, Japan, 2005. 10

[11] Bastian Hecht, Alexander Gloye, Achim Liers, Marian Luft, Artem Petakov, Ra´ul Rojas, Mark Simon, Oliver Tenchio, and Fabian Wiesel. Fu-fighters small size 2005. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *Proceedings of The 9th RoboCup International Symposium*, Osaka, Japan, 2005. 10

[12] H. Jaeger and 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 (AROB '97), Beppu, Japan*, pages 76–79, 1997. 13

[13] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 35–48, Cambridge, MA, USA, 1990. MIT-Press. 12

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*, 1995. 8

[15] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991. 11

[16] S. Lenser, J. Bruce, and M. Veloso. A modular hierarchical behavior-based architecture. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001*, 2001. 16

[17] Alan K. Mackworth. On seeing robots. Technical Report TR-93-05, Department of Computer Science, University of British Columbia, 1993. 7

[18] William C. Messner and Dawn Tilbury. *Control Tutorials for MATLAB and Simulink: A Web-Based Approach*. Prentice Hall, 1999. 28

[19] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950. 7