Endbericht der Projektgruppe 416

PG 416

21.05.2003

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Die Projektgruppe 416 . . . 1

1.2 Roboterfussball am LS1 . . . 2

1.2.1 Das Hostsystem *robotsoccer* . . . 3

1.2.2 Die Roboter . . . 5

1.3 Übersicht . . . 6

2 Hostsystem **9**

2.1 Strategie . . . 9

2.1.1 Ausgangssituation . . . 9

2.1.2 Beschreibung der Testumgebung . . . 10

2.1.3 Durchgeführte Tests . . . 12

2.1.4 Korrekturen und Erweiterungen . . . 36

2.2 Die Bildverarbeitung . . . 42

2.2.1 Verbesserung des Trackings . . . 42

2.2.2 Test der Bildverarbeitung . . . 49

2.3 Funk . . . 58

2.3.1 Aufbau eines Datenpakets . . . 61

2.3.2 Befehle . . . 62

2.3.3 offene Tests . . . 64

2.4 Oberfläche und Bedienung 67
2.4.1 Neustrukturierung der Preferences 67
2.5 Allgemeine Probleme 71
2.5.1 Restrukturierung der Headerdateien 71
2.5.2 Compiler Warnungen, Debugging 74

3 Hardware **81**
3.1 Roboter 81
3.1.1 Aufbau 81
3.1.2 Ablaufsteuerung 84
3.1.3 Steuerung / Regelung 84
3.1.4 Ansteuerung und Abfrage der Hardware 88
3.1.5 Interrupt Verarbeitung 90
3.1.6 Funk 91
3.1.7 Durchgeführte Tests 92
3.1.8 offene Tests 97
3.2 Inbetriebnahme der Koreanischen Roboter 102
3.3 Kamera 104
3.3.1 Ausgangssituation 104
3.3.2 Eigenschaften einer neuen Kamera 104
3.3.3 Kamera- und Adapterkartentest 105
3.4 Spielfeld und Beleuchtung 110

4 Veranstaltungs- und Erfahrungsberichte **113**
4.1 Auftritt im Rahmen des JOY-Wettbewerbs 113
4.2 Auftritt im Rahmen der Aktionstage Dortmund-Project 114
4.3 GI-Tagung 115
4.4 Workshop 119

5 Offene Ideen und Konzepte **123**

5.1 Physikalisches Modell . . . 123

5.1.1 Abkürzungen und Konstanten . . . 124

5.1.2 Reibungskraft . . . 124

5.1.3 Zentripetalkraft . . . 125

5.1.4 Beschleunigung . . . 126

5.1.5 Beschleunigungskraft, Reibungskraft und Zentripetalkraft . . . 127

5.1.6 Kipp- und Standmoment . . . 127

5.1.7 Zwischenergebnisse . . . 128

5.1.8 Bedeutung für die Implementierung von Rampen . . . 129

5.2 Neue Pfadplanung über Kreisanfahrtsalgorithmus . . . 132

5.2.1 Die Nachteile der CMU-Anfahrtsplanung . . . 132

5.2.2 Der Algorithmus im Detail . . . 132

5.2.3 Hinderniserkennung und -meidung . . . 141

5.2.4 Aufteilung der Programmteile auf das DSP- und Hostsystem . . . 142

5.2.5 Stand der Umsetzung . . . 145

5.2.6 Erweiterungs- und Verbesserungsmöglichkeiten . . . 146

5.3 Roboter Mechanik . . . 149

5.4 Systematische Farbeinstellung mittels Graphen . . . 156

6 Services **157**

6.1 Webpräsenz . . . 158

6.1.1 Inhalte . . . 159

6.1.2 Struktur . . . 160

6.1.3 Design . . . 162

6.1.4 Redaktionssystem . . . 165

6.1.5 Ergebnis . . . 167

6.2 Arbeitsabläufe . . . 168
6.3 Dokumenten- und Informationsverwaltung . . . 169
6.4 automatische Quelltext Dokumentation . . . 170
6.5 Wiki . . . 172
6.6 Fehler- und Featureverwaltung . . . 172
6.7 Ausblick . . . 174

A Arbeitsumgebung und Werkzeuge **177**

A.1 Rechnerlandschaft . . . 177
A.1.1 IP-Addressbereich der Dortmund Droids am LS1 . . . 177
A.1.2 soccer . . . 177
A.1.3 fira . . . 178
A.1.4 kicker . . . 178
A.1.5 goalkeeper . . . 179
A.1.6 devon . . . 179
A.2 Eigenes robotsoccer-System installieren . . . 179
A.2.1 Linux installieren . . . 180
A.2.2 Grafikkartentreiber . . . 181
A.2.3 nVidia . . . 181
A.2.4 robotsoccer installieren . . . 182
A.3 Doxygen . . . 183
A.3.1 Warum doxygen ? . . . 183
A.3.2 Quick Guide . . . 183
A.4 Laufzeit Tools . . . 187
A.4.1 Einleitung . . . 187
A.4.2 Mpatrol . . . 187
A.4.3 ccmalloc . . . 189

A.4.4 LeakTracer 189

A.4.5 valgrind 190

A.5 Corporate Identity 191

B Checklisten **199**

Kapitel 1

Einleitung

1.1 Die Projektgruppe 416

Gegenstand dieses Endberichts ist die Arbeit der Projektgruppe (PG) 416. Während der Semester *SS02* und *WS02/03* befassten sich die Teilnehmer dieser PG mit dem Thema "Autonome, fussballspielende Roboter". Die Mittel hierzu wurden vom Lehrstuhl 1 der Universität Dortmund bereit gestellt.

Der Lehrstuhl 1 hat in diesem Zusammenhang bereits zum vierten mal eine Projektgruppe angeboten, die sich mit dem Thema Roboterfussball beschäftigen konnte. Die Arbeitsgrundlage der PG 416 war daher auch ein System, wie es aus den drei vorangegangenen Projektgruppen ([PG340, PG362, PG384]) sowie diversen Diplomarbeiten und Arbeiten von studentischen Hilfskräften, hervorgegangen ist.

Die Aufgaben der PG 416 waren unter anderem die Pflege und Erweiterung des bestehenden Systems, die Organisation von diversen Veranstaltungen, die Integration neuer Hard- und Software, die Überprüfung der einzelnen Komponenten des bestehenden Systems, die Erstellung von Dokumentationen, etwa zur Inbetriebnahme des Systems oder der Wartung der Roboter, sowie die Erstellung und Bereitstellung von Presse Material. Desweiteren wurde eine Webpräsenz[1] geschaffen, und diese im Verlauf der PG auf aktuellem Stand gehalten.

Die PG 416 nahm ihre Arbeit im Febraur 2002 auf. Alle Teilnehmer der PG hatten einen Seminarvortrag vorzubereiten, eventuell zu zweit, und diesen Vortrag dann im Rahmen einer Seminarfahrt, die Anfang April 2002 statt fand, den übrigen Teilnehmern zu präsentieren. Neben einer Einarbeitung

[1] `http://www.robosoccer.de/`

in das zu bearbeitende Thema, bot sich die Seminarfahrt auch dazu an, dass sich die Teilnehmer gegenseitig, sowie die Betreuer, näher kennen lernen konnten. Die Ergebnisse der Seminarfahrt sind im Seminarband der PG 416 ([SB416]) dokumentiert und werden hier nicht näher diskutiert.

Die PG 416 bestand aus folgenden Teilnehmern:

1. Michael Bunzel
2. Sven Bursch
3. Carsten Fenneker
4. Tim Hussein
5. Sung-Kwon Jung
6. Frank Karger
7. Christoph Michalski
8. Matthias Schweinoch
9. Johann Sladzcyk
10. Marco Wickrath

Betreuer der PG 416 waren:

1. Prof. Dr. Bernd Reusch
2. Dr. Norbert Jesse
3. Dr. Lars Hildebrand

1.2 Roboterfussball am LS1

Um den Einstieg und das Verständnis vor allem der Kapitel 3 und 4 zu erleichtern, wollen wir hier einen kurzen Überblick über das Roboterfussball System des LS1 geben. Für einen detailierten Einblick verweisen wir auf [PG340, PG362, PG384], sowie auf die jeweiligen Diplomarbeiten ([Klu01, Moh01, Wei03]).

Am LS1 wird (derzeit) ein 5-gegen-5 System betrieben. Die Randbedingungen, wie etwa Spielfeld- oder Robotermaße, genügen den Anforderungen der MiroSot Liga der FIRA ([FMS]).

1.2.1 Das Hostsystem *robotsoccer*

Die auf dem Computer ablaufende Software, das Hostsystem, heißt am LS 1 *robotsoccer*. Diese Software übernimmt diverse Aufgaben: Bildverarbeitung, Strategie, Steuerung der Roboter (per Funk), und im Weiteren stellt sie noch einen voll funktionsfähigen Simulator mit graphischer (OpenGL) Darstellung. Mit mittlerweile über 250 einzelnen Parametern ist die Software hochgradig konfigurierbar.

1.2.1.1 Die Bildverarbeitung

Momentan wird am LS1 eine NTSC Kamera eingesetzt. Es wird von der Bildverarbeitung nur jedes zweite Bild ausgewertet. In den Halbbildern (Abb. 1.1), die ausgewertet werden, wird zunächst nach Blobs gesucht. Blobs sind einfarbige Oberflächen, die eventuell einem Patch zugeordet werden können. Patches sind die farbigen Kennzeichnungen auf den Robotern[2].

Wurden die Blobs gefunden (Abb. 1.2, wird versucht, diese zu Patches zusammen zu führen: ein Patch besteht typischerweise aus einer Team- und einer Roboterfarbe (Abb. 1.3). Am LS1 wird derzeit eine Bildverarbeitung eingesetzt, die lediglich auf die Teamfarbe angewiesen ist. Die Roboter werden in dem System zu Beginn identifiziert und zugeordnet, und während des Spiels wird versucht, die einzelnen Roboter zu verfolgen, um einer Verwechslung vorzubeugen.

Aus den Patches lassen sich Position und Orientierung (zumindest der eigenen) Roboter gewinnen. Diese Information kann dann von der Strategie ausgewertet werden.

Für eine detailierte Betrachtung der Bildverarbeitung verweisen wir zusätzlich auf [Wei03].

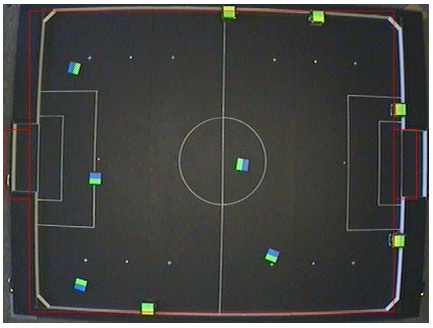


Abbildung 1.1: Capture Bild der Kamera

[2]Das Regelwerk der FIRA verbietet einige Farben für Patches, siehe [FMS]

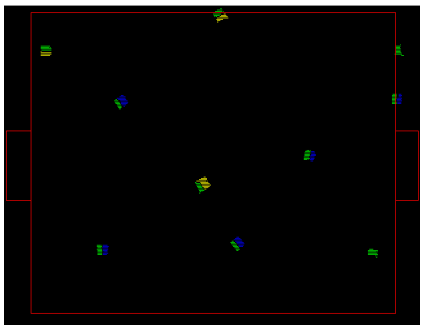


Abbildung 1.2: Von der BV erkannte Blobs

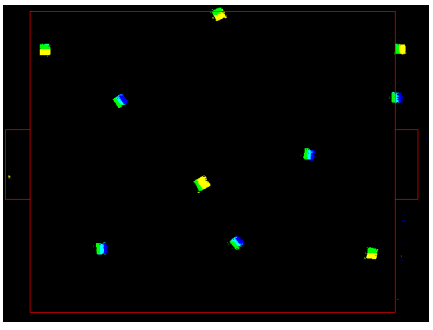


Abbildung 1.3: Filterergebnis der BV, erkannte Patches

1.2.1.2 Die Strategie

Die Strategie gliedert sich grob gesprochen in drei Ebenen. Wir unterscheiden *Spielzüge*, *Rollen* und *Handlungen*.

Die Spielzüge stehen dabei auf oberster Ebene. Spielzüge sind zum Beispiel *Angriff* oder *Verteidigung*. Für unterschiedliche Spielzüge gibt es unterschiedliche *Bewertungskriterien*. Der Spielzug, der die höchste Bewertung abhängig von der aktuellen Spielsituation erhält, wird ausgewählt.

Ein Spielzug, der ausgeführt wird, weisst den einzelnen Robotern[3] *Rollen* zu. Die Verteilung der Rollen erfolgt abhängig von der aktuellen Spielsituation - so spielen etwa die Positionen der einzelnen Roboter eine Rolle. Beispiele von Rollen sind etwa *Torwart*, *Verteidiger* oder *Stürmer*.

Haben alle Roboter ihre Rolle zugewiesen bekommen, so erfolgt eine Auswahl der *Handlung* für jeden Roboter abhängig von der ihm zugeteilten Rolle sowie der aktuellen Spielsituation. Für einen Roboter, der die Rolle des Stürmers hat, könnte dies also etwa bedeuten, dass ein Torschuss ausgeführt werden soll (wenn diese Handlung zur Situation passt).

1.2.1.3 Steuerung der Roboter

Die Strategie liegt fest, *was* gemacht werden soll. Jedem einzelnen Roboter wird dabei nach oben beschriebenem Verfahren eine Rolle zugewiesen, die wiederum eine Handlung abhängig von der Spielsituation festlegt. Die Handlung des Roboters führt schliesslich zu einem sogenannten *Zielvektor*. Dieser beinhaltet die Komponenten *Endgeschwindigkeit*, *Endausrichtung* und *Endposition*. Der Zielvektor legt dabei nicht fest, *wie* die Ziele zu erreichen sind; dies ist Aufgabe der Steuerung.

Die Funktionsweise der Steuerung selbst wird ausführlich in [PG362] besprochen. Weitere Informationen finden sich unter anderem auch in [Klu01].

1.2.2 Die Roboter

Bei den am LS1 eingesetzten Robotern handelt es sich um sogenannte *Differential-Drive* Modelle: die Roboter verfügen über zwei Räder, die jeweils von einem eigenen Motor angesteuert werden. Damit sind die Roboter in der Lage, neben Kreis- und Geradeaus Fahrten auch auf der Stelle (also um die eigene Achse) zu drehen.

Bestückt sind die Roboter unter anderem mit einem DSP der Firma *Texas Instruments*[4]. Angetrieben

[3] hierbei sind natürlich *unsere* Roboter gemeint
[4] Modell *TMS320F240*

wird ein Roboter von 2 Eletromotoren der Firma *Faulhaber*. Die Kommunikation mit dem Hostsystem *robotsoccer* erfolgt über Funk. Dazu sind die Roboter mit Funkmodulen der Firma *Radiometrix* bestückt, die zur bi-direktionalen Kommunikation fähig sind[5]. In Zukunft sollen die Roboter zudem mit CMOS Kameras ausgestattet werden, damit sie über ein eigenes Sichtsystem verfügen, um etwa selbständig den Ball verfolgen zu können.

Beim Antrieb der Roboter ergibt sich zwangsweise ein Problem: es ist bekannt, dass selbst baugleiche Elektromotoren bei gleicher Spannung nicht gleiche Drehzahlen liefern. Für Differential-Drive Roboter bedeutet dies in der Konsequenz, dass eine einfache Geradeaus Fahrt ohne weiteres nicht möglich ist: drehen die Räder verschieden schnell, gerät der Roboter auf eine Kreisfahrt. Um ein brauchbares Fahrverhalten herzustellen, werden daher auf dem Roboter durch PID Regler PWM Encoder realisiert.

Weitere Informationen zu den Robotern und den Komponenten sind in [Klu01] nachzulesen.

1.3 Übersicht

Kapitel 2 bespricht die Arbeit der PG 416 am Hostsystem *robotsoccer*. Im Vordergrund stehen dabei die Änderungen und Erweiterungen des Systems. Zusätzlich werden mögliche Erweiterungen und Tests, die im unmittelbaren Zusammenhang mit den jeweiligen Systemen stehen, besprochen.

Die Hardware umfasst unter anderem die Roboter selbst, sowie Kamera und Computer, die zum Betrieb nötig sind. Desweiteren gehören auch die Spielfeldbeleuchtung sowie das Spielfeld selbst zur benötigten Hardware. Kapitel 3 bespricht die Details der Hardware, aufgetretene Probleme mit der vorliegenden Implementierung auf den Robotern, sowie die Integration von neuer Hardware - etwa am Beispiel der Korea Roboter - in das System. Desweiteren werden weitere Tests im Bereich Hardware besprochen.

Im Verlauf der Projektgruppe wurden mehrere Veranstaltungen besucht, auf denen Roboterfussball vorort demonstriert wurde. Demonstrationsspiele wurden etwa auf der *JOY* sowie bei einer Veranstaltung des *Projekt Dortmund* gehalten. Desweiteren richtete die PG 416 das *FIRA Invitation Championship 2002* aus, ein Wettkampf, der während der GI Jahrestagung 2002 gehalten wurde. Hier traten die weltweit 6 besten Teams in der *MiroSot* Liga gegeneinander an. Diese Veranstaltungen, sowie der *Dortmunder Robotesoccer Workshop*, werden in Kapitel 4 besprochen.

Über den gesammten Zeitraum der Projektgruppe hinweg haben sich die Teilnehmer wöchentlich zu

[5] gegenwärtig erfolgt keine Rückmeldung der Roboter an das Hostsystem

Projektgruppensitzungen getroffen. Dies bot sich unter anderem als ein Forum für Ideensammlung an. Einige dieser Ideen wurden realisiert, andere stellten sich als (noch) nicht realisierbar heraus. Kapitel 5 behandelt dieses Thema ausführlich.

Um die Arbeit der Projektgruppe möglichst effizient zu gestalten, wurde unter anderem auf diverse Software Werkzeuge zurück gegriffen. So wurde als Dokumentationssoftware das Programm *Intrexx* eingesetzt, und *Typo 3* als Contentmanagement System für die Webpräsenz. Die Eigenheiten dieser und anderer Software Produkte, wie sie im Rahmen ihres Einsatzes durch die Projektgruppe aufgetreten sind, sowie die spezifische Einrichtung durch die Projektgruppe, wird in Kapitel 6 besprochen.

Abschließend findet sich im Anhang eine Sammlung von Kurzanleitungen, etwa zur Inbetriebnahme des Roboterfussball Systems, zur Montage des Spielfeld Überbaus, oder auch zur Rechnerlandschaft im Projektgruppen Raum. Dieser Anhang ist hauptsächlich als Nachschlagwerk angedacht, kann aber auch zum schnellen Einstieg in das Roboterfussball System am LS1 genutzt werden.

Kapitel 2

Hostsystem

2.1 Strategie

2.1.1 Ausgangssituation

Die Strategiekomponente des Robotsoccer-Projektes basiert auf einem, in drei Ebenen gegliedertem, Spielzugmodell. Über die genauen Gründe sowie Vor- und Nachteile dieser Entscheidung gibt der Endbericht der PG 384 Auskunft ([PG384]). Um jedoch dem Leser das unnötige "Blättern" zu ersparen hier eine Erläuterung der Grundzüge dieses Modells.

Wie bereits erwähnt ist das System in drei Entscheidungsebenen unterteilt. Die Grundidee, das Strategie-Problem zu lösen basiert auf einem top-down Prinzip. Jeder Ebene liegen Informationen vor zur Ausrichtung, Position sowie Geschwindigkeit von allen relevanten Spielfeldobjekten, zu denen auch die eigenen Roboter, die Gegner und der Ball zählen.

Die oberste Ebene, die Spielzugebene, entscheidet über die Spielzüge die vorgenommen werden müssen. Dies können normale Spielzüge wie z.B. Angriff oder Verteidigung sein, aber auch Sonderspielzüge wie z.B. Anstoß oder auch Strafschuss. Um den geeigneten Spielzug zu bestimmen, prüft jeder der Spielzüge die aktuelle Spielsituation und bewertet diese. Die Bewertungsfaktoren sind von Spielzug zu Spielzug unterschiedlich. Der Spielzug dessen Bewertung am höchsten ausfällt ist der am meisten geeignete für diese Spielsituation.

Der gewählte Spielzug verteilt an die eigenen Roboter Aufgaben, sog. Rollen. Die Rollen die hier vergeben werden unterscheiden sich abhängig von dem ausgewählten Spielzug. Der Spielzug Angriff verteilt z.B. aggressivere Rollen als *NeutralEigeneHealfte*. Die Roboter bekommen ihre Rolle aufgrund

des aktuellen Spielgeschehens zugewiesen. Es findet eine Prüfung statt welcher Roboter eine spezifische Rolle aus dem Rollenrepertoire des aktiven Spielzuges am besten erfüllen kann. Hier endet auch die Arbeit der Spielzugebene. Es wird nun die Rollenebene aktiviert.

Die folgende Ebene ist wie schon erwähnt die Rollenebene. Die in der Spielzugebene verteilten Rollen verfeinern die Analyse des Spielgeschehens und teilen aufgrund dieser dem Roboter einzelne Handlungen zu. In der Rollenebene werden Faktoren analysiert wie z.B. die Lage des Balles an der Bande oder die relative Position des Roboters zum Ball. Nach der Auswahl der für eine Rolle geeigneten Handlung wird die "Kleinarbeit" an die Handlungsebene weitergereicht.

Die "unterste" Ebene wird Handlungsebene genannt. Hier werden die konkreten Berechnungen angestellt, die es dem Roboter ermöglichen im Spiel überhaupt teilzunehmen. Dazu gehören Kalkulationen wie z.B. der richtige Anfahrtswinkel des Roboters zum Ball (z.B. im Falle einer Angriffshandlung) oder auch die optimale Torwartposition. In dieser Phase ist die Arbeit des Strategiemoduls beendet. Die berechneten Soll-Positionen der Roboter werden an die Robotersteuerung weitergereicht.

Wie man unschwer erkennen kann ist das Entscheidungssystem modular aufgebaut. Diese Tatsache ermöglicht gezieltes arbeiten an bestimmten Modulen ohne die Gesamtfunktionalität des ganzen Systems zu gefährden. Dieser Vorteil wird durch die Notwendigkeit einer genauen Abstimmung der einzelnen Module untereinander erkauft.

Genau diese Abstimmung schien jedoch etwas aus dem Gleichgewicht geraten zu sein. Bei der Analyse des Spielverlaufes legte das Gesamtsystem nämlich in einigen Situationen ein äußerst merkwürdiges und nicht vorhersagbares Verhalten an den Tag. Da dieses Verhalten durch die Beschreibung der einzelnen Module aus dem Projektgruppen Endbericht der PG 384 ([PG384]) nicht erklärt werden konnte, wurde schnell klar, dass ein ausgiebiger Software-Test der gesamten Strategie unumgänglich ist.

2.1.2 Beschreibung der Testumgebung

Das Strategie-Modul besteht bekanntlich aus den Entscheidungsebenen:

1. Spielzug-Ebene, die den Robotern Rollen zuweist,
2. der Rollen-Ebene, die Handlungen zuweist,
3. und einer Handlungsebene, in der die einzelnen Handlungen implementiert sind.

Unsere Vorgänger haben uns eine Schnittstelle im Code hinterlassen (`Gesamtsystem.h2`), in der man mit einer einfachen Code-Änderung die einzelnen Komponenten der Ebenen einbinden und einzeln ausführen kann. Weiterhin existiert der Logger, der die Bewertungen der Spielzugebene, und die daraufhin zugewiesenen Rollen und Handlungen aufzeichnet.

Aufgrund der Gesamtarchitektur des Systems, ist die funktionalität des Loggers nur im Betrieb des gesamten Systems gegeben, woraufhin man die Entscheidungen der einzelenen Spielzüge und Rollen nur manuell im laufenden Betrieb untersuchen kann. In der Testphase wurden die einzelnen Komponenten jeder Entscheidungsebene untersucht, um anschließend auf dieser Basis das Bewertungssystem des ganzen Moduls zu Analysieren.

Die Entscheidungshierarchie wurde von unten nach oben, d.h von der Handlungsebene bis zur Spielzugebene abgearbeitet. Mit Augenmerk auf die Aufgabe jeder Komponente wurde bei allen Klassen ersteinmal die Implementierung unter die Lupe genommen. Diese wurde auf logische Fehler, einheitliche Benutzung globaler Konstanten wie auch der Bibliotheken und die korrekte Umstellung auf 5-gegen-5 Roboter überprüft.

Nach dem der Code einer Komponente untersucht wurde, wurde diese dann in das System eingebunden und im Betrieb getestet. Hierfür musste man alle Situationen manuell kreieren. Bei den Handlungen wurden dann eingefügte Logausgaben mit dem Verhalten der Roboter verglichen. Zudem wurde mit verschiedenen Einstellungen der DSP-Parameter gearbeitet. Bei den Rollen und Spielzügen wurde schließlich die Bewertung und die korrekte Vergabe der Handlungen und Rollen überprüft.

Um zu prüfen, ob die Module ihre Aufgabe fehlerfrei erfüllen, wurde der Aufbau der Tests in vier Bereiche unterteilt. Die *Beschreibung* gibt die Intention der Autoren wieder. Die *Funktionsweise* beschreibt in welcher Weise diese umgesetzt wurde. Die *Bewertung des Moduls* ist ein Versuch, zu beurteilen ob die angestrebten Ergebnisse auch erzielt wurden. Der Punkt *Verbesserungsvorschläge* ist selbsterklärend.

In vielen Fällen ist für das Verständnis der Funktionsweise sowie der Verbesserungsvorschläge eine tiefere Einsicht in die internen Mechanismen der Strategie notwendig. Entsprechende Erklärungen bzw. Verweise finden sich im den Tests folgenden Glossar. Den Anfang machen die Handlungen. Dies erschien die richtige Vorgehensweise, da ein Testen der Rollen und sogar Spielzüge, ohne Gewissheit zu haben ob die tiefeliegenden Handlungen einwandfrei funktionieren, sinnlos erschien.

2.1.3 Durchgeführte Tests

2.1.3.1 Handlungen

AnstossTick

Beschreibung: Diese Handlung wurde für den eigenen Anstoß entwickelt. Da es laut den Regeln nicht erlaubt ist den Ball sofort in die Gegnerhälfte zu spielen, wird der Ball vom ausführenden Roboter in Richtung eigener Hälfte angetickt, von wo aus dann ein zweiter Spieler einen Angriff starten kann.

Funktionsweise: Der ballanstoßende Roboter fährt 0,35s mit der halben Maximalgeschwindigkeit in Richtung eigene Hälfte, dann mit der Maximalgeschwindigkeit nach hinten um den folgenden Angreifer Platz zu machen. Dabei werden die Radgeschwindigkeiten manuell gesetzt.

Bewertung: Diese Handlung funktioniert.

Verbesserungsvorschläge: Die Handlungslaufzeit wird über die Aufrufe der Strategie ermittelt (eine Konstante in Konstanten.cpp). Dies resultiert in verschieden Laufzeiten der Handlung, abhängig von tatsächlichen Aufrufen der Strategie. Die Systemzeit wäre da geeigneter.

Apos

Beschreibung: Angriffspositionierung. Positioniert den Roboter als zweiten Angreifer. Die Idee die dahinter steht ist, im Falle von einem nicht erfolgreichen Angriff, einen Roboter auf einer Position stehen zu haben, von der aus ein sofortiger zweiter Angriff möglich ist.

Funktionsweise: Zuerst findet eine Prüfung statt, ob der Ball oberhalb oder unterhalb der horizontalen Spielfeldmitte liegt. Die Zielposition des Roboters wird in die andere (horizontale) Hälfte gelegt. Die Zielpositionen werden aus der Konstanten.cpp ausgelesen. Sollte der Ball sich zusätzlich in einem Rechteck mit der Kantenlänge von 100 vor dem gegnerischen Tor befinden, werden die Zielpositionen weiter in Richtung Gegentor um den Faktor Konstanten.AposAngriff verschoben. Weiterhin wird geprüft ob im näheren Umkreis der Zielposition sich ein fremder Roboter befindet, sollte es der Fall sein wird die Zielposition verschoben um Kollisionen zu vermeiden.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Ballaufenthalt für tornähere Zielposition erweitern (x=hälfe des Strafraumes y=volle Spielfeldlänge). Zielposition abhängig von gegnerischen Robotern machen. Fremdrobotererkennung auf der Zielposition verschiebt diese nur minimal, Roboter kollidieren trotzdem. Kollisionsvermeidung?

AufdenBall

Beschreibung: Der Roboter soll auf die Position des Balles fahren und diesen dann wegstoßen.

Funktionsweise: Die Handlung schickt den Roboter mit der maximalen Geschwindigkeit auf die aktuelle Ballposition. Die RoboterSollPosition wird auf die aktuelle Ballposition gesetzt. Der Ausgangswinkel wird aus der direkten Verbindung dieser Punkte berechnet. Es findet keine Zukunftsberechnung statt. Bei Zielpositionen innerhalb des eigenen Strafraumes wird die x-Koordinate außerhalb des Strafraumes und die Geschwindigkeit auf 0 gesetzt, d.h. der Roboter wartet außerhalb des Strafraumes.

Bewertung: Handlung funktioniert. Kollisionsvermeidung funktioniert bei geringen Geschwindigkeiten. Bei höheren Geschwindigkeiten versagt sie wegen dem Anfahrtsproblem.

Verbesserungsvorschläge: Die Strafraum-Vermeidung in eine externe Bibliothek auslagern.

BallBlocken

Beschreibung: Diese defensive Handlung soll den Roboter zwischen dem Ball und dem eigenem Tor positionieren. Sollte ein Gegner den Ball vor dem Abfangpunkt erreichen so wird es berücksichtigt und die Bahn des Balles um die Reflektion an dem gegnerischen Roboter korrigiert. Falls sich der Ball im direkten Roboterweg befindet, wird er umfahren um Eigentore zu verhindern.

Funktionsweise: Der Ballabfangpunkt wird errechnet. Die zukünftige Ballposition wird über die ZunkunftsBerechnung-Bibliothek errechnet. Eine andere Bibliothek (reihenfolge) ermittelt die Reihenfolge in der alle Roboter den Ball erreichen würden. Diese Funktionalität wird hier benutzt um herauszufinden, ob ein eigener oder ein gegnerischer Roboter den Ball zuerst erreicht. Das umfahren des Balles wird durch eine einfache Zielpositionsverschiebung erzielt.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Es findet eine lokale In-den-Strafraum-nicht-einfahren Berechnung statt.

BallFuehren

Beschreibung: Wie schon der Name vermuten lässt, soll diese Handlung den Ball kontrolliert vor dem Roboter her schieben.

Funktionsweise: Zu erst wird unterschieden ob der Ball eine bestimmte Mindestgeschwindigkeit unterschritten hat. Sollte das der Fall sein, wird die sollRoboterposition auf die zukünftige Ballposition gesetzt und mit maximaler Geschwindigkeit angefahren. Im Falle einer Ballgeschwindigkeit die höher als dieser Wert dieser Mindestgeschwindigkeit (Konstanten.cpp minvxball) ist, wird versucht die Robotergeschwindigkeit an die Ballgeschwindigkeit anzupassen.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Es finden lokale In-den-Strafraum-nicht-einfahren Berechnungen statt. Es werden lokale Konstanten statt der aus der Konstanten.cpp benutzt. Ein langsamer Ball wird mit der maximalen Geschwindigkeit angefahren, besser wäre eine entfernungsabhängige Lösung. Den Teil überarbeiten, der die Gewichtung zwischen Ballzukunftsberechnung und der aktuellen Ballposition, für die Anfahrtsposition der Phase2 bestimmt. Die sollRoboterausrichtung wird immer auf 0.0 gesetzt. Dieses noch mal überdenken.

BallvonBande

Beschreibung: Die Handlung soll einen am der Bande liegenden Ball zuerst anfahren um ihn dann von der Bande wegzustoßen. Die Richtung des "Bandenkicks" wird von der Bande bestimmt. Sollte sich der Ball an der eigenen Torbande befinden wird er über eine Reflektion an der vom Tor abgewandten Bande in Richtung gegnerisches Tor befördert. An den Längsbanden geschieht es analog. Bei einer Balllage an der gegnerischen Torbande wird versucht den Ball parallel zur Bande zu spielen um einen Torschuss zu ermöglichen.

Funktionsweise: Die gesamten Berechnungen für "BallvonBand" werden nur für die obere Spielfeldhälfte getätigt, die korespendierenden Werte für die unteren Spielfeldabschnitt werden durch einfache Spiegelung errechnet. Zuerst wird erkannt an welcher Bande sich der Ball befindet. Dies geschieht durch Positionsvergleiche des Balles. Dann startet die eigentliche Handlung. Diese arbeitet in zwei Phasen. Als erstes wird geprüft ob der Roboter in der korrekten Ausrichtung am Ball steht, sollte es nicht der Fall sein wird die Anfahrtsphase aktiviert. Die Phase bringt den Roboter in eine Position, die diese Bedingung erfüllt. Wenn es dann so weit ist wird die zweite

Phase, die “drehkick” Phase, gestartet. Diese dreht den Roboter, entsprechend seiner Ausrichtung, auf der Stelle um einen bestimmten Winkel und befördert den Ball in die gewünschte Richtung.

Bewertung: Handlung funktioniert bedingt. Man kann beobachten wie sich die Roboter an den Banden festfahren. Da diese Handlung sehr präzise arbeiten muss, hängt das u.a. mit den Ungenauigkeiten in der Bildverarbeitung zusammen. Durch den Fischaugeneffekt treten Positionsverschiebungen verstärkt in den Bandenregionen auf.

Verbesserungsvorschläge: Es wurde überlegt diese Handlung neu zu schreiben. Aufgrund der Tatsache, dass demnächst eine neue Kamera angeschafft wird, wurde diese Entscheidung jedoch verschoben.

Enthaken

Beschreibung: Diese Handlung wird von einem HiWi (Hendrik Valentin) gepflegt und weiterentwickelt. Wir haben hier von einem Test abgesehen.

FreistossTorwart

Beschreibung: Das ist die Torwarthandlung die in der Standardsituation “FreisossDie” zum Zuge kommt.

Funktionsweise: Die Handlungsweise des Torwartes lässt sich durch die Ausrichtung zur Torlinie beeinflussen. Es wird zwischen zwei verschiedenen Fällen unterschieden. Falls der Torwart parallel zur Torlinie aufgestellt wird, agiert er als passiver Torhüter. Es wird der voraussichtliche Schnittpunkt des Balles mit der Torlinie mithilfe der Zukunftsberechnungs-Bibliothek berechnet dann wir der Torwart an diese Position gefahren. Der Zweite Fall tritt dann ein wenn der Torwart orthogonal zur Torlinie steht. Hier wird die Routine des aktiven Torwartes aktiviert. Hier wird die aktuelle Ballposition als zukünftige Roboterposition gesetzt.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Zur Zeit wird die Standardtorwarthandlung, TorwartAbwehr, neuimplementiert. FreistossTorwart evtl. durch diese ersetzten.

FreistossWir

Beschreibung: Angreifer-Handlung für die Standardsituation "FreistossWir". Nach dem Anpfiff des Freistosses wird versucht ein Tor zu erzielen.

Funktionsweise: Die Handlung erkennt den gegnerischen Torwart und versucht den Roboter in die offene Ecke zu fahren. Sollte der Torwart mittig stehen wird zufällig zwischen rechts und links entschieden. Die Kurvenfahrt des Roboters wird über das setzten der Radgeschwindigkeiten erzielt. Zusätzlich werden DSP-Parameter übertragen um die optimale Traktion zu gewährleisten. Die Handlung wird so lange ausgeführt bis der eigene Roboter im gegnerischen Tor ist oder bis die Handlungszeit überschritten wird.

Bewertung: Bei der Ausführung der Handlung fährt der Roboter erst ein Stück zurück (abhängig von der Ausrichtung) und dann erst nach vorne. Dieses Problem hängt wahrscheinlich mit der DSP-Parameterübertragung zusammen. Beim Weglassen dieser und einer manuellen Einstellung der DSP-Rampen trat das Problem sehr selten, und falls, dann wesentlich weniger ausgeprägt auf.

Verbesserungsvorschläge: Der Roboter erfährt eine zu starke Beschleunigung, dadurch ist die vorgegebene Kurvenfahrt nicht durchzuführen. Das äußert sich dadurch, dass der Roboter geradeaus fährt obwohl er eine Kurven fahren will (d.h. die Radgeschwindigkeiten werden unterschiedlich gesetzt). Handlung evtl. durch den Torschuss ersetzten.

GegnerBlocken

Beschreibung: Eine Abwehrhandlung. Der Abwehrroboter wird zwischen dem Ball und dem ballnächsten gegnerischen Roboter positioniert. Diese Handlung soll Pässe zwischen den Gegnern unterbinden und für allgemeine Störung des gegnerischen Spielflusses sorgen.

Funktionsweise: Zu erst wird mithilfe der "Reihenfolge"-Bibliothek der ballnächste Gegner bestimmt. Der optimale Abfangpunkt wird durch die Berechnung der kürzesten Strecke zwischen dem eigenen Roboter und dem Gegner-Ball Vektor bestimmt. Es folgt eine Prüfung ob der Gegner diesen Punkt schneller erreichen kann als wir. Sollte das der Fall sein, wird der Abfangpunkt solange in Richtung Ball verschoben bis wir die ersten sind oder der Punkt auf dem Ball liegt. Sollten wir uns schon auf dem Vektor Gegner-Ball befinden, wird unserer Roboter in, bis zu einem bestimmten Mindestabstand, Richtung Gegner gelenkt, um sich dann in dessen Weg querzustellen.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Bis auf die lokale Strafraum-nicht-einfahr-routine keine. Sauber programmiert!

Geradeaus

Beschreibung: Eine Handlung die den Roboter simpel geradeaus in die x-Richtung des Balles fahren lässt. D.h. sollte das "Heck" des Roboters dem Ball zugewandt sein fährt er geradeaus rückwärts, sonst vorwärts.

Funktionsweise: Es wird der Winkel zum Ball bestimmt und dann die Radgeschwindigkeiten manuell entsprechend auf Maximalgeschwindigkeit vorwärts oder rückwärts gesetzt.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Radgeschwindigkeiten werden manuell gesetzt.

Klaeren

Beschreibung: Die Handlung "Klaeren" ist dazu gedacht worden um den Ball aus eigener Hälfte und vor allem aus dem Torbereich wegzustoßen.

Funktionsweise: Der Roboter wird auf die zukünftige Ballposition geschickt. Im Falle, dass der Ball sich direkt hinter dem Roboter befindet, wird dieser umfahren.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Lokale Strafraum-nicht-einfahr-routine.

Mittelstuermer

Beschreibung: Diese Handlung stellt den Roboter auf eine Position aus der ein Angriff von der Mittellinie günstig erscheint.

Funktionsweise: Positioniert den Roboter 40cm hinter dem Ball (zwischen dem Ball und dem eigenem Tor) mit der Ausrichtung auf das Gegentor. Sollten sich gegnerische Roboter in der Nähe der Zielposition befinden, oder der Roboter zu nah am eigenen Tor sein, wird die Zielposition korrigiert.

Bewertung: Handlung funktioniert.

Verbesserungsvorschläge: Die Kollisionsvermeidung ist ungenau. Zuerst wird geprüft ob sich ein Gegner im Umfeld der Zielposition befindet, falls das der Fall ist wird die Gegnerposition (!) zur Zielposition gemacht und dann korrigiert um eine Kollision zu vermeiden.

Npos

Beschreibung: Handlung positioniert den Roboter auf der Mittellinie in dem Maximalen Abstand zu allen gegnerischen Robotern. Ziel ist den Roboter eine Position zu geben aus er gleichermaßen gut in den Angriff und in die Verteidigung gehen kann.

Funktionsweise: Es wird von sieben vorbestimmten Positionen auf der Mittellinie der Abstand zu allen Gegnern berechnet. Die Position die dem meisten Abstand zu allen gegnerischen Robotern hat wird zu Zielposition.

Bewertung: Handlung wurde ursprünglich für das 3-gegen-3 Spiel geschrieben und nicht an die 5-gegen-5 Variante angepasst. Handlung funktioniert nicht!

Verbesserungsvorschläge: Handlung auf 5-gegen-5 umschreiben. Globale Konstanten benutzen.

Erklärungen: Handlung wurde umgeschrieben.

Pass

Beschreibung: Wie der Name schon vermuten lässt, dient diese Handlung dazu einem weiter vorne stehenden Roboter den Ball zuzupassen um dann einen Angriff zu starten.

Funktionsweise: Es wird ermittelt welcher von unseren Robotern am weitersten vorne steht, dann wird versucht den Ball auf 2/3 des Entfernungsvektors des Angreifers zum Gegentor zu spielen.

Bewertung: Handlung war nicht zu testen, da es erhebliche Anfahrtsprobleme gab. Dadurch wurde jegliches “Zielen” unmöglich. Außerdem scheint es so, dass der ausführende Roboter immer versucht dem Roboter mit der Nr.4 zu passen.

Verbesserungsvorschläge: Es gibt eine eigene Zukunftsberechnung in der Handlung. Lieber die globale benutzen. Evtl. Routinen zum Passen aus dem Torschuss übernehmen. Fehlersuche (Passen auf Nr.4). Im Quelltext werden nicht gesetzte IDs nicht abgefangen. PassID=-1 abfangen. Der

Entfernungsvektor (bzw. die 2/3 davon) wird falsch berechnet. Dadurch entstehen z.T. unsinnige Positionen. Die Wahl des Zielroboters sollte überdacht werden, nicht immer ist der Roboter mit der größten x-Koordinate der geeigneteste um einen Pass zu empfangen (z.B. in den gegnerischen Ecken). Massiv überarbeiten!!!

PassiverGegnerBehindern

Beschreibung: Diese Handlung soll den zweiten Angreifer der Gegner daran behindern einen erfolgreichen Angriff durchzuführen.

Funktionsweise: Zunächst werden der gegnerische Torwart und der ballnächste Gegner, also der aktive Angreifer, bestimmt. Der Zielroboter dieser Handlung ist der, von den übriggebliebenen Gegner, der dem Ball am nächsten ist. Unsere Zielposition wird dann auf die Mitte des Vektors "Gegner-unser Tor" gelegt. Die Ausrichtung ist orthogonal zum Vektor.

Bewertung: Handlung wurde ursprünglich für das 3-gegen-3 Spiel geschrieben und nicht an die 5-gegen-5 Variante angepasst. Handlung funktioniert nicht!

Verbesserungsvorschläge: Handlung auf 5-gegen-5 umschreiben. Fehlerhafte Vektorberechnung korrigieren. Nicht-in-den-Strafraum-einfahren in eine Bibliothek auslagern.

Erklärungen: Handlung wurde umgeschrieben.

Torschuss

Beschreibung: Die Handlung "Torschuss" dient dazu den Ball ins gegnerische Tor zu befördern und damit im Endeffekt das Spiel zu gewinnen.

Funktionsweise: Zunächst wird die Position des gegnerischen Torwartes bestimmt, abhängig hiervon und von der Position des Balles wird über eine aufwendige Abfrage der Zielpunkt des Torschusses ermittelt. Aus diesem Punkt sowie dem mittels Zukunftsberechnung bestimmten Punkt, an dem der angreifende Roboter den Ball erreichen kann, ergibt sich schließlich der Anfahrtwinkel.

Bewertung: Handlung funktioniert.

Verbesserungvorschläge: Keine.

TorschussBlocken

Beschreibung: Die Intention von "TorschussBlocken" ist, den Versuch des Gegners, ein Tor direkt zu schießen, zu stören. Der eigene Roboter wird zwischen dem Ball und dem eigenen Tor platziert. Es wird darauf geachtet, dass unserer Roboter durch diese Aktion keine Eigentore schießt.

Funktionsweise: Unserer Roboter wird mittig in dem Verkor "Ball-eigenes Tor" positioniert. Es findet eine lokale Strafraum-verhinderung statt. Sollte sich der Ball zwischen dem handelnden Roboter und unserem Tor befinden, wird die y-sollPosition um einen Sicherheitsabstand korrigiert.

Bewertung: Handlung funktioniert bedingt.

Verbesserungsvorschläge: Nicht-in-Strafraum-einfahren funktioniert sehr schlecht. Externe Bibliothek benutzen. Ballumfahrfunktion verschiebt die Zielposition in y-Richtung unabhängig von der Entfernung zur Zielposition. Bei größeren Entfernungen reicht die Verschiebung nicht aus um den Ball zu umfahren. Manchmal ist die Verschiebung gar nicht nötig. Man sollte lieber mit Vektoren arbeiten statt simple x-vergleiche zu benutzen. In einigen Fällen (Ball über oder unter dem Strafraum) werden für den Roboter Zielpositionen außerhalb des Spielfelds gesetzt. Evtl. Problemursache ist das Ballumfahren. Sollte die Roboterzielposition innerhalb des Strafraumes liegen und der Ball umfahren werden, wird die Zielposition immer weiter verschoben und liegt dadurch irgendwannmal außerhalb des Spielfeldes. Handlung überarbeiten.

Torwartabwehr

Beschreibung: "Torwartabwehr" ist die standard-Torwarthandlung. Um den Gegner daran zu hindern Tore zu erzielen wird der Torhüter entsprechend der Ballposition und Geschwindigkeit im Tor platziert. Bei langsamen Bällen in der Tornähe ist es auch vorgesehen, dass der Torwart, um ein Tor zu verhindern, diesen dann wegkickt.

Funktionsweise: Zuerst wird geprüft ob der Ball sich im eigenen Strafraum befindet und falls, ob sich dieser auch langsam bewegt. Die Geschwindigkeit hierfür wird in der Konstanten.cpp festgelegt (VBallLangsam). Sollte dieser Fall eintreten versucht der Torwart den Ball wegzuschießen. Es wird die relative Position des Balles zum Torwart berücksichtigt und versucht Eigentore zu verhindern. Befindet sich der Ball ober- bzw. unterhalb des Strafraumes macht der Torwart "die Ecken zu" d.h. er fährt jewals in die obere bzw. untere Ecke des Tores. Falls der Ball sich schnell

oder nicht in Richtung unseren Tores bewegt, nimmt der Torwart den zukünftigen Schnittpunkt des Balls mit der Torlinie an. In diese Berechnung fließen auch Bandenrefflektionen mit ein.

Bewertung: Handlung funktioniert bis auf die Eigentorverhinderung.

Verbesserungsvorschläge: Eigentorverhinderung muss überarbeitet werden. Die Torinnenbanden werden bei der Anfahrt nicht beachtet, das führt dazu, dass der Torwart sich im eigenen Tor verfängt. Die "bei langsamen Ball wegschießen" -Routine um Gegnererkennung erweitern (nicht aus dem Tor rausfahren wenn Gegner in der Nähe). In der Handlung findet eine eigene Zukunftsberechnung statt. Die globale Bibliothek benutzten! Da es sich hier um eine sehr wichtige Handlung handelt haben wir uns entschlossen diese neuzuimplementieren.

Vorstopper

Beschreibung: Diese Handlung soll den eigenen Roboter in eine optimalen Vorstopperposition bringen.

Funktionsweise: Die Vorstopperposition wird auf eine relativ aufwendige Weise berechnet. An dieser Stelle wird empfohlen in den kommentierten Quelltext zu schauen. Eine einfache Erläuterung der Funktionsweise erscheint hier nicht unproblematisch.

Bewertung: Handlung funktioniert. Es treten Probleme bedingt durch die Anfahrt und die Zielausrichtung auf.

Verbesserungsvorschläge: Es findet eine lokale Strafraum-verhinderung statt.

ZweiterTorwart

Beschreibung: Die Handlung "ZweiterTorwart" wurde für einen vorgezogenen Verteidiger konzipiert. Es wurde eine (virtuelle) zweite Torlinie außerhalb des Strafraumes gezogen, auf der der Roboter als ein zweiter Torwart fungiert.

Funktionsweise: Die zweite Torwartlinie ist in der Konstanten.cpp festgelegt (ZweiterTorwartLinie). Der "ZweiterTorwart" funktioniert analog zu "Torwartabwehr" bis auf die Erkennung und Ausführung von "Ecken zumachen". Um die Funktionsweise zu garantieren wurde ein zweiter "Strafraum" bestimmt. Daher auch das fast identische Verhalten zur "Torwartabwehr". Zusätzlich wird verhindert dass der eigene Roboter in den Strafraum einfährt.

Bewertung: Handlung funktioniert nach Konstantenkorrektur. Weitere Bewertung sehe "Torwartabwehr".

Verbesserungsvorschläge: Siehe Torwartabwehr. Konstante ZweiteTorwartLinie lag innerhalb des Strafraumes!!! Wurde korrigiert. Durch die Tatsache dass die Torwarthandlung neu geschrieben wird stellt sich die Frage inwiefern Verbesserungen hier noch zu machen sind. Eine Alternative wäre den neuen Torwart zu modifizieren.

2.1.3.2 Rollen

Schon bei erster Besichtigung wurde schnell klar, dass die Rollenebene nicht so differenziert ist wie die Handlungsebene. Viele der hier vorhandenen Rollen dienen nur dazu Entscheidungen von der Spielzugebene in die Handlungsebene zu übertragen. Insgesamt gibt es 18 Rollen von denen zwei zu Testzwecken geschrieben wurden. Von den restlichen 16 sind tatsächlich nur drei dazu konzipiert verschiedene Handlungen zu vergeben. Die übrigen Rollen können nur eine Handlung "wählen" d.h. sie reichen die vom Spielzug getroffene Entscheidung an die Handlung weiter.

AktiveAbwehr

Beschreibung: Die Rolle, wie der Name schon sagt, ist für die Handlungsverteilung in der aktiven Abwehr zuständig.

Funktionsweise: Zuerst wird geprüft ob der Roboter an der Bande steht. In diesem Falle wird ihm die Handlung BallVonBande zugewiesen. In sonstigen Fällen wird, wenn ein gegnerischer Roboter den Ball innerhalb der "MaxFahrzeit" (Konstanten.cpp) nicht erreichen kann, die Handlung "BallBlocken" vergeben. Sollte es nicht der Fall sein (d.h. der Gegner erreicht den Ball) wird "Klearen" (im Falle dass unserer Roboter schneller am Ball ist als der Gegner) oder "Gegnerblocken" (Gegner ist schneller und der Ball befindet sich zwischen unserem Roboter und dem eigenem Tor) bzw. "BallBlocken" (Gegner ist schneller und der Ball ist nicht zwischen dem eigenen Roboter und unserem Tor) vergeben.

Bewertung: Alle Handlungszuweisungen, bis auf das erste "BallBlocken" funktionieren. Es ist unklar wann für den schnellsten der gegnerischen Roboter die Zeit zum erreichen des Balles=999 gesetzt wird.

Verbesserungsvorschläge: Die gesamte Abfrage ist sehr undurchschaubar, dadurch ist die Intention des Autors schwer erkennbar. Evtl. neustrukturieren.

Erklärungen: Falls die "Funktionsweise" nicht ganz klar geworden ist hier ein (vereinfachter) Quelltext:

```
if (Ball an der Bande) {
  waehle Handlung = BallVonBande
  return;  //beende Rollenverteilung
}
if (Zeit die der schnellste Gegner zum Ball braucht ==
  konstante.MaxFahrzeit) {
  waehle Handlung = BallBlocken
}
else {
  if (Ich schneller am Ball als der schnellste Gegner)
  waehle Handlung = Klaeren
  else
  {
    if (Ball zwischen mir und dem eigenem Tor)
    {waehle Handlung = GegnerBlocken}
    else
    {waehle Handlung = BallBlocken}
  }
}
```

AngriffMitBall

Beschreibung: Auch hier ist der Name selbsterklärend. Diese offensive Rolle dient dazu einen Angriff zu starten um evtl. ein Tor zu erzielen.

Funktionsweise: Zuerst wird überprüft ob der Ball an einer Bande liegt. In diesem Fall wird die Handlung BallvonBande ausgewählt. Sollte die Bandenprüfung negativ ausfallen, wird die eigene Position mit der Torschussgrenze (Konstanten.cpp) vergleichen. Befindet sich der Roboter jenseits dieser Grenzen bekommt er die Handling "Torschuss" zugewiesen. Ist unserer Spieler nicht nah genug am gegnerischem Tor, es befindet sich aber ein anderer Spieler vor uns dem zugepasst werden kann (d.h. seine Position ist richtig und die Passbahn ist frei) wird die Handlung "Pass" aktiviert. In anderen Fällen kommt "BallFuehren" zum Zuge.

Bewertung: Handlungszuweisung funktioniert. Bei der Betrachtung des Spielers zu dem gepasst werden kann ist die Vektorberechnung (wie schon in der Handlung "Pass") fehlerhaft. Der Bandentest ist lokal. Die globale Bibliothek benutzen.

AngiffsPositionierung

Beschreibung: siehe Handlung Apos auf Seite 12

Funktionsweise: Hier wird dem Roboter die Handlung "Apos" zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

AufdenBall

Beschreibung: siehe Handlunge AufdenBall auf Seite 13

Funktionsweise: Hier wird dem Roboter die Handlung "AufdenBall" zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

Enthaken

Beschreibung: siehe Handlung Enthaken auf Seite 15

Funktionsweise: Hier wird dem Roboter die Handlung "Enthaken" zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

FreistossTorwart

Beschreibung: siehe Handlung FreistossTorwart auf Seite 15

Funktionsweise: Hier wird dem Roboter die Handlung "FreistossTorwart" zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

FreistossWir

Beschreibung: siehe Handlung FreistossWir auf Seite 16

Funktionsweise: Hier wird dem Roboter die Handlung "FreistossWir" zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

Geradeaus

Beschreibung: siehe Handlung Geradeaus auf Seite 17

Funktionsweise: Hier wird dem Roboter die Handlung “Geradeaus” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

Klaeren

Beschreibung: siehe Handlung Klaeren auf Seite 17

Funktionsweise: Hier wird dem Roboter die Handlung “Klaeren” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

Mittelstuermer

Beschreibung: siehe Handlung Mittelstuermer auf Seite 17

Funktionsweise: Hier wird dem Roboter die Handlung “Mittelstuermer” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

NeutralePositionierung

Beschreibung: siehe Handlung Npos auf Seite 18

Funktionsweise: Hier wird dem Roboter die Handlung “Npos” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

PassiveAbwehr

Beschreibung: Diese Rolle ist zur Vergabe von Abwehrhandlungen gedacht, die sich gegen den passiven Angreifer richten.

Funktionsweise: Es wird geprüft ob ein passiver Angreifer zum abwehren da ist. Wenn sich mehr als ein Gegner in unserer Spielfeldhälfte befindet wird die Handlung “PassiverGegnerBlocken” zugewiesen, in allen anderen Fällen “TorschussBlocken”.

Bewertung: Bei der Abfrage der eigenen Hälfte waren die x und y Koordinaten vertauscht!!! Fehler behoben.

Rammbock

Beschreibung: Testrolle

TestRolle

Beschreibung: Testrolle

Vorstopper

Beschreibung: siehe Handlung Vorstopper auf Seite 21

Funktionsweise: Hier wird dem Roboter die Handlung “Vorstopper” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

Torwart

Beschreibung: siehe Handlung TorwartAbwehr auf Seite 20

Funktionsweise: Hier wird dem Roboter die Handlung “TorwartAbwehr” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

ZweiterTorwart

Beschreibung: siehe Handlung ZweiterTorwart auf Seite 21

Funktionsweise: Hier wird dem Roboter die Handlung “ZweiterTorwart” zugewiesen.

Bewertung: Handlungszuweisung funktioniert.

2.1.3.3 Spielzüge

Wie schon an Anfang erwähnt bilden die Spielzüge die höchste Entscheidungsebene. Von Ihnen hängt ab welche Rollen ausgewählt werden dadurch letztendlich welche Handlungen zum Zug kommen. Hier ist eine Feinabstimmung zuwischen den einzelnen Spielzügen spielentscheidend. Während der Tests zeigte sich jedoch ein etwas anderes Bild. Viele der Spielzüge werden nur selten gewählt und

erscheinen redundant andere trafen die Rollenauswahl fast willkürlich. Um eine bessere Übersicht zu bieten haben wir Diagramme erstellt die die Verknüpfungen zwischen den Spielzügen, den Rollen und den Handlungen verdeutlichen. Jeder Beschreibung folgt eine entsprechende Abbildung. In diesem Teil der Tests ist der Abschnitt "Funktionsweise" unterteilt worden. Zuerst wird die Bewertung erklärt, die der Spielzug anhand der aktuellen Spielsituation vornimmt, dann die Art und Weise auf die die Rollen verteilt werden.

Angriff

Beschreibung: Der Name dieses Spielzuges ist selbsterläuternd.

Funktionsweise: Siehe Abb. 2.1. **Bewertung:**

In die Bewertung fließen folgende Kriterien ein: Die Zeit in der unserer schnellster Roboter am Ball ist. Bei 2 sec wird der Bewertungsfaktor um 0.0 erhöht, bei 0 sec um 0.2. Die Zwischenwerte werden linear skaliert. Falls wir auch noch schneller als der Gegner am Ball sind, wird die Bewertung nochmals um 0.1 erhöht. Sollte unserer ballnächster Roboter sich hinter dem Ball befinden resultiert das in einer um 0.1 höheren Bewertung.Ist die Ballbewegung grob auf das gegnerische Tor gerichtet erhöht sich die Bewertung nochmals um 0.1. Anschließend wird noch die absolute Ballposition beachtet. Befindet der Ball sich links von dem "freekick"-Punkt in unserer Spielfeldhälfte, beträgt die Erhöhung des Bewertungsfaktor 0.0. Sollte sich der Ball rechts von dem entsprechenden Punkt in der Gegnerhälfte befinden, ist dieser Anteil 0.3. Zwischen diesen beiden Punkten wird der Wert linear skaliert.Die Maximale Bewertung, die dieser Spielzug erfahren kann ist 0.8.

Rollenverteilung: Mit Hilfe der "TorwartTest" Bibliothek wird der optimale Torwart bestimmt. Dieser Bekommt die Rolle "Torwart" zugewiesen. Die Rolle "Vorstopper" bekommt der Roboter der den berechneten optimalen Vorstopperposition am nächsten ist und nicht der Torwart ist. Von den übriggebliebenen Robotern wird die Rolle "AngriffmitBall" an den ballnächsten Roboter vergeben. Anschließend wird "Mittelstuermer" nach dem selben Kriterium vergeben. Der letzte Roboter führt "AngrffsPositioniereung" aus. Es existiert eine alternative defensive Rollenverteilung, die ist aber nicht für das Spiel 5 vs. 5 bestimmt.

Bewertung: Rollenzuweisung funktioniert.

Verbesserungsvorschläge: Keine.

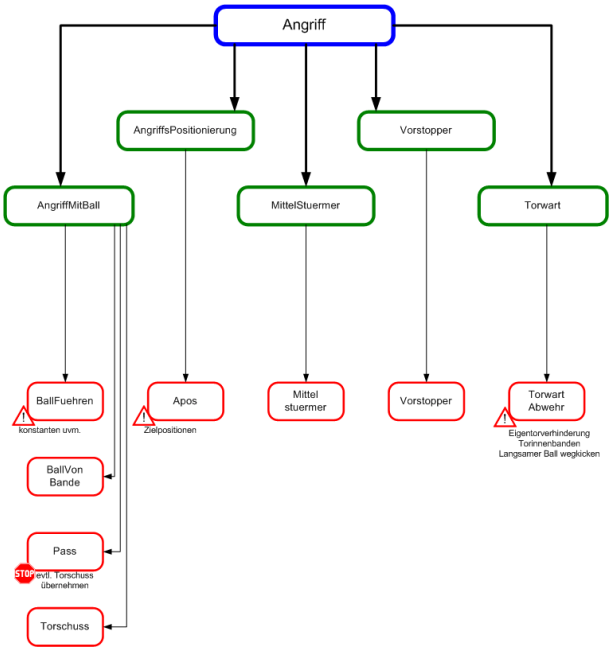


Abbildung 2.1: Spielzug Angriff

AnstossDie

Beschreibung: Der Spielzug “AnstossDie” ist für die Durchführung der Standardsituation “Anstoß für den Gegner” zuständig.

Funktionsweise: Siehe Abb. 2.2. **Bewertung:**

Es findet keine Bewertung statt (bzw. die Bewertung ist konstant 1.0), da der Spielzug manuell aktiviert wird.

Rollenverteilung: Der, mit Hilfe von “TorwartTest”, bestimmte optimale Torwart bekommt diese Rolle auch. Alle anderen Roboter bekommen “AufdenBall”. Der Spielzug läuft maximal 1 Sekunde lang oder bis zu dem Zeitpunkt in dem der Ball den Mittelkreis verlassen hat.

Bewertung: Die Rollenverteilung funktioniert. Ob diese sinnvoll ist, ist eine ganz andere Frage. Allgemein lässt sich sagen, dass es nicht empfehlenswert ist mehreren Robotern dieselbe Rolle, oder noch schlimmer dieselbe Handlung zuzuweisen. In solchen Fällen fahren alle Roboter dieselben Zielpunkte an, was in einem Crash endet. Hier fällt dies nicht auf, da dieser Spielzug nur 1 Sekunde aktiv bleibt.

Verbesserungsvorschläge: Rollenverteilung überdenken. Zeit nicht über die Strategieaufrufe ermitteln.

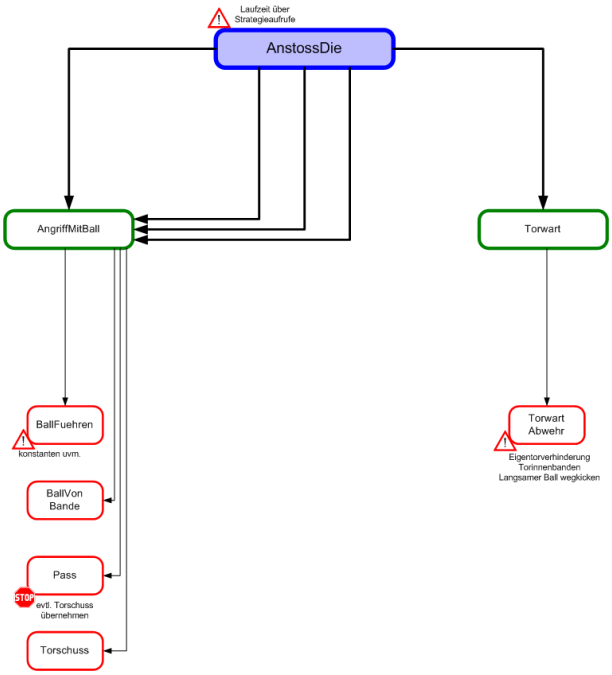


Abbildung 2.2: Spielzug AnstossDie

AnstossWir

Beschreibung Dieser Spielzug ist für die Standartsituation “unserer Anstoß” gedacht.

Funktionsweise: Siehe Abb. 2.3. **Bewertung:**

Wie bei allen anderen Sandartsituationen ist die Bewertung konstant bei 1.0.

Rollenverteilung: Der beste Torwart wird über den “TorwartTest” bestimmt. Danach werden die Rollen (“Geradeaus”) fast willkürlich vergeben. Eine Intention der Autoren ist hier nicht zu erkennen. Eindeutig ist, dass die ersten 0.8 Sekunden an einen Roboter die Rolle “AnstossTick” vergeben wird danach “Angriffspositionierung”. Den restlichen Robotern wird “AngriffmitBall” zugewiesen.

Bewertung: Da die Intention bei der Rollenvergabe nicht erkenntlich war, lässt sich hier nicht viel zur Bewertung sagen.

Verbesserungsvorschläge: Massiv überarbeiten! Es scheint, dass die Handlung mehrmals “auf die schnelle” überarbeitet worden ist, ohne Rücksicht auf Strukturierung zu nehmen oder auch erklärende Kommentare zu erstellen. Eine Erweiterung auf 5 vs. 5 scheint auch ohne viel Vorüberlegung geschrieben worden zu sein.

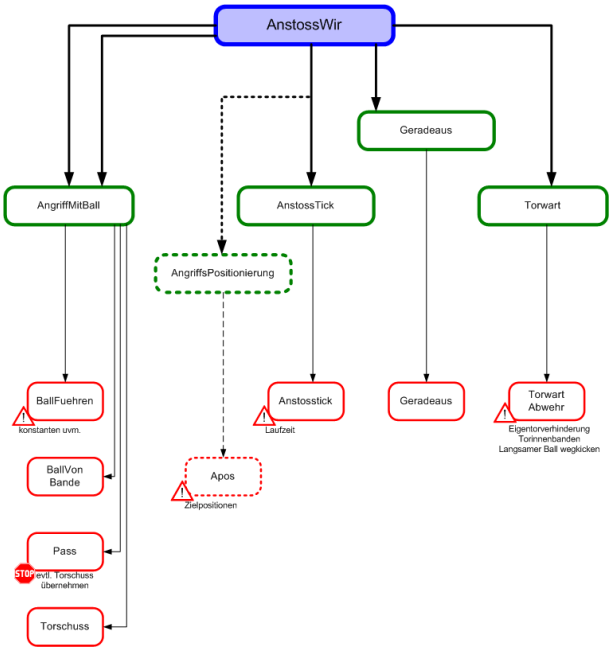


Abbildung 2.3: Spielzug AnstossWir

Freiball

Beschreibung: Hier wird die Standardsituation “Freiball” durchgeführt.

Funktionsweise: Siehe Abb. 2.4. **Bewertung:**

Wie bei allen Standardsituationen ist die Bewertung konstant bei 1.0, da der Spielzug auf Knopfdruck aktiviert wird.

Rollenverteilung: Der dafür geeigneteste Roboter wird zum “Torwart”. Der Roboter der in einem bestimmten Toleranzradius (Konstanten.cpp, FreeBallDistance) um einen der Freiballpunkte steht wird angewiesen die Rolle “geradeaus” auszuführen. Alle anderen Roboter bekommen die Rolle “AngriffmitBall”. Dieser Spielzug hat die Laufzeit von einer Sekunde.

Bewertung: Rollenzuweisung funktioniert. Es werden hier nur drei verschiedene Rollen verteilt. Das Zuweisen von “AngriffmitBall” hat bei einer Laufzeit von einer Sekunde keinen Sinn.

Verbesserungsvorschläge: Rollenverteilung überdenken. Statt irgendwelche Rollen zu vergeben, wäre vielleicht ratsam eine neue Rolle “MacheNichts” entwerfen, die die Roboter einfach nicht bewegt. Laufzeitermittlung nicht über die Strategieaufrufe realisieren.

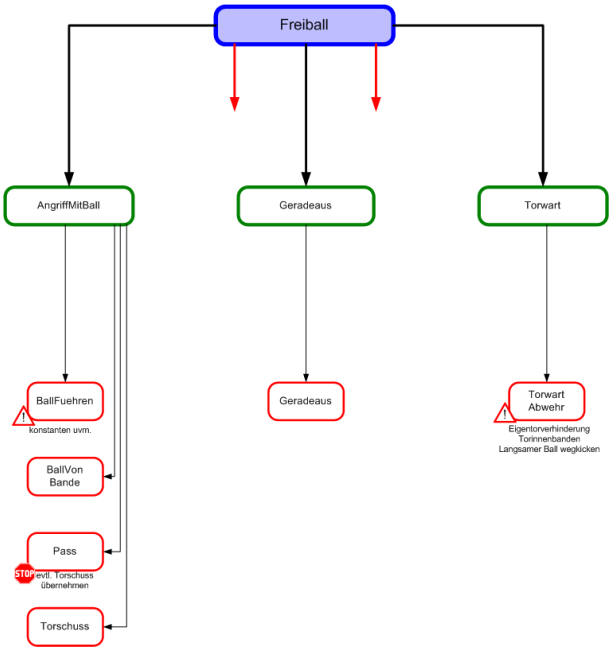


Abbildung 2.4: Spielzug Freiball

FreistossDie

Beschreibung: Spielzug für die Standardsituation: Freistoß für den Gegner.

Funktionsweise: Siehe Abb. 2.5. **Bewertung:**

Die Bewertung ist 1.0 da es sich um eine Standardsituation handelt.

Rollenverteilung: Der optimale Torwart ("TorwartTest") bekommt die Rolle "FreistossTorwart". Nach der Torwartbestimmung wird dem letzten Roboter der sich außerhalb des Strafraumes (y<450 oder y>850) die Rolle "AktiveAbwehr" zugewiesen. Alle anderen Roboter führen "AngriffmitBall" aus. Abbruchbedingung für diesen Spielzug ist das überschreiten der maximalen Laufzeit von einer Sekunde oder eine zu große Annäherung des Balles an den Roboter der die "AktiveAbwehr"-Rolle ausführt.

Bewertung: Rollenzuweisung funktioniert.

Verbesserungsvorschläge: Es werden lokale Konstanten benutzt. Dadurch waren die Strafraumgrenzen fehlerhaft (noch an das kleine Spielfeld angepasst). Konstanten.cpp benutzten. Die Abbruchbedingung ist unsinnig. Es wird drei mal dieselbe Rolle vergeben. Rollenverteilung überdenken.

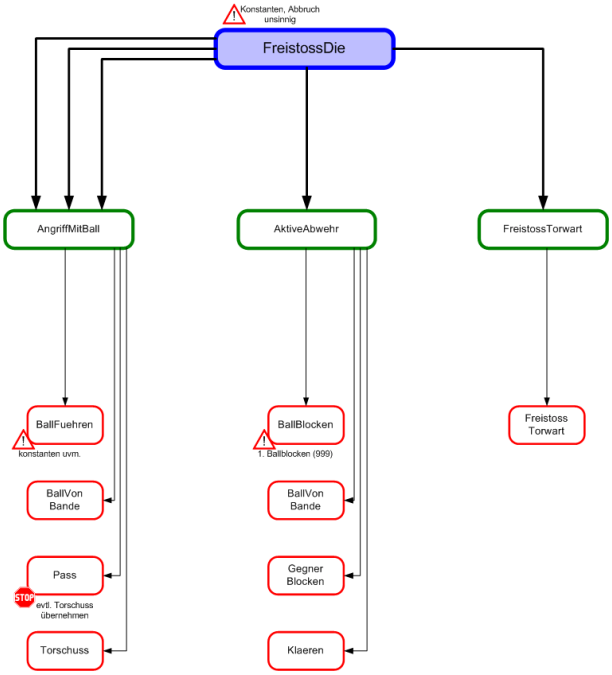


Abbildung 2.5: Spielzug FreistossDie

FreistossWir

Beschreibung: Spielzug für unseren Freistoß (Standardsituation).

Funktionsweise: Siehe Abb. 2.6. **Bewertung:**

Die Bewertung ist 1.0 da es sich um eine Standardsituation handelt.

Rollenverteilung: Nach der Torwartbestimmung ("TorwartTest") wird dem letzten Roboter der über der Spielfeldmitte steht die Rolle "FreistossWir" zugewiesen. Die restlichen Roboter bekommen die Rolle "AngriffmitBall". Die Spielzugdauer ist der Konstanten.cpp festgelegt (Freistossdauer).

Bewertung: Spielzug funktioniert nur selten. Im Quellcode ist die x mit der y-Koordinate vertauscht! Der Spielzug wurde fast identisch aus "FreistossDie" übernommen.

Verbesserungsvorschläge: Eine Rolle wird drei mal vergeben. Rollenverteilung überdenken. Den Spielzug überarbeiten!

NeutralEigeneHaelfte

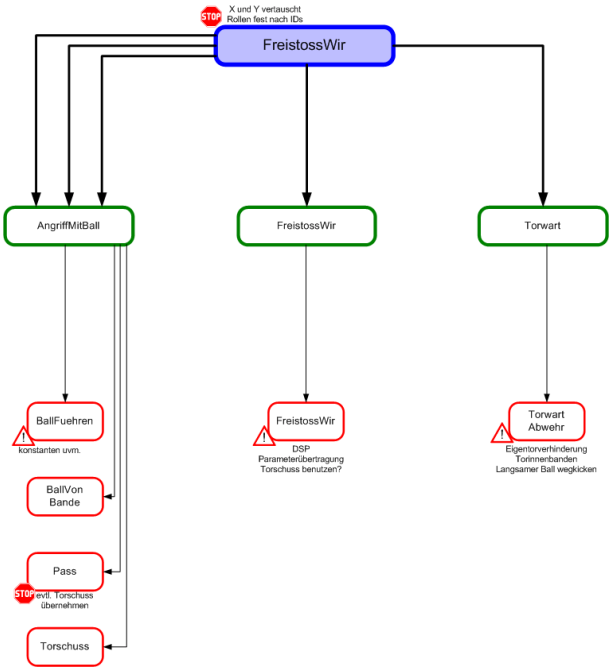


Abbildung 2.6: Spielzug FreistossWir

Beschreibung: Der Spielzug ist für eine Spielsituation gedacht nicht eindueutig dem Angriff oder der Verteidigung zugeordnet werden kann. Der Ball befindet sich in unserer Hälfte.

Funktionsweise: Siehe Abb. 2.7 **Bewertung:**

Wie schon beim Spielzug “Angriff” ist die absolute Ballposition ein Faktor für die Bewertung. Befindet sich der Ball links von dem “freekick” Punkt in unserer Spielfeldhälfte, beträgt der Bewertungsfaktor 0.01. Sollte sich der Ball rechts von dem entsprechenden Punkt in der Gegnerhälfte befinden, ist dieser Anteil 0.0. Zwischen diesen beiden Punkten wird der Wert linear skaliert. Außerdem fließt die Entfernung aller Spieler zum Ball in die Bewertung ein. Je weiter alle vom Ball entfernt sind (Reihenfolge) umso höher die Bewertung. Hier ist bei einer Entfernung die größer ist als eine Sekunde ein Zuwachs des Bewertungsfaktors um 0.05 möglich. Wieder folgt eine lineare Skalierung die gegen Null geht, wenn der eigene Roboter direkt am Ball ist. Zusätzlich wird die Bewertung um 0.05 erhöht wenn einer unserer Roboter schneller am Ball ist als die Gegner. Die Maximalbewertung die dieser Spielzug erfahren kann liegt bei 0.11. Die Gesamtbewertung des Spielzuges ist insgesamt sehr niedrig gehalten, dadurch wird dieser praktisch nie ausgewählt. Die Intention der Autoren ein solches Verhalten zu hervorzurufen lässt sich nur vermuten. Wahrscheinlich brachte der Spielzug die Spielbalance aus dem Gleichgewicht und aus diesem Grund wurde die Bewertung abgestuft.

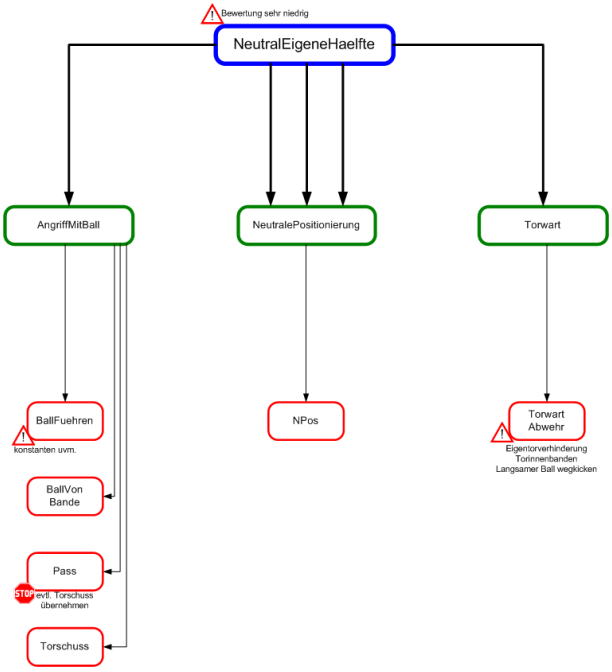


Abbildung 2.7: Spielzug NeutralEigeneHaelfte

Rollenverteilung: Der ballnächste Roboter bekommt die Rolle “AngriffmitBall”. Danach wird der optimale Torwart mit Hilfe vom “TorwartTest” bestimmt. Sollte der beste Torwart der ballnächste Roboter sein wird dem zweitbesten Torwart die “Torwart”-Rolle zugewiesen. Die übriggebliebenen Roboter führen “NeutralePositionierung” aus.

Bewertung: Rollenverteilung funktioniert. Diese Handlung ist eindeutig für ein Spiel 3 vs. 3 geschrieben worden. Die dreifache Vergabe der Rolle “NeutralPositionierung” ist umso kritischer, da hier nur eine Handlung (“Npos”) vergeben wird. Dadurch fahren alle Roboter die selben Zielkoordinaten an.

Verbesserungsvorschläge: Gründe für die niedrige Bewertung erfahren. Spielzug entweder ganz deaktivieren oder massiv umschreiben.

NeutralGegnerHaelfte

Beschreibung: Der Spielzug ist für eine Spielsituation gedacht nicht eindueutig dem Angriff oder der Verteidigung zugeordnet werden kann. Der Ball befindet sich in der gegnerischen Hälfte.

Funktionsweise: Siehe Abb. 2.8 **Bewertung:**

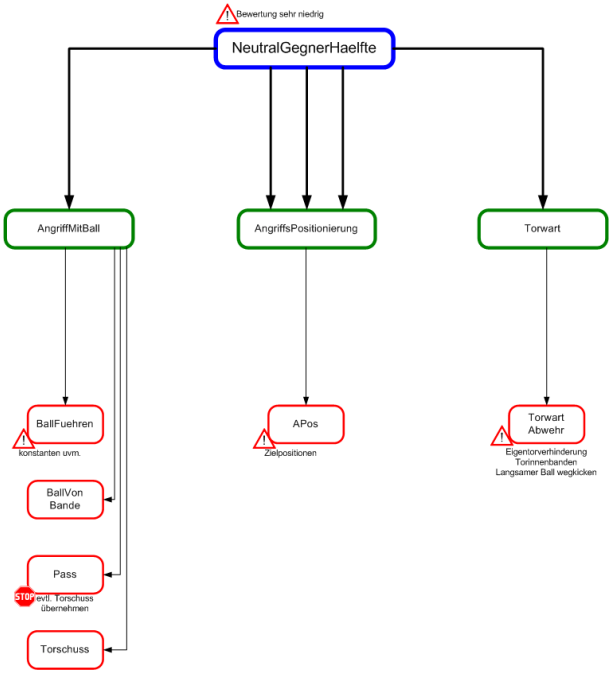


Abbildung 2.8: Spielzug NeutralGegnerHaelfte

Die Bewertung läuft analog zu “NeutralEigeneHaelfte”. Es wurde nur die Richtung des Faktors “absolute Balllage” umgekehrt. Sonst ist die Bewertung identisch und liefert auch dieselben niedrigen Werte.

Rollenverteilung: Hier sieht die Situation genauso aus. Rollenverteilung ist identisch mit “NeutralEingeneHaelfte” bis auf die Tatsache, dass statt “NeutralePositionierung” “AngriffsPositionierung” vergeben wird. Bewertung, Verbesserungsvorschläge siehe “NeutralEigeneHaelfte” auf Seite 32.

Verhakt

Beschreibung: Dieser Spielzug, genauso wie das gesamte “Verhakt”-Verhalten wird von einem HiWi bearbeitet und befindet sich noch in der Entwicklung. Aus diesem Grund haben wir von einem Test abgesehen.

Verteidigung

Beschreibung: Dieser Spielzug koordiniert die eigene Verteidigung.

Funktionsweise: Siehe Abb. 2.9. **Bewertung:**

Die Bewertung für diesen Spielzug unterscheidet zwischen zwei verschiedenen Fällen. Befindet sich der Ball in unserem Strafraum wird die Bewertung sehr hoch gesetzt (0.9) um sicherzustellen, dass eine Verteidigung auf jeden Fall stattfindet. Für den anderen Fall sind drei verschiedene Kriterien ausschlaggebend. Zuerst wäre da die absolute Ballposition. Abhängig hiervon wird die Bewertung um 0.2 für die Balllage in der eigenen Hälfte und um 0.08 für die Lage in bis zu $\frac{3}{4}$ der Spielfeldlänge heraufgesetzt. Das zweite Kriterium ist der Ballbesitz. Ist der Gegner zuerst am Ball erhöht sich die Bewertung um 0.2, im anderen Fall um 0.08. Der letzte Faktor der die Bewertung beeinflusst ist die festvorgegebener Wert wird zu dem Bewertungsfaktor 0.08 addiert. Zusätzlich spielt der Winkel der Bewegung eine Rolle. Bewegt sich der Ball in einem Winkel von 5° auf unserer Tor zu, wird die Bewertung um 0.32 aufgewertet. Bei einem Winkel bis zu 45° sind es 0.24, bis zu 90° 0.12 und über 90° immer noch 0.08. Die maximale Bewertung des Spielzugs in dieser Situation liegt bei 0.8.

Rollenverteilung: Der optimale Torwart wird über den "TorwartTest" bestimmt. Die "Vorstopper" Rolle bekommt der Roboter der sich einer günstigen Vorstopperposition am nächsten befindet. "AktiveAbwehr" wird abhängig vom Ballbesitz vergeben. Sollte sich der Gegner im Ballbesitz befinden, bekommt die Rolle der Roboter der den ballführenden Gegner erreichen kann und am weitesten vom Ball entfernt ist. Ansonsten wird die Rolle einfach an den am weitesten vom Ball entfernten Roboter vergeben. Derjenige von unseren Spielern der sich am weitesten in der gegnerischen Hälfte befindet, also die maximal x-Position hat, bekommt die Rolle "AngriffsPositionierung". Der letzte Spieler führt "PassiveAbwehr" aus.

Bewertung: Rollenverteilung funktioniert bedingt. Es kann passieren, dass die Rolle des "Vorstoppers" durch "AktiveAbwehr" überschrieben wird.

Verbesserungsvorschläge: Die Rollenverteilung überdenken. Durch Änderungen kommt die Rolle "ZweiterTorwart" gar nicht zum Zuge. Das "Vorstopper"-"AktiveAbwehr" Problem lösen.

2.1.4 Korrekturen und Erweiterungen

Im Laufe der Strategietests wie auch bereits zuvor wurden einige Korrekturen und Erweiterungen der strategischen Softwarekomponente vorgenommen. Die getätigten Änderungen sollen in diesem Abschnitt Erwähnung finden.

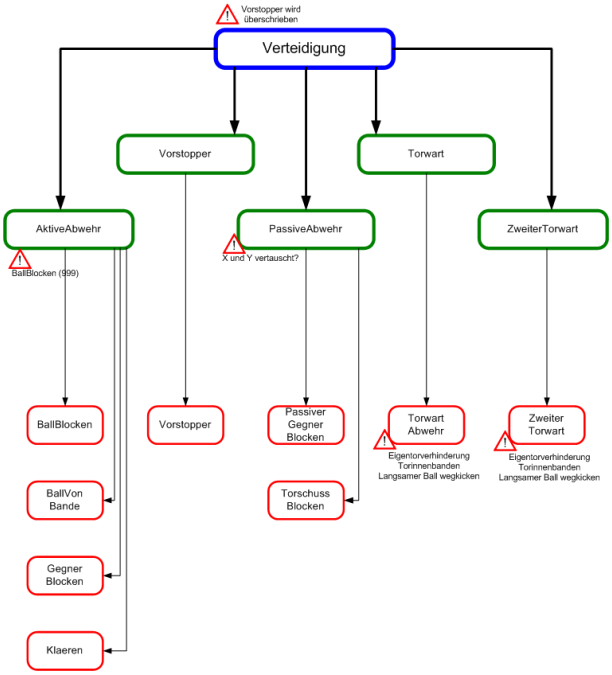


Abbildung 2.9: Spielzug Verteidigung

Schon vor den Softwaretests war in Spieltests eindeutig zu erkennen, dass die Handlung “Torschuss” unbedingt verbessert werden musste. Vorallem die fehlende Betrachtung der Torwartposition und die unnötig komplizierte Berechnung des Schussweges machten dies nötig. Die aktuelle Handlung ermittelt nun zunächst die Position des gegnerischen Torwartes und setzt schließlich die Schusszielposition entsprechend der eigenen Position und der Ermittlung des Winkels zum Tor. Hierdurch werden auch Sonderfälle abgefangen, wie z.B. Schuss auf die kurze Ecke des Tores bei einem zu “spitzem” Winkel. Neben den genannten Verbesserungen sollte noch zusätzlich eine Erweiterung folgen, die sich der Regel “Torwart charging” widmen sollte. Hierbei wird zugunsten der verteidigenden Mannschaft auf einen Abstoss entschieden, weil ein angreifender Spieler den Torwart mit Ball ins Tor geschoben, gesperrt, manchmal auch nur berührt hat. Vereinzelt wurde hierbei ein korrekt erzieltes Tor durch ein anschliessendes Berühren des Torwartes annuliert. In der Torschuss-Handlung sollte dies durch ein Abbremsen des Stürmers im Strafraum und nach erfolgtem Torschuss durch ein anschließendes Abdrehen vermieden werden. Praktische Tests ergaben aber, dass sich diese Erweiterung aufgrund der aktuellen Robotersteuerung für einen Torschuss als eher hinderlich erweist. Somit wurden die entsprechenden Bereiche im Quellcode vorerst auskommentiert.

Auch bereits während der Durchführung der Strategie-Tests wurden Änderungen vorgenommen. Dinge, die leicht zu ändern waren, manchmal auch noch offensichtlich falsch waren und somit beispielsweise

eine Handlung unbrauchbar werden ließen, fallen in diesen Bereich, wie z.B. die Vertauschung von x- und y-Koordinate. Während der Durchsicht der Quellcodes wurde u.a. darauf geachtet, dass sich keine relevanten Konstanten mehr in den einzelnen Dateien befinden sollten; diese wurden in die dafür vorgesehene Konstanten-Datei übertragen. Diese Datei selbst wurde auch einer Überprüfung unterzogen, in der sich zeigte, dass nicht alle Spielfeld-Konstanten korrekt waren. So bezogen sich die Werte des Strafraumes für das größere Spielfeld, auf dem mit 5 gegen 5 Robotern gespielt wird, lediglich auf den Torraum, und beeinflusste somit fast alle Handlungen. Dieser Fehler scheint darauf zurückzuführen zu sein, dass die Strategie ursprünglich für die Spielversion 3-gegen-3 geschrieben und anschliessend auf 5-gegen-5 erweitert wurde. Hierbei sind offensichtlich einige Dinge vergessen worden. In diesen Bereich gehören auch die folgenden beiden Korrekturen, die sich auf komplette Handlungen konzentrieren, die noch in der Version 3-gegen-3 verblieben sind. Die Handlung "NPos", die einen Roboter auf eine Position auf der Mittellinie, die allen Gegner-Robotern am entferntesten ist, "neutral positioniert", berücksichtigte somit nur 3 Gegner und konnte somit schlimmstenfalls eine Position ergeben, auf der sich bereits ein gegnerischer Roboter mit der ID 4 oder 5 befand. Somit konnte sich aus "neutral positionieren" sogar eine offensichtlich gegenteilige Handlung ergeben, die den Roboter gegen einen gegnerischen fahren ließ. Diese Handlung wurde dahingehend korrigiert, dass sie nun alle gegnerischen Spieler berücksichtigt. Außerdem werden die ursprünglich festen Positionen auf der Mittellinie nun abhängig von der Spielfeldgröße errechnet.

Dasselbe Problem ergab sich bei der Handlung "PassiverGegnerBehindern". Hierbei werden zunächst der Gegner, der seinem eigenen Tor und der, der dem Ball am nahsten ist ausgeschlossen. Bei der Version 3-gegen-3 ist der dritte Spieler nun der zweite Angreifer. Dieser soll in seinen Handlungen behindert werden. Wie schon zuvor können jetzt aber auch die Roboter mit den IDs 4 oder 5 zweiter Angreifer sein. Die berichtigte Version ergibt nun eine Behinderung des zweit-ballnahsten Gegnerroboters; außerdem wurden hier noch fehlerhafte Vektorberechnungen korrigiert.

Während der Ausarbeitung dieses Endberichtes wurden noch zusätzliche drei Verbesserungen bzw. Erweiterungen in Angriff genommen, die jetzt noch nicht als vollendet und getestet beschrieben, über deren grundlegende Idee und zukünftige Funktionsweise hier jedoch schon Aussagen getätigt werden können.

Wie schon im vorangegangenen Abschnitt über die Strategietests erwähnt, erschien eine der wohl wichtigsten Handlungen, die Torwartabwehr, als noch nicht ausgereift und fehlerhaft. Ein Problem, dass sich in vielen Testspielen zeigte war die Tatsache, dass der Torwart sich des öfteren im eigenen Tor am "Torpfosten" verhakte und somit als Torwart nicht mehr einsatzbereit war, hieraus ergaben

sich vielfach Torchancen für die gegnerische Mannschaft, weiterhin schien auch die Vermeidung von Eigentoren, wenn sich der Ball zwischen Torwart und eigenem Tor befand, nicht richtig zu funktionieren. Aufgrund der Wichtigkeit des Torwartes erfährt diese Handlung eine komplette Überarbeitung. Somit wird sich der Torwart in Zukunft folgendermaßen verhalten: Die Grundhaltung des Torwartes wird nach wie vor darin bestehen auf einer Linie vor dem eigenen Tor auf und ab zu fahren. Neu hierbei wird sein, das als Konstante nicht die feste Torwartlinie geladen wird, sondern in Zukunft wird der Abstand dieser Linie zur Bande als Grundeinstellung zur Verfügung stehen. Diese Änderung wird vorgenommen, da sich der Torwart vielfach schon mit der Bande verhakte; Tests werden einen geeigneten Abstand ergeben, alle weiteren Dateien, die die Konstante "Torwartlinie" benutzen werden sich diese Linie lokal aus der Addition des Torwartabstandes zur linken Torlinie errechnen. Zusätzlich wird der Torwart etwas weiter zur Seite fahren als nur bis zum Torpfosten. Die Position, die der Torwart auf der Linie einnimmt ist abhängig von der Ballposition. Die Berechnung hierzu wird dementsprechend geändert, daß sich der Torwart immer mittig zwischen den Linien "Ball - oberer Pfosten" und "Ball - unterer Pfosten" befindet. Tests werden zeigen, ob sich eine direkte Y-Anpassung der Torwart- an die Ballposition, befindet sich dieser sehr nahe dem eigenen Tor, als sinnvoll erweist. Weiterhin wird der Torwart, wie gehabt, einen langsamen Ball im Strafraum in einem 45 Grad - Winkel nach oben bzw. unten aus dem Strafraum herausspielen. Hierbei soll aber entgegen der Ursprungsversion zusätzlich berücksichtigt werden, dass der Torwart auch derjenige Spieler ist, der den Ball als erster erreichen kann. Ist zuerst ein eigener Abwehrspieler am Ball, so treten in Zukunft keine Behinderungen mehr auf, ist ein Gegner eher am Ball, so kann die Möglchkeit, dass ein gegnerischer Angreifer einem leeren Tor gegenübersteht ausgeschlossen werden. Als zusätzliche Neuerungen soll verhindert werden, dass sich ein Torwart im Tor am Pfosten verhakt. Hierzu soll in der Handlung selbst eine eigene "Torwart-verhakt" - Erkennung implementiert werden, die den Torwart beim Eintreten dieses Falles durch eine Rotation um das spielfeldinnere Rad zurück auf die Torwartlinie schwenken lassen soll. Weiterhin soll durch eine ähnliche Aktion eine Eigentor-Verhinderung realisiert werden. Eine letzte optionale Erweiterung soll sich der Tatsache widmen, dass ein vielfach auftretender Torerfolg durch eine Art Alleingang eines einzelnen Angreifers erzielt wird, der kurz vor dem Tor einen kurzen Schlenker in eine Ecke macht. Der Torwart bleibt auf seiner Linie und ist dadurch meistens unterlegen. Mit einem Blick auf den realen Fußball entstand die Idee, sich dem anstürmenden Gegner entgegen zu bewegen und zwar nach dem Prinzip, das auch für die Bewegung auf der Linie eingesetzt werden soll. Der Torwart befindet sich bezüglich des Balls immer optimal zu seinem Tor und verkürzt damit zusätzlich den Winkel. Gleichzeitig würde ein herannahender Abwehrspieler automatisch die Rolle des Torwartes übernehmen. Probleme hierbei ergeben sich vorallem in der Erkennung eines solchen Alleinganges, die

verhindern muss, dass sich der Torwart während einer normalen Spielsituation einem gegnerischen Spieler entgegenwirft.

Desweiteren befindet sich derzeit die sogenannte Strafraum-Bibliothek in der Ausarbeitung. Die Spielregeln sehen vor, dass sich im eigenen Torraum lediglich der verteidigende Torwart und im Satrfraum zusätzlich nur ein einzelner weiterer Spieler befinden darf; Regelverstöße hiergegen werden mit Strafstößen geahndet. Fast jede Handlung besitzt demzufolge eine Routine, die dafür sorgen soll, nicht in den eigenen Strafraum einzufahren. Die praktischen Tests zeigten, dass die Roboter aufgrund ihrer Trägheit meist doch erst eine gewisse Strecke in den Strafraum und dann wieder herausfahren. Die aktuell eingesetzten Routinen innerhalb der Handlungen erzeugen somit ein Hin- und Herfahren der verteidigenden Spieler an der Strafraumgrenze. Weiterhin unterscheidet sich die Zielposition, die ausserhalb des Strafraumes gesetzt wird, von der eigentlichen im Strafraum nur in der x-Koordinate, die Roboter werden also entsprechend ihrer y-Koordinate parrallel zu den Seitenbanden vor den Strafraum geschickt. Der Nachteil hieraus ergibt sich bei Betrachtung des Zusammenhanges zwischen Zielposition und dafür verantwortlicher Referenz-Position dieser Handlung. Beispielsweise wird in der Handlung "BallBlocken", die Position innerhalb des Strafraumes aus dem Verbindungsvektor zwischen Ballposition und Tormitte ermittelt. Eine Verschiebug der x-Koordinate dieser Position beeinflusst den Roboter in seiner eigentlichen Handlung, da er sich nun nicht mehr auf dem Verbindgsvektor befindet; ein Torschuss kann somit ungehindert an dem betreffende Roboter vorbeigehen statt an ihm abzuprallen. Zusätzlich sorgte noch die Tatsache für Verwirrung, dass sich die Routinen innerhalb der Handlungen aufgrund verschiedener Autoren in ihrer Implementierungsart und auch Wirkungsweise unterscheiden. Somit wurde entschieden, die Strafraum-Routinen aus den Handlungen in eine zentrale separate Bibliothek auszulagern. Als Parameter für den Aufruf werden die Zielposition des Roboters sowie die für die Handlung relevante Referenzposition angegeben. Die Bibliothek berechnet hieraus eine äquivalente Position ausserhalb des Strafraumes über einfache Strahlensätze durch Anpassung der x- und anschließend entsprechender y-Koordinate. Sonderfälle wie beispielsweise die Tatsache, dass sich die Referenz-Position zwischen Strafraum und Seitenbande befindet und somit zuerst eine Änderung der y-Koordinate erfolgen muss, werden hierbei berücksichtigt. Weiterhin soll die Bibliothek es sinnvoll einem einizigen Roboter erlauben, den Strafraum, aber nicht den Torraum zu betreten, um den Torwart bei der Verteidigung zu unterstützen.

Die dritte Erweiterung, die sich derzeit in Entwicklung befindet, widmet sich der automatischen Anfahrt der Positionen für Standardsituationen. Bisher wurden diese Positionen fest nach Roboter-IDs verteilt. Der Nachteil hierbei ergab sich wenn die Roboter beispielsweise von Hand aufgestellt wurden,

denn auch die Verteilung der Handlungen für Standardsituationen erwartet diese feste Verteilung; ein hiervon verschiedene Aufstellung der Roboter kann somit beispielsweise dazu führen, dass der Torwart im Tor den Anstoss ausführen will und sich der Roboter am Mittelpunkt ohne den Ball zu spielen auf eine defensive Position zurückzieht. Eine dynamische Anfahrt der Zielpositionen abhängig von den Entfernungen der Roboter zu den einzelnen Positionen sowie eine Anpassung der Handlungsverteilungen für Standardsituationen soll diese Probleme beheben. Ein nützlicher Nebeneffekt hierbei ist, dass sich die Roboter bei der Anfahrt ihrer Standard-Positionen in Zukunft nicht mehr gegenseitig behindern.

Alle genannten Erweiterungen, die sich derzeit noch in der Entwicklung befinden werden bis Ende der Laufzeit dieser Projektgruppe fertiggestellt sein, andernfalls wird diese Arbeit von den übernommenen HiWis beendet werden.

2.2 Die Bildverarbeitung

2.2.1 Verbesserung des Trackings

2.2.1.1 Was ist Tracking

Im Robotsoccersystem hat die Bildverarbeitung eingehende Daten der Kamera derart zu verarbeiten, dass z.B. die Positionen der Roboter bestimmt werden. Dies ist Vorraussetzung für die Arbeit übergeordneter Systemteile, wie der Strategie. Die Strategie kann dann in Kenntnis der Roboterpositionen, -ausrichtungen und -geschwindigkeiten eine sinnvolle Fortsetzung des Geschehens ermitteln. Ohne genügend präzise Daten der Bildverarbeitung kann dies schwerlich geschehen, die Strategie würde dann quasi blind und mit völlig unrealistischen Werten arbeiten.

Das Tracking ist nun ein Teil des Gesamtverfahrens zur Ermittlung der o.g. 3 entscheidenden Roboterdaten.

Man habe nun eine Menge von Robotern R. Man numeriere nun die Elemente dieser Menge durch, also bei n Robotern z.B. $0, ..., n-1$. Jedes Element dieser Menge habe die 3 Attribute Position, Ausrichtung und Geschwindigkeit. Man bestimme nun initial für alle Roboter die Belegung dieser Variablen. Praktisch kann dies z.B. so geschehen, dass man die Roboter auf das Spielfeld setzt und das System automatisch eine Durchnumerierung vornimmt, d.h. jedem real auf dem Spielfeld stehenden Roboter wird intern, d.h. im Programm auf dem Hostrechner, eine Zahl aus $[0, n-1]$ zugeordnet. Nach dieser Initialisierung beginnt das eigentliche Tracking. Im folgenden wird das Trackingverfahren als Eingabe einerseits die Menge der alten Roboter R und andererseits die Menge R' der neuen Roboter bekommen. Die Aufgabe des Verfahrens besteht nun möglichst fehlerfrei zuzuordnen, d.h. der Roboter R_0 sollte dem Roboter aus R' zugeordnet werden, der in der Realität demselben Roboter entspricht. Wenn es also darum gehen soll die alte Menge R mit Hilfe der neuen Menge R' zu aktualisieren, dann geht es letztlich darum die Elemente aus R' auf die richtigen Elemente aus R zu kopieren. Kopiert man falsch, kommt es zum Problem der Vertauschung, d.h. Roboter 0 wird beispielsweise die Position von Roboter 2 zugeordnet. Das hat fatale Folgen für strategische Entscheidungen: Roboter 0 kann dann z.B. in Ballnähe gewähnt werden, steht aber in der Realität vielleicht am anderen Ende des Spielfeldes in weiter Ferne des Balles, wodurch ein Schussversuch zwar unterhaltsam ist, aber wohl wenig dienlich im Sinne des Erfolges. Ein Beispiel mag letzte Fragen zerstreuen: man habe zwei Roboter A und B, die als Attribut nur ihre Position mitbringen. Die alten Positionen seien A = 0,0 und B = 2,2. Neue Positionen seien nun Pos1 = 0,1 und Pos2 = 2,3. Die sich stellende Frage ist dann: steht A nun bei Pos1 oder bei Pos2? Daraus ergibt sich hier natürlich die Position von B. Ohne weitere Information ist die

Frage hier schwerlich mit absoluter Sicherheit zu beantworten, es wird sich aber zeigen, dass diese Frage grundsätzlich nur mit einer gewissen nicht 100%-igen Genauigkeit beantwortet werden kann. In dieser Lage relativer Ratlosigkeit, würde man wohl mit Nähe argumentieren und meinen, dass Pos1 eher A entspricht. Es zeigt sich schon hier die Implikation einer Heuristik. Eine Verbesserung springt natürlich ins Auge: wenn man fortlaufend neue Positionen bekommt und misst wieviel Zeit vergangen ist, kann man daraus die Geschwindigkeiten und Bewegungsrichtungen der Roboter ableiten und damit die Qualität der Zuordnungen stark verbessern. Derartige Ideen werden später genauer diskutiert werden.

2.2.1.2 Grundlegende Probleme

Beim Tracken haben wir alte Werte für Objekte und neue, bei denen man sich fragt, welchen Objekten sie zuzuordnen sind. Man hat hier konkret Momentaufnahmen von beweglichen Objekten. Die Hardware ist in der Lage mit einem bestimmten Zeittakt diese Bilder zu liefern. In dieser Diskretisierung, dem Takt, liegt der wesentliche Teil des Problems. Ist der Zeittakt sehr klein und die maximale Geschwindigkeit der Roboter ebenfalls, dann genügt ein simples Verfahren zur Realisierung des Trackings, denn dann sind Zuordnungen sehr leicht, da sich die Werte nur minimal ändern werden und so eine Abstandsmessung schon genügt. Bewegen sich die Roboter dagegen sehr schnell oder ist der Zeittakt sehr gross, kann irgendwann ein beliebig kluges Verfahren keine korrekte Zuordnung mehr sicherstellen. Man kann von der aktuellen Lage ausgehend berechnen, sofern man die Grenzen der Roboterbeweglichkeit kennt, welche Positionen nach dem Verstreichen einer bestimmten Zeit einnehmbar sind. Es gibt hier eine Grenze der Erreichbarkeit, einen gewissen Radius der Möglichkeiten, was natürlich auch für die weiteren Parameter, wie Geschwindigkeit und Ausrichtung gilt. Vergeht nur genügend Zeit, können sich die Ereignishorizonte zweier Roboter derart überlagern, dass beide sich danach exakt im selben Zustand (hinsichtlich Position,...) befinden können. Ist das der Fall, ist die Frage nach der exakt korrekten Zuordnung unentscheidbar. Das aktuelle Tracking bekommt Momentaufnahmen im Takt von 1/30 Sek. Diese Auflösung sollte genau genug sein.

2.2.1.3 Ausgangslage PG 416

Hier wird der Stand des Trackings beschrieben, den die PG 416 dann verbesserte.

Es geht hier um die Bildverarbeitung 2, die sich im Robotsoccer- Projekt unter `robotsoccer/BV/BV2` befindet. Das hier zu beschreibende Verfahren fand sich in Tracker.h und Tracker.cpp, genauer in der Methode doTrackRobs(). Es bekommt als Eingabe Rob robs, das sind die alten nun zu aktualisierenden Roboter und PatchList inputList, was die neuen Werte beinhaltet, die es zuzuordnen gilt.

Nun folgen 2 Schleifen. In der ersten wird über alle alten Roboter iteriert, in der zweiten über alle verbliebenen neuen Werte. Jedem Roboter i wird nun derjenige neue Wert zugeordnet, bei dem der Abstand nach Positionswerten am kleinsten ist (kleinster euklidischer Abstand). Die Positionswerte sind dabei zweidimensional und beziehen sich auf den Mittelpunkt eines Roboters. Ist der neue Wert gewählt, wird er aus der Liste aller neuen Werte entfernt. Es kann vorkommen, dass zuwenig neue Werte vorliegen.

Konsequenzen des Verfahrens: Da nur Positionen verglichen werden, ist die Genauigkeit der Zuordnung prinzipiell (unnötig) beschränkt. Ist die Geschwindigkeit der Roboter hoch sind Vertauschungen fast garantiert. Speziell problematisch ist es, wenn ein Roboter einem anderen schnell hinterher fährt. In diesem Fall erscheint es sehr wahrscheinlich, dass die Positionen genau umgekehrt zugeordnet werden. Da die neuen Werte in keiner Weise sortiert werden und zudem Werte fehlen können, ist durch den Algorithmus (fehlende Werte vorausgesetzt) in bestimmten Situationen eine Vertauschung auch bei beliebig kleinem Takt sichergestellt. Wird derjenige Roboter dem mangels Wissen seine alte Position zugeordnet werden müsste, nicht zuletzt aktualisiert, wird er zwangsläufig einer neuen Position zugeordnet werden und damit einem anderen Roboter dessen neue Position nehmen. Dieser kann dann ebenfalls nur falsch zugeordnet werden.

In der Praxis zeigte sich, dass das letzte Problem besonders relevant war. D.h. wurden nicht alle neuen Positionen erkannt, kam es oft zu Vertauschungen. Dagegen schien das erstgenannte Problem von geringer Bedeutung zu sein, was man der recht hohen zeitlichen Auflösung zuschreiben kann.

Zur praktischen Seite weiterhin darf man nicht vergessen, dass ein bedeutsames Turnier (im Rahmen der GI-Tagung) bevorstand. Die allgemeine Einschätzung war, dass mit dem hier beschriebenen Stand des Trackings eine Wettbewerbsfähigkeit kaum gegeben sein dürfte.

2.2.1.4 Wichtige Situationen

Hier sollen die wichtigsten Situationen vorgestellt werden, die die wesentlichen Probleme des Trackings beinhalten. Allgemein kann man sagen, dass diese Probleme etwas damit zu tun haben, dass zwei Roboter besonders viele Gemeinsamkeiten durch die spezielle Situation erhalten (z.B. gleiche Bewegungsrichtung). Besonders hohe Ähnlichkeit fördert natürlich die Vertauschungswahrscheinlickeit...

Verfolgung Verfolgt ein Roboter den anderen, dann können diese sich in der Position, der Geschwindigkeit und der Ausrichtung sehr ähnlich sein. Die beiden letztgenannten Attribute können sogar identisch sein.

Frontalzusammenstoss Hierbei sind zwar Position und Ausrichtung verschieden, es ergeben sich aber spezielle Probleme, wenn man mit vorausschauenden Verfahren arbeitet. Diese müssen kurz vor dem eigentlichen Zusammenstoss genau genug vorhersehen, wo die Roboter danach stehen werden. Führt man einfach die Bewegung der Roboter gedanklich fort, so können die zukünftigen Positionen genau auf dem jeweils anderen Roboter liegen. Die Bewegungsrichtung scheint hier ein stärkeres Argument zu sein, jedoch vergesse man nicht, dass genau diese sich unmittelbar mit dem Zusammenprall umkehrt.

Nah aneiander Vorbeifahren Hier fahren zwei Roboter im Extremfall parallel zueinander mit minimalem Abstand, sodass ein Zusammenprallen gerade ausbleibt. Es gibt hier zwei Unterarten, je nachdem ob die Bewegungsrichtung dieselbe ist (bis auf Parallelverschiebung). Bei gleicher Bewegungsrichtung können bis auf die Position wieder alle (aktuellen) Attribute identisch sein. Speziell in dem Fall, dass ein Roboter den anderen überholt, also diese Situation in die Situation Verfolgung übergeht, kann es bei nur wenig Bildern unmöglich werden korrekt zu tracken. Der Fall entgegengesetzte Bewegungsrichtung kann Probleme bei vorausschauenden Verfahren hervorrufen, da es für diese schwer sein mag zu entscheiden, ob ein Zusammenprall stattfinden wird. Weiterhin kann dies Schwierigkeiten für Methodem hervorrufen, die versuchen Winkeländerungen auszunutzen, da diese zwischen zwei so schnell aneinander vorbeifahrenden Robotern sehr gross sind.

2.2.1.5 Diskutierte Methoden

Die zunächst diskutierten Verbesserungsideen des Trackings liessen sich hauptsächlich von der Betrachtung soeben genannter kritischer Situationen leiten. Zunächst, als die alte Methode noch nicht bis in die Detail verstanden war, sah es einfach so aus, dass diese nur mit der Nähe von Positionen arbeite und man deshalb weitere Informationen nutzen müsse. Daraus entstand die folgende Idee.

Benutzung von Winkeländerungen Diese Methode sollte auf dem bisherigen Verfahren aufsetzen. Es sollte nicht nur versucht werden, Positionsänderungen minimal zu halten, sondern auch Winkeländerungen. Gemeint sind hierbei die Winkel die jeder Roboter zu jedem anderen hat. Dies sollte besonders der Situation Verfolgung Rechnung tragen, die als besonders kritisch für das alte Verfahren angesehen wurde. Würde man im verfolgenden Fall nun eine Vertauschung gemäss dem alten Verfahren vornehmen wollen, so hätte dies eine große Winkeländerung zwischen Verfolger und Verfolgtem zur Folge. Waren die Winkel z.B. vorher 90 Grad und -90 Grad, so wären sie danach

praktisch umgekehrt. Letztlich muss dieses Verfahren diese zwei Argumente (kleine Positionsänderung, kleine Winkelveränderung) gewichten. Um sie vergleichbar zu machen, könnte man z.B. jeweils eine prozentuale Veränderung benutzen, wobei sich dann die Frage stellt, was 100% sind. Dazu kann man Maxima verwenden, also die maximale Positionsveränderung und die maximale Winkeländerung. Dieses Verfahren scheint so in der Situation Verfolgung tatsächlich eine Verbesserung zu sein. Beim Frontalzusammenstoss hatte die alte Methode keine besonderen Probleme und das scheint sich so nicht zu ändern. Das nah aneinander Vorbeifahren macht aber gewisse Schwierigkeiten, denn hier sind auch grosse Winkeländerungen zu erwarten. Letztlich entscheidet die Auflösung (Bilder pro Sekunde), ob das Verfahren greift. In jedem Fall scheint hier eine Verbesserung des alten Verfahrens vorzuliegen. Leider bedingt es aber einen recht hohen Aufwand bei der Gewinnung der nötigen Winkelinformationen. Bei n Robotern wären das n^2 Winkel. Da es hier um die Winkeländerung eines Roboters zu allen anderen Robotern geht, sind auf den ersten Blick sehr viele Vergleiche durchzuführen. Um das Verfahren praktisch nutzen zu können, erscheint es deshalb sinnvoll, es lokal einzuschränken. Durch die maximale Positionsveränderung, die durch die Auflösung und Maximalgeschwindigkeit der Roboter gegeben ist, kommen für eine bestimmte Position nur wenige Roboter in Frage. Genau für diese sollte man Winkelveränderungen erst als zusätzliches Argument in Betracht ziehen. Man könnte z.B. zunächst wie zuvor nach Positionsnähe zuordnen und dann betrachten, ob sich irgendwo dadurch große Winkelveränderungen ergeben. Ist das der Fall, so könnte man wie angedeutet lokal beschränkt optimieren.

Zukunftsberechnung Berechnet man die Entwicklung der aktuellen Situation mit großer Genauigkeit und führt danach einen Positionsvergleich wie bisher durch, dann sollte ein nahezu perfektes Tracking erreicht sein. Je genauer diese Berechung sein soll, desto aufwändiger wird sie. Deshalb zieht man zunächst vereinfachte Zukunftsberechnungen in Betracht. Eine besonders einfache besteht darin die Bewegung der Roboter in Kenntnis ihrer Ausrichtung und Geschwindigkeit einfach fortzuführen, d.h. ohne jegliche Kollisionsbetrachtung. Dies wurde deshalb ins Felde geführt, da es die Problemsituation Verfolgung löst. Leider erhält man dadurch beim frontalen Zusammenstoss besonders falsche Vorhersagen. Diese einfache Variante erbringt also keine Verbesserung. Eine Verbesserung schien eine Einbeziehung von Kollisionen zu erfordern. Was auch bedeutet, dass man die Grenzen des Spielfeldes (Banden), und die Ausmaße der Roboter einbeziehen muss. Auch stellt sich die Frage, wie man im Falle der Kollision dann die Bewegungen der Roboter fortführen soll. Eine implementierte Variante wird weiter unten vorgestellt.

Filtersystem Diese Methode stellt einen Rahmen zur Zusammenführung mehrerer Methoden dar und kann wohl im Zusammenhang mit beliebigen Verfahren eingesetzt werden. Da die Bildverarbeitung 2 auf Filtern basiert, hätte man so auch eine ästhetische Fortführung der Philosophie.

Der Grundgedanke besteht hier darin, zunächst alle Methoden einzusetzen, die mit (höchster) Sicherheit Kandidaten für einen neuen Roboter (die zuzuordnenden neuen Werte) ausschliessen. Nach diesem Ausschluss sind dann Heuristiken auf die Kandidaten anzuwenden.

Die zunächst einzusetzende Methode ist dann eine Menge von Filtern, die man nacheinander auf die Menge der Kandidaten anwendet. Z.B. könnte ein solcher Filter Roboter ausschliessen, die von einer zuzuordnenden Position zu weit entfernt sind, als das sie in Frage kommen könnten. Man kann so alle sicher ausschliessenden Argumente benutzen, die wohl letztlich auf physikalischen Grenzen der Roboter beruhen (z.B. maximale Beschleunigung, maximale Drehgeschwindigkeit...). Angenehm hierbei ist, dass man so leicht erweitern kann, falls einem weitere Filter in den Sinn kommen. Oder man könnte optimieren, falls man erkennt, dass eine bestimmte minimale Filtermenge ausreicht. Im Idealfall bleibt nach Anwendung der Filterkette genau ein Kandidat übrig. Dann hätte man eine sichere Zuordnung. Falls nicht, so wird man Heuristiken einsetzen müssen. Letztlich basieren diese wohl darauf geringere änderungen für wahrscheinlicher zu halten. Hat man mehrere Heuristiken, kann man diese zu einer verbinden, indem man sie gewichtet und zusammenzählt. Jede Heuristik muss dazu einen Zahlenwert liefern, der besagt, wie wahrscheinlich eine bestimmte Zuordnung ist. Letztlich wird dann die wahrscheinlichste gewählt.

Da jede Methode letztlich eine Heuristik bleiben muss, kann man sie auch in dieses System einbinden.

Zur Beurteilung dieser Methode kann man sagen, dass sie schliesslich nur einen Rahmen zur Einbindung anderer Methoden darstellt, der aber eine große Hilfe sein mag und in ein bequemes System münden kann.

2.2.1.6 Implementierte Methoden

Abwandlung der alten Methode durch Sortierung Die alte Methode bestand wie oben beschrieben aus zwei Schleifen bei denen bei fester Reihenfolge der Betrachtung einem Roboter derjenige neue Roboter mit der geringsten Distanz zugeordnet wurde. Nach der Zuordnung wurde der neue Roboter dann aus der Menge der neuen entfernt. Da zu einem späteren Zeitpunkt festgestellt wurde, dass die angesprochenen Problemsituationen ein geringeres Problem darstellen, als das Verlieren von Robotern (also die Anzahl der neuen Roboter ist geringer als die der alten), wurde die alte Methode

leicht abgewandelt, mit dem Ziel, dass die richtigen Roboter ihre alte Position beibehalten, falls deren neue nicht bekannt ist. Das liess sich, wie sich zeigte, erfolgreich durch folgende Sortierung erreichen. Man bilde dazu zunächst alle Paare von alten Robotern mit neuen. Man merke sich die (euklidischen) Distanzen zwischen diesen und sortiere die Liste von Paaren danach aufsteigend. Für all diejenigen Roboter, für die der neue zuzuordnende Roboter abhanden gekommen ist, erhält man so nun Paare bei denen der Distanzwert normalerweise recht hoch sein wird. Ist die Auflösung nur genügend hoch, dann wird der Distanzwert solcher falscher Paarung stets höher sein als der der richtigen. Ordnet man also nun entsprechend dieser sortierten Liste zu (man streiche dabei bereits zugeordnete Roboter!), dann werden die Roboter für die kein neuer existiert, keine Zuordnung erhalten und damit ihre alte Position beibehalten, was also genau der Absicht entspricht.

Zukunftsberechnung Ein aufwändigeres Verfahren wurde durch Implementierung einer Zukunftsberechung umgesetzt. Es findet sich in `robotsoccer/BV/BV2/Vorherseher2d`. Aus dem Namen kann man schliessen, dass es sich dabei um ein allgemeineres Vefahren handelt. Das ist richtig! Es ist für die Vorhersage von Bewegungen von beliebigen 2D-Objekten gedacht. Der Klasse Vorherseher2D übergibt man dazu einen Vector mit 2D-Objekten (dazu von der Klasse Objekt2d ableiten), die Zeit, die simuliert werden soll und einen Wert für die Genauigkeit der Simulation (SchrittweiteSek), denn diese wird zeitlich getaktet durchgeführt. Die konkret eingesetzte Simulation läuft dann folgendermaßen ab: mit äquidistanter Schrittweite wird die Gesamtzeit verstreichen gelassen. Bei jedem Zeitschritt werden alle 2D-Objekte (hier Roboter) dem Zeitschritt entsprechend bewegt. Dann werden all diese Objekte paarweise befragt, ob sie einen Punkt gemeinsam haben, also Kollision vorliegt. Ist dies der Fall, werden zwei Fälle unterschieden:

1. ein Objekt entfernt sich von dem anderen
2. ein Objekt bewegt sich auf das andere zu

Im ersten Fall wird die Simulation wie bisher fortgeführt. Im zweiten Fall wird das Objekt angehalten und im folgenden nicht mehr bewegt. Diese Art der Kollisionsbehandlung ist natürlich eine Vereinfachung, soll aber dem Rechnung tragen, dass zum einen bei Kollisionen der zweiten Art im Robotsoccersystem tatsächlich meist ein Stillstand erfolgt (z.B. beim wichtigen Frontalzusammenstoss) und weiterhin die Simulation so effizient und genau genug erfolgen kann. Der erste Fall ist von Bedeutung, wenn im Falle der Verfolgung eine Kollision erfolgt, da die Roboter in diesem Fall

weiterfahren werden. Nachdem derart dann die Zukunftsberechnung erfolgt ist, erhält man die vorhergesehenen Roboterwerte. Man kann nun einfach diese nehmen um anstatt alte Roboter mit neuen zu vergleichen nun vorhergesehene mit den neuen zu vergleichen.

2.2.1.7 Erfahrungen und Empfehlungen

Während der GI-Tagung und in einigen Testspielen zuvor wurden beide implementierete Verfahren getestet. In diesen Tests zeigte sich zunächst eine starke Verbesserung gegenüber der alten Methode. Man konnte beispielsweise auch Roboter vom Spielfeld nehmen und wieder draufsetzen und die korrekte Zuordnung blieb erhalten. Während des Turniers gab es dann kaum Probleme, die auf das Tracking zurückgeführt werden konnten. Spätere Erkenntnisse liessen vermuten, dass andere Probleme des Systems zu diesem Zeitpunkt schon derartig gewichtiger waren, dass eventuelle Trackingprobleme schon nicht mehr relevant waren. Obwohl die zweite Methode mit der Zukunftsberechnung als das wesentlich aussichtsreichere Verfahren erscheint, konnte man keine besondere Verbesserung gegenüber der zuvor genannten beobachten. Dies hängt wohl sehr damit zusammen, dass das Tracking zu diesem Zeitpunkt nicht mehr das Nadelöhr darstellte. Falls dies wieder der Fall sein sollte (z.B. wegen Vergrösserung des Spielfeldes, schnellerer Roboter oder schlechterer Kamera), ist zu vermuten, dass sich die Zukunftsberechnung klarer absetzen wird. Man sollte an der Genauigkeit dieser Methode arbeiten, falls dann noch eine Verbesserung vonnöten ist. Sollte das immer noch nicht genügen, implementiere man oben genanntes Filtersystem, füge soviele Filter wie möglich hinzu und kombiniere (falls nötig) ein paar Heuristiken.

2.2.2 Test der Bildverarbeitung

Links oben im Hauptfenster vom Programm “robotsoccer” sieht man unter dem Titel “Kontrolle” eine sich kontinuierlich aktualisierende Zahl “Kamera”(2.10). Dieser Bereich dient eigentlich für die Anzeige der durch die Bildverarbeitung (BV) analysierte, gesamte Frame-Anzahl innerhalb einer Sekunde.

Die aktuell verwendeten NTSC-Kameras von Samsung liefern bei einer Auflösung von 640×480 Pixeln insgesamt 60 Halbbilder pro Sekunde (bzw. 30 Vollbilder/s), unabhängig von der Leistung der Framegrabberkarte des Hostcomputers. Wenn die Framegrabberkarte ein Vollbild digitalisiert hat, speichert sie diese Bildinformationen sofort in einen bereitgestellten Ringbuffer. Dieses gespeicherte Bild wird daraufhin vom BV-Modul analysiert und sobald ein neues Bild von der Kamera gesendet wird, wird der nicht mehr benötigte Buffer gelöscht bzw. mit neuen Bildinformationen überschrieben.

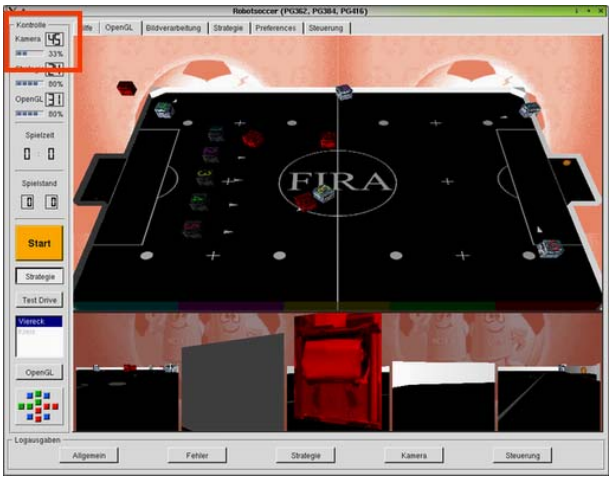


Abbildung 2.10: Programm “robotsoccer”

In dem Roboterfussball-System sind die Kamerabilder bis zum jetzigen Zeitpunkt die einzigen eingehenden Informationen. Entscheidend für die Performance des Gesamtsystems ist, wie genau diese Eingaben analysiert und wie schnell sie bearbeitet werden können. Auch eine höhere Framerate und eine höhere Auflosung der eingesetzten Kamera könnte die Leistung des System noch steigern.

Das BV-Modul sollte daher unbedingt getestet werden, damit man vertrauensvoll das gesamte System darauf aufbauen und weiter entwickeln kann.

2.2.2.1 Was man testet

Seit kurzem (Stand: Mitte Januar 2003) gibt es ein von Norman Weiss völlig neu entwickeltes BV-Modul. Ein Test dieses neuen Moduls bietet sich an, konnte aber bisher leider nicht durchgeführt werden, da es noch nicht zu 100% in das aktuelle System eingefügt worden ist.

Der Test des alten BV-Moduls ist aber trotzdem sinnvoll, weil das neue BV-Modul immer noch auf der alten Hardware (Samsung Kamera, Matrox Meteor I Framegrabber-Karte) basiert und mit demselben Linux-Treiber und demselben Strategie- und Kommunikationssystem zusammenarbeiten wird.

Aus diesem Grund werden Tests zur Messung einiger wichtiger Kennzahlen der Bildverarbeitung durchgeführt. Einer davon ist der Framezahl-Test, der die Anzahl der Frames, die innerhalb einer Sekunde von der Kamera zur Framegrabber-Karte übertragen und verarbeitet werden, misst. Bisher beträgt die Frameanzahl -30 Vollbilder pro Sekunde - ohne überprüft zu haben, ob der Test selbst korrekte Ergebnisse liefert. Deshalb wurden, basierend auf diesen Test, weitere Tests durchgeführt,

ob das Synchronisationssignal zwischen den Frames richtig erzeugt wird, ob alle durch die Kamera gelieferten Bilder bearbeitet werden und ob ein Frame wirklich nur einmal analysiert wurde.

2.2.2.2 Was man durch diese Tests gewinnen kann

Wenn man davon ausgehen kann, dass die Framegrabber-Karte wirklich 30 Vollbilder pro Sekunde bekommt und diese korrekt in den Ringbuffer speichert, wird die Latenzzeit eines einmaligen Durchlaufs durch alle Programmteile, von Kamera bis Robotersteuerung, genau auf unter 33 ms begrenzt. D.h. innerhalb von 33 ms sollten Bildverarbeitung, Strategie und Robotersteuerung abgearbeitet worden sein, damit man alle einkommenden Bildinformationen analysieren kann, um so die aktuelle Spielsituation auswerten und entsprechend schnell reagieren zu können.

Eine doppelte oder mehr als zweifache Bildanalyse des selben Frames bedeutet, dass das Roboterfussball-System nacheinander den gleichen Spielzustand bekommt. Daraus folgt, dass die Strategie zwei oder mehrfach nacheinander den selben Befehl ausgibt und somit die Roboter zwei oder mehrmals den gleichen Befehl erhalten.

Wir können uns z.B. einen Fall vorstellen: Nach der aktuellen Spielsituation soll sich der Roboter sofort nach links drehen und auf dem nächsten Spielzustandsbild sofort wieder nach rechts drehen. Wenn aber der Roboter den Befehl “Drehe nach links” zweimal nacheinander bekommen würde, der durch die doppelte Bildanalyse eines Frames gesendet wird, würde sich der Roboter nicht nach links und wieder nach rechts, sondern zweimal nach links drehen. Dieser Fehler kann, je nach Spielstand, fatale Folgen haben.

Der Torwart-Roboter kann sich unter diesen Bedingungen nicht exakt genug bewegen und auch die anderen Roboter haben ebenfalls Schwierigkeiten, ihre Bewegung zu kontrollieren. Der schlimmste Fall wäre ein sehr häufiges Auftreten dieses Fehlers während eines Spiels.

2.2.2.3 Annahme für den Test

Für den Test müssen bestimmte Voraussetzung gegeben sein:

Fehlerfrei installierte Framegrabber-Karte, mit den dazugehorigen Treibern, basierend auf einem Linux-System, die Kamera liefert die Bilder mit maximaler Geschwindigkeit, usw. Weil die meisten Tests eine genaue Zeitmessfunktion brauchen, wird die Zeitmessfunktion “gettimeofday()” als geeignete Funktion betrachtet. Für die Testausgabe wird die “fprintf()” Funktion benutzt. Angenommen ist,

dass die Latenzzeit von dieser fprintf() Funktion sehr gering und somit vernachlässigbar ist. In der Praxis benötigt eine Ausgabezeile mit der fprintf() Funktion unter 0,001 ms.

2.2.2.4 Durchgeführte Tests

Test 1: FPS Getestet wird, ob die Framegrabberkarte genau 30 Frames pro Sekunde liefert. Das Testprogramm ist in der "startcam()"-Funktion von Weltmodell.cpp integriert. Fur jede Sekunde wird die Framecounterzahl, die durch den Treiber erzeugt wird, ausgelesen und in die Ausgabedatei "FrameAnzahlTestDaten_1.txt" geschrieben.

Sourcecode für Test 1:

```
void Weltmodell::startcam(){

  static double t0 = 0, t5 = 0, counter=1;
  t0=elapsed();
  while (!thread2Cancel) {
      FILE* myBVtest = fopen("FrameAnzahlTestDaten_1.txt","a+");
      t5=elapsed();
      if(t5-t0 > counter){
        fprintf(myBVtest,"%f\t",t5-t0);
        fprintf(myBVtest,"%d\n",BV2Space::fg_getfps2());
        counter++;
      }
   fclose(myBVtest);
   }
}
```

Ergebnis siehe Abb. 2.11.

Analyse Man sieht in 2.11 , dass 30 Frames pro Sekunde nicht durchgehend, aber zumindest relativ regelmäßig zur Verfügung stehen. Trotzdem sollte man beachten, dass nicht jeder neue einzelne Frame exakt einmal pro 33 ms in der Ringbuffer gespeichert wird. Deswegen sollte die gesamte Analyse nur auf das Synchronisationssignal zwischen den Frames synchronisiert werden. Außerdem sollte ein Durchlauf von der Bildverarbeitung bis zur Robotsteuerung innerhalb von 33 ms abgeschlossen sein.

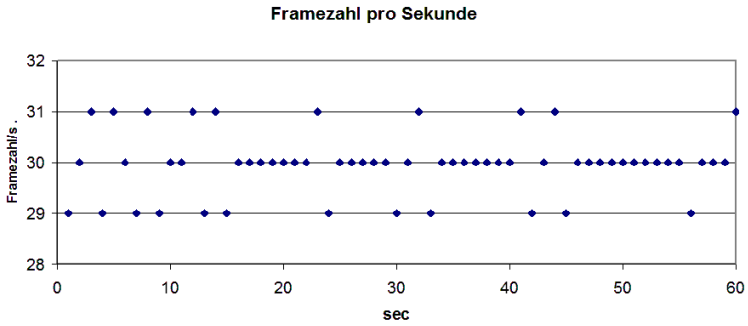


Abbildung 2.11: Framezahl pro Sekunde

Test 2: Doppelte Verarbeitung einzelner Bilder Getestet wird, ob ein Bild mehr als 2 mal verarbeitet wird.

Sourcecode für Test 2:

```
if (ignore_sigusr1)
{
    while ((elapsed() - t0) < 0.033333);
}
else
{
    c0=elapsed();
    sem_wait(&kameraSemaphore);
    c1=elapsed();
    FILE* myBVtest = fopen("FrameAnzahlTestDaten_2.txt", "a+");
    fprintf(myBVtest,"%f\t",c1-c0);
    fprintf(myBVtest,"%d\n",BV2Space::fg_getfps2());
    fclose(myBVtest);
}
```

Ergebnis siehe Abb. 2.12

Analyse In diesem Graphen(Abbildung 3) bedeutet die Y-Achse “Frame Abstand”. Das ist der Frameabstand (z.B. 1 Frame) zwischen dem neuen Frame (X-Achse, z.B. 4831. Bild), der direkt nach der Analyse eines Frames abgeholt wurde und dem zuvor analysierten Frame (z.B. 4830. Bild).

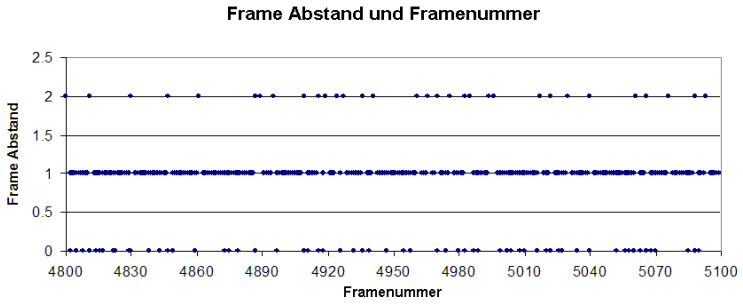


Abbildung 2.12: Frame Abstand und Framenummer

Wenn die BV schnell genug und framemäßig synchronisiert ist, dann sollte der Frame Abstand immer 1 sein. Frame Abstand ist großer als 1, wenn die BV länger als 33 ms benötigt und 0 bedeutet eine doppelte Analyse des selben Frames.

Wie zu erkennen ist, wird sehr oft ein Bild mehr als einmal bearbeitet und als aktuelles Bild anerkannt. Obwohl eine Vermeidungsfunktion `sem_wait( &kameraSemaphore)` vorgesehen war, wurde scheinbar nach der Implementierung nicht überprüft, ob diese korrekt arbeitet. Die Vermeidungsfunktion ist für den Fall implementiert, dass die gesamte Analyse für ein Bild - von BV bis Robotersteuerung- in kurzer Zeit als ein 33 ms-Intervall abgearbeitet ist oder mehr als ein Intervall gedauert hat. Auf jeden Fall soll BV bis auf das nächste Synchronisationssignal des Frames warten und danach beginnen, die nächste Analyse mit dem Bild, das zu diesem Zeitpunkt im Ringbuffer gespeichert ist, durchzuführen. Wahrscheinlich hat `sem_wait( &kameraSemaphore)` eine Fehlfunktion. Also wurde folgender Code nach der if-Anweisung geschrieben und noch mal getestet.

```
while (sem_trywait(&kameraSemaphore)==0)
{;}
```

Ergebnis Siehe Abb. 2.13.

Analyse Nun funktioniert `sem_wait( &kameraSemaphore)` korrekt. Es gibt nun zumindest keine mehrmalige Benutzung von ein und demselben Bild, aber der Frame Abstand ist nicht immer 1, weil das zu testende BV-Modul manchmal zu langsam läuft. Das liegt auch daran, dass dieses BV-Modul vorher noch nie getestet bzw. optimiert wurde.

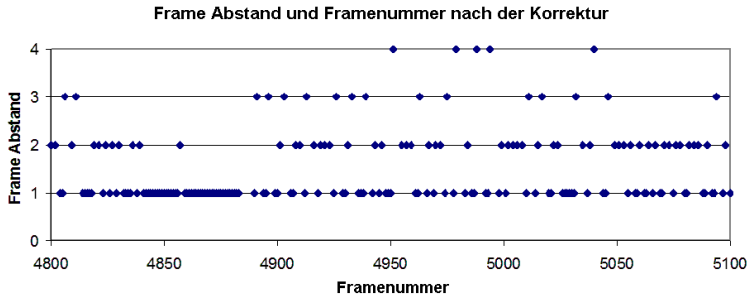


Abbildung 2.13: Frame Abstand und Framenummer nach der Korrektur

Test 3: Latenzzeit Die Latenzzeit der Bildverarbeitung, der Robotersteuerung und des Funks werden gemessen und analysiert.

Sourcecode für Test 3:

```
void Weltmodell::startcam(){
    static double t0 = 0, t1 = 0, t2 = 0, t3 = 0, t4 = 0;
    while (!thread2Cancel) {
if (ignore_sigusr1)
{
        while ((elapsed() - t0) < 0.033333);
}
else
{
     sem_wait(&kameraSemaphore);
}
     while (sem_trywait(&kameraSemaphore)==0) {;}

     t0=elapsed();
     bv->update();
     t1=elapsed();
     fprintf(myBVtest,"%f \t",t1-t0);

     t2=elapsed();
     for (i=0;i<numberOfRobots;i++){
```

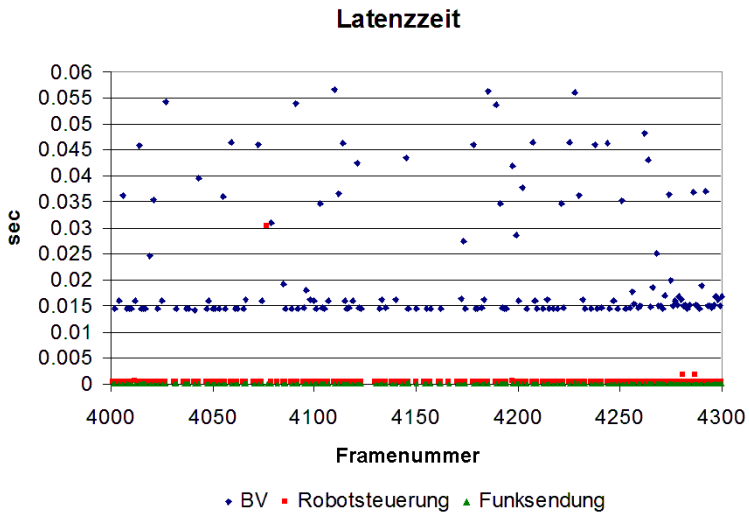


Abbildung 2.14: Latenzzeit

```
        freunde[i]->updateRS();
    }
    t3=elapsed();
    fprintf(myBVtest,"%f \t",t3-t2);

    if(!(++fm_counter%1)) {
        fm->update();
    }
    t4=elapsed();
    fprintf(myBVtest,"%f \t",t4-t3);
    ...
   }
}
```

Ergebnis Siehe Abb. 2.14.

Analyse Diese Latenzzeitmessung ist sehr abhängig von der Optimierung der Farbeinstellungen und der Lichtverhältnisse auf dem Spielfeld(siehe Seite 156,“Systematische Farbeinstellung mittels Graphen”). Man sieht, dass die BV den größen Anteil der Analysezeit einnimmt. Den zweitgrössten Anteil benötigt die Robotersteuerung. Außerdem benötigt die BV im schnellsten Fall nur 13,585 ms,

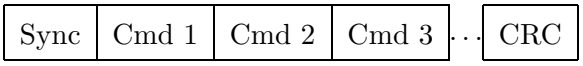
aber im schlimmsten Fall bis zu 194,284 ms. Diese höhen Schwankungen könnten durch Verbesserungen im BV Modul verringert werden.

2.3 Funk

Nach einigen Tests mit dem damals benutzten Funkprotokoll der PG 362[PG362, S. 36], Gesprächen mit Teilnehmern vorangeganger Projektgruppen und dem Studium der Datenblätter der Funkmodule [Rad02a, Rad02b, Rad02c], ergaben sich ehrere verbesserungswürdige Punkte.

Bei der Benutzung des Funkprotokolls der PG 362 wird die Übertragungsrate auf 9600 Baud beschränkt, weil entgegen den Empfehlungen im Datenblatt, keine bitsymmetrischen Daten übermittelt werden und deshalb bei höheren Übertragungsraten die Fehlerrate soweit ansteigt, dass am Ziel keine brauchbaren Daten mehr ankommen. Auch wird die Empfehlung, nach Sendepausen für 3 ms eine Präambel mit dem Muster 01010101 zu senden, ignoriert. Die Möglichkeit einer bidirektionalen Kommunikation wird mangels Implementierung nicht genutzt.

Die bislang verschickten Datenpakete sehen folgendermaßen aus (s. [SB416, S. 168]):



- Sync

  Besteht aus einem Byte, das zur Synchronisation der Datenübertragung dient.

- Cmd 1, Cmd 2,...

  Befehle für die Roboter 1, 2, ...

- CRC

  Überprüfung, ob das Paket korrekt übertragen wurde.

Während die Datenpakete bei Spielen mit drei Robotern mit durchschnittlich 11 Bytes noch relativ kurz sind, ist die Größe bei 5 Robotern bereits auf 17 Bytes angewachsen. Da die Fehlerwahrscheinlichkeit abhängig von der Länge der Datenpakete ist, ist es wünschenswert, dass ein neues Funkprotokoll jeden Befehl in einem eigenen Datenpaket verschickt, damit sich ein Fehler nur auf einen und nicht auf alle gesendeten Befehle auswirken kann.

Das von Thomas Klute im Rahmen seiner Diplomarbeit [Klu01, S. 111ff] entwickelte Funkprotokoll wurde von den Vorgänger-PGs nicht benutzt, weil zum einen die bidirektionale Kommunikation, die für dieses Protokoll zwingend notwendig war, auf dem eingesetzten Linux-Systemen nicht zuverlässig funktionierte, zum anderen, weil bei der Benutzung des alten Funkprotokolls mehr Bandbreite zur Verfügung stand.

Ziele für die Implementierung eines neuen Funkprotokolls sind somit:

- Ansteuerung des Funkmoduls gemäß Datenblatt, um die Fehlerrate zu minimieren
- Nutzung der höheren Datenraten (19200, 38400 Baud)
- Möglichkeit einer bidirektionalen Kommunikation
- Ein separates Datenpaket für jeden Befehl
- Die Länge der Datenpakete niedrig halten

Als Basis für die Implementierung wurde das Funkprotokoll der PG 362 genutzt.

2.3.0.5 Implementierung

Aufgrund der benutzten Funkmodule kann eine bidirektionale Kommunikation während des Spieles nicht sinnvoll eingesetzt werden. Nach jeder Sendepause muss eine 3 ms lange Präambel gesendet werden, was dazu führt, dass für das Senden eines Pakets und für das anschließende Empfangen der Anwort alleine 6 ms für die beiden Präambeln aufgewendet werden müssen. Bei 5 Robotern wären somit alleine durch das Senden der Präambeln die Funkstrecke für 30 ms blockiert. Da alle 33 ms neue Befehle an die Roboter geschickt werden, ist die bidirektionale Kommunikation somit nur in den Spielpausen, zum Beispiel für Debugzwecken nutzbar.

Damit die im Datenblatt geforderte Bitsymmetrie gewährleistet wird, werden die Daten nach dem Manchester Standard codiert. Hierbei werden für jedes Datenbit zwei Bits übertragen. Das eine Bit entspricht dem Datenbit, während das zweite Bit dem Komplement dessen entspricht. Durch dieses Verfahren wird die geforderte Bitsymmetrie erreicht, jedoch verringert sich die effektive Übertragungsrate um die Hälfte. Dieses kann durch die Erhöhung der Datenrate von 9600 auf 19200 Baud ausgeglichen werden. Durch die jetzt mögliche Benutzung von 38400 Baud verdoppelte sich sogar die effektive Übertragungrate gegenüber 9600 Baud.

Zur Vermeidung der 3 ms langen Präambel nach jeder Sendepause wurde die Übermittlung der Daten an die serielle Schnittstelle in einen eigenen Thread ausgelagert. Die Roboterbefehle werden zuerst in einen Ringbuffer geschrieben und anschließend vom Funkthread ausgelesen. Sollte der Ringbuffer leer sein, werden solange bitsymmetrische Daten (0x55) gesendet bis sich wieder Daten im Ringbuffer befinden. Bei der Implementierung dieses Threads traten verschiedene Probleme auf, deren Lösung teilweise recht aufwendig waren.

Linux benutzt neben dem in der seriellen Schnittstelle vorhanden Buffer noch einen internen Buffer. Leider ist keine Möglichkeit implementiert, um festzustellen, wie weit diese Buffer gefüllt sind. Dieses

ist jedoch notwendig, weil man mit dem Senden von bitsymmetrischen Daten in den Sendepausen erst beginnen will, wenn der Buffer leer ist, da anderenfalls nicht mehr die Möglichkeit besteht, neu eingetroffene Daten im Ringbuffer sofort zu senden. Eine suboptimale Lösung ist, mit Hilfe des folgenden Codesegments darauf zu warten, dass der Hardware-Buffer komplett leer ist und danach erst beginnt neue Daten zu senden:

```
do
{
ioctl(cuafd, TIOCSERGETLSR, &bits);
} while (bits != TIOCSER_TEMT);
```

Nachteil dieser Lösung ist, dass bei 19200 Baud ungefähr alle 0,5 ms ein neues Byte an die Schnittstelle geschickt werden muss.

Damit der Funkthread nicht permanent die Endlosschleife durchläuft und somit unnötig Rechenzeit verbraucht, muss der Thread für ungefähr 0,5 ms anzuhalten sein. Der Befehl Usleep erfüllt diese Anforderung nicht, weil die theoretische Auflösung bereits bei einer Millisekunde liegt und die tätsächliche sogar nur bei 5 ms. Durch die Nutzung der internen Realtime-Clock, deren Frequenz bis zu 8192Hz einstellbar ist, kann der Thread für die gewünschte Zeit angehalten werden. Der Zugriff auf die Realtime-Clock erfolgt unter Linux über das Device /dev/rtc [Gor02]. Das Device /dev/rtc hat die Aufgabe, Prozesse, die lesend auf dieses Devices zugreifen, solange anzuhalten, bis die Realtime-Clock einen Interrupt generiert. Da normale User jedoch maximale eine Frequenz von 64Hz einstellen können, ist eine Veränderung am Kernel notwendig, damit höhere Frequenzen einstellbar sind.

Weiterhin hat der Linux-Scheduler (s. [Heu99, Der Linux Scheduler]) die Eigenschaft, Threads zeitweise für mehr als 5 ms keine Rechenzeit zuzuweisen. Hierdurch wird die harte Echtzeitbedingung von 0,5 ms verletzt. Aus diesem Grund wurde für den Funkthread das Round Robin Scheduling (s.[Heu99, Austauschbarkeit des Schedulers]) aktiviert, da ein solcher Thread nur mit minimaler Verzögerung Rechenzeit zugewiesen bekommt, wenn der Prozess nicht mehr angehalten wird. Da diese Art von Zuweisungen normalerweise Root-Rechte erfordern, wird ein Suid-Root [1] Programm benutzt, das das Round Robin Scheduling für beliebige Prozesse aktivieren kann.

[1] Suid-Root Programme können von normalen Benutzern gestartet werden, aben jedoch die gleichen Rechte, wie Programme, die als Root gestartet erden.

2.3.0.6 Aussichten

Die meisten im vorherigen Kapitel beschriebenen Probleme sind darauf zurückzuführen, dass Linux die Interrupts der seriellen Schnittstelle nicht an Programme weiterleiten kann. Aus diesem Grund wäre es wünschenswert, ein Kernelmodul zu schreiben, dass auf die serielle Schnittstelle zugreifen kann und speziell auf die Bedürfnisse unserer Funkübertragung abgestimmt ist. Leider war es innerhalb der PG nicht möglich, ein solches Kernelmodul zu implementieren. Als Basis für ein solches Modul könnte Lirc[Lir02] benutzt werden. Lirc ist ein Programmpaket, welches eine IR-Fernbedienung durch eine an der seriellen Schnittstelle angeschlossenen IR-Diode simuliert. Da die harten Echtzeitbedingungen für diesen Aufgabenbereich schärfer sind als für die Funkübertragung, sollte das Kernelmodul als Basis geeignet sein.

Ein alternativer Ansatz wäre, zwischen der Sendestation und der seriellen Schnittstelle des Hostsystems einen DSP zu schalten, der die Kommunikation mit der Sendestation übernimmt und auf der anderen Seite dem Hostsystem eine bidirektionale und fullduplexfähige Kommunikation vortäuscht. Die Idee, den zur Zeit genutzten DSP TMS320F240 [Tex02] von Texas Instruments für diesen Zweck einzusetzen, wurde verworfen, weil dieser DSP nur eine serielle Schnittstelle hat. Realisiert werden kann diese Idee erst, wenn der neue DSP TMS320F2812 [Tex03] verfügbar und einsatzbereit ist.

2.3.1 Aufbau eines Datenpakets

Die nachfolgende Grafik verdeutlicht den neuen Aufbau der Datenpakete.

SYNC	Start	CRC	RobotID	CMD	DATA
1-2 Bytes	1 Byte	2 Bytes	1 Byte	1 Byte	befehlsabhängig

- SYNC

  Sync besteht je noch Funkmodul-Modell aus einem oder zwei 0xFF Bytes. Dieses ist laut Datenblatt notwendig, damit sich die Empfangsstation auf den Datenstrom einstellen kann. Die Anzahl der 0xFF Bytes kann über die Datei Prefs.conf geändert werden, wobei Tests keine signifikanten Verbesserung der Datenübertragung bei der Verwendung von zwei 0xFF Bytes anstatt von nur einem 0xFF Byte ergeben haben.

- Start

  Start besteht aus einem 0x8E Byte und markiert den Anfang eines Datenpakets. Das 0x8E-Byte wurde gewählt, weil es bitsymmetrisch (10001110) ist und zugleich in der von uns benutzen

Manchester Codierung keinem gültigen Zeichen entspricht. Da dadurch die Menge der Steuerzeichen und die Menge der für die Nutzdaten benötigten Zeichen disjunkt sind, entfällt das sonst notwendige Byte-Stuffing (s. [Klu01, S. 116f]).

- CRC

  Mit Hilfe der CRC wird überprüft, ob das Datenpaket korrekt übermittelt worden ist.

- RobotID

  Gibt an, für welchen Roboter der Befehl bestimmt ist. Falls die bidirektionale Kommunikation zum Einsatz kommt, ist der Wert 0 für das Hostsystem reserviert.

- CMD

  Spezifiziert das Kommando. Im Kapitel 2.3.2 werden die einzelnen Befehle näher erläutert.

- DATA

  Optionale Daten, die für das Kommando notwendig sind. Die Länge dieses Bereiches ist von dem Befehl abhängig.

2.3.2 Befehle

Es werden im wesentlichen zwei Arten von Befehlen unterschieden. Zum einen Befehle, die für die Bewegung des Roboters zuständig sind (Stop, Radgeschwindigkeiten, DeltaSpeed) und zum anderen Befehle, die nicht direkt eine Bewegung auslösen, sondern z.B. DSP Parameter ändern oder Daten zwischen dem Hostsystem und dem Roboter transferieren.

2.3.2.1 STOP

Setzt die Radgeschwindigkeiten auf 0.

Da keine aktive Bremsung (z.B. durch kurzzeitige Setzen entgegengesetzter Radgeschwindigkeiten) implementiert ist, rollt der Roboter aus und wird nur durch Reibung gebremst.

2.3.2.2 Radgeschwindigkeiten

Mit Hilfe dieses Befehls werden die Geschwindigkeiten der Räder direkt gesetzt.

Aufbau von DATA:

linke Radgeschwindigkeit	rechte Radgeschwindigkeit
10 bits	10 bits

Da für jede Radgeschwindigkeit effektiv nur 9 Bit [2] vorhanden sind, ergibt sich somit eine maximale Genauigkeit von $\frac{2000\,\mathrm{mm}/s}{2^9} = 3,9\mathrm{mm/s}$.

2.3.2.3 DeltaSpeed

Bei diesem Befehl wird dem Roboter mitgeteilt, in welche Richtung er fahren, um wieviel Grad er sich drehen und wie hoch die Endgeschwindigkeit sein soll.

Aufbau von DATA:

Geschwindigkeit	Richtung	Winkel
11 bits	1 bit	8 bits

Für den Winkel ergibt sich eine Genauigkeit von 1,4°, für die Geschwindigkeit von 0,97 mm/s.

2.3.2.4 Position anfahren

Dieser Befehl ist nur aus historischen Gründen hier aufgeführt. Da dieser Befehl bei unseren Tests nicht funktionierte und er auf dem CMU-Algorithmus basiert wurde er komplett entfernt.

2.3.2.5 Nichtstun

Die Bezeichnung “Nichtstun” hat historischen Gründe. Anderes als der Name assoziiert, bewirkt dieser Befehl, dass der Roboter weiterhin seinen zuletzt übermittelten Fahrbefehl ausführt und der Timeout-Zähler zurückgesetzt wird.

2.3.2.6 Parameter

Mit Hilfe dieses Befehls können verschiedene Parameter des Roboters (Rampeneinstellung, usw.) verändert werden.

Aufbau von DATA:

Parameter-Nummer	Wert
8 bits	8 bits

[2] Ein Bit ist für die Bestimmung der Fahrtrichtung notwendig.

2.3.2.7 Token

Mit Hilfe dieses Befehls kann das Hostsystem einen Roboter veranlassen, vom Empfangs- in den Sendemodus umzuschalten. Der Roboter sendet umgehend einen Tokenbefehl zum Hostsystem (ROBOTID=0) zurück und kann im DATA-Bereich Daten unterbringen, die vom Hostsystem verarbeitet werden können.

Länge	Daten
4 bits	variable (max. 15 Bytes)

Störanfälligkeit

2.3.3 offene Tests

Anforderungen Die per Funk übertragenen Daten müssen korrekt und möglichst schnell bei den Robotern ankommen.

Mögliche Fehler

- Funkpakete werden nicht korrekt übertragen.
- Funkpaktete werden fehlerhaft übertragen.
- Die Roboter reagieren zu langsam, weil die Latenzzeiten zu groß sind.

Güte

1. Güte $= \frac{\text{korrekte Funkpakete}}{\text{Funkpakete insgesamt}}$
2. Güte $= \frac{1}{\text{Latenzzeit}}$

Beide Gleichungen sollen auf ihrem Definitionsbereich maximiert werden.

Entfernungstest

- bisherige Annahme

  je weiter die Sendestation von der Empfangsstation entfernt steht, desto mehr Pakete gehen verloren bzw. werden fehlerhaft übertragen.

- Testverfahren

  Die Sendestation immer weiter von der Empfangsstation wegbewegen, kontinuierlich Pakete versenden und die Fehlerrate bestimmen. Als Sende- und Empfangsstation sollten sowohl normale Computer als auch die Roboter zum Einsatz kommen.

  Der Test wird für die verschiedenen Geschwindigkeiten (9600, 19200, 38400, 57600 (nur BIM 2 Module)), für die verschiedenen Serien und mit Paketen durchgeführt, deren Verhältnis von Nullen zu Einen variert. (60:40/50:50/40:60 usw.)

- Durchführbarkeit

  möglich

- Bemerkung

  Bei diesem Test kann der Durchsatz ermittelt werden.

Latenzzeit

- Testverfahren

  1. Latenzzeit der seriellen Schnittstelle + Linux

     Die beiden seriellen Schnittstellen eines Rechner mit Hilfe eines Null-Modem-Kabels verbinden und die Zeit zwischen Absenden und Empfangen messen.

  2. Latenzzeit inkl. Funkstrecke

     An die beiden seriellen Schnittstellen eines Rechner jeweils eine Funkstation anschließen. Von einer Station zur anderen Pakete senden und die Zeit zwischen dem Absenden und dem Empfangen messen. Den Test mit unterschiedlichen Abständen wiederholen.

  3. Latenzzeit in der bidirektionalen Kommunikation

     Roboter sendet ein Paket mit Timestamp, Computer schickt Antwort zurück und Roboter sendet zweites Paket mit Timestamp.

     Dieser Test macht es nicht erforderlich, dass die Uhren des Roboters und des Computers synchron laufen. Die Latenzzeit des Roboters geht jedoch doppelt in die Zeitmessung ein, während die des Computers nur einfach eingeht.

- Durchführbarkeit

  möglich

- Bemerkung

  Dieser Test sollte mit verschiedenen Funkmodulen durchgeführt werden, um festzustellen, ob es Unterschiede zwischen den Funkmodulen einer Serie gibt.

2.4 Oberfläche und Bedienung

2.4.1 Neustrukturierung der Preferences

Das Robosoccer-Projekt ist sehr stark konfigurierbar. Es können Einstellungen für alle möglichen Bereiche verändert werden, angefangen mit den Speicherorten für bestimmte Log-Dateien über die verwendete Bildverarbeitung bis hin zu den Parametern der Rampen.

Bisher war es so, dass diese einfach in einer Datei prefs.cfg nach dem Muster

```
# Kommentar
Variable = Wert
[...]
```

also z.B.:

```
# Geschwindigkeitsbegrenzung der Roboter (in mm/sec)
maxBotSpeed = 1500
```

abgespeichert wurden und beim Programmstart eingelesen wurden.

Die Gesamtzahl der verwendeten Variablen ist mittlerweile ca. 250. Das ursprüngliche Muster ist auf Dauer nicht mehr geeignet, diese große Menge an Einstellungsmöglichkeiten effizient zu verwalten. Außerdem hat es sich bei Turnieren gezeigt, dass das System in dieser Form kompliziert in der Benutzung ist. Es ist aus diesem Grund notwendig geworden, die Art und Weise, wie auf diese Variablen zugegriffen werden kann, benutzerfreundlicher zu gestalten.

Konkret gab es folgende Anforderungen:

1. Die Einstellungen sollten in einem grafischen Menü geändert werden können.
2. Die Einstellungen sollten zur Laufzeit vorgenommen werden können.
3. Es sollte eine Möglichkeit geben, die Änderungen schnell zu übernehmen.
4. Verschiedene Konfigurationen sollten im Bedarfsfall importiert bzw. exportiert werden können.
5. Es sollte Schutzmechanismen geben, die unsinnige Einstellungen verhindern.

Die Struktur der Prefs.cfg Es wurde entschieden, die Datei prefs.cfg als zentrale Preferences-Datei beizubehalten, sie aber grundlegend in ihrer Struktur zu verändern. Ein Eintrag ist nun wie folgt aufgebaut:

```
# Kommentar
Variable = AKTUELLER WERT; min = ZULÄSSIGES MINIMUM;
max = ZULÄSSIGES MAXIMUM; default = VOREINSTELLUNG; kat = KATEGORIE;
[...]
```

Der oben gezeigte Ausschnitt sieht somit folgendermaßen aus:

```
# Geschwindigkeitsbegrenzung der Roboter (in mm/sec)
maxBotSpeed = 1500; min = 0; max = 1000; default = 1500; kat = steuerung;
[...]
```

Bedienung über die grafische Oberfläche Die Bedienung ist intuitiv. Dank der Begrenzung des Wertebereichs ist es in den meisten Fällen nicht möglich, unsinnige Werte anzugeben. Sofort nachdem in der GUI ein Wert verändert wurde, wird dieser auch in der prefs.cfg verändert - allerdings wirkt sich das noch nicht auf die aktuelle Instanz des Programms aus, da die Voreinstellungen in der Regel zu Initialisierungszwecken benutzt werden. Erst nachdem ein Neustart (z.B. über den Neustart-Button) herbeigeführt wird, gelten die veränderten Einstellungen.

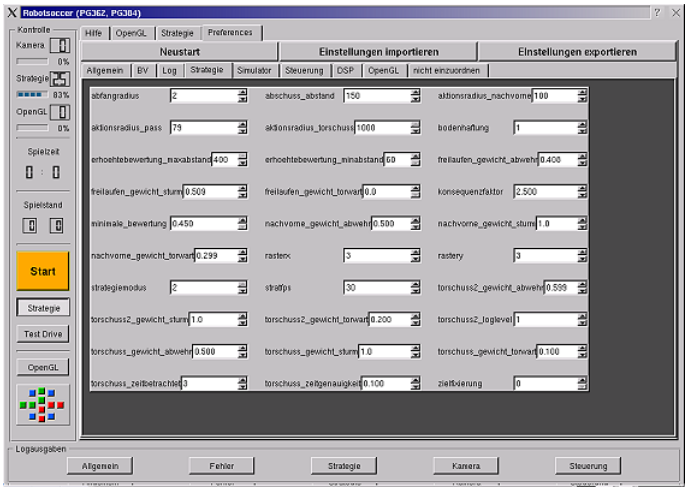


Abbildung 2.15: Der Preferences-Tab

Der Neustart-Button Mit diesem Button wird ein Neustart des Weltmodells ausgelöst: Alle Threads des laufenden Programms werden vernichtet und das Programm robosoccer wird über einen execl-Befehl neu gestartet.

Grober Überblick über die Funktionsweise Beim Start von *robosoccer* wird die Datei prefs.cfg ausgelesen. Mithilfe der Klasse Preferences wird jeder Parameter der Datei ausgelesen und dynamisch ein passendes GUI-Objekt angelegt, bis keine Parameter mehr da sind. Es ist also problemlos möglich, neue Parameter in der o.g. Datei hinzuzufügen - solange man sich an die angegebene Form hält.

Hinzufügen von neuen Parametern Wie im vorhergehenden Abschnitt beschrieben, können problemlos neue Parameter hinzugefügt werden. Dafür ist es lediglich notwendig, die Parameter irgendwo in der Datei prefs.cfg im oben beschriebenen Format anzulegen.

Import und Export von Einstellungen Manchmal ist es wünschenswert, bestimmte Einstellungen zu speichern, um später wieder auf sie zurückgreifen zu können. Eine exportierte Prefs-Datei hat immer automatisch die Endung '.cfg'.

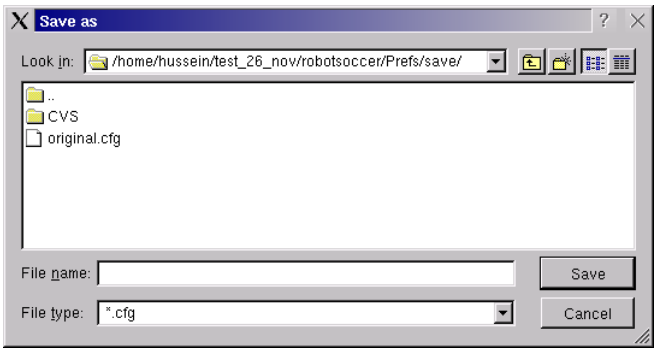


Abbildung 2.16: Der Export-Dialog

2.4.1.1 Testdrive

Testdrive dient zum Testen der Robotersteuerung. Nachdem man auf der GUI den Button "Test Drive" betätigt hat, beginnt der Roboter mit der ID 1, entsprechend des ausgewählten Tests, ein bestimmtes Muster abzufahren. Zur Zeit sind zwei Tests implementiert:

- Fahrt entlang eines Rechtecks: Siehe Abb. 2.17.

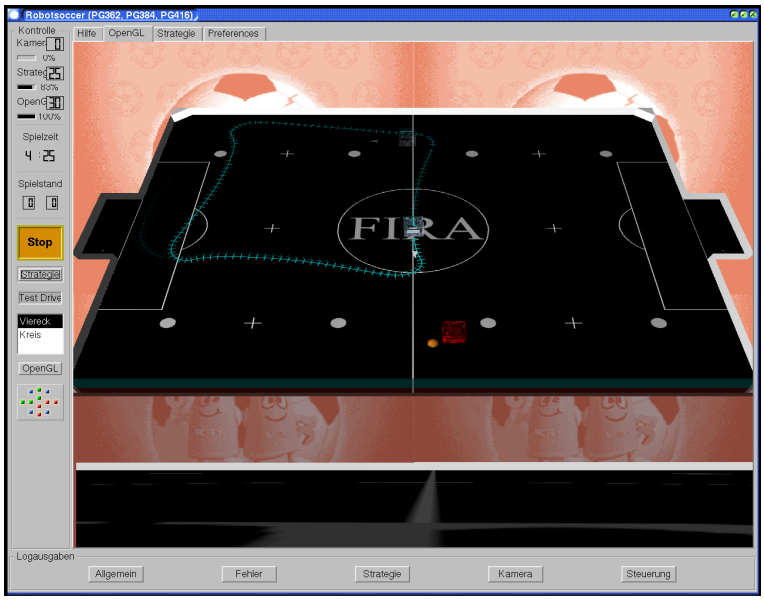


Abbildung 2.17: Fahrt entlang eines Rechtecks

- Fahrt auf einer Kreisbahn: Siehe Abb. 2.18.

Neue Tests können leicht hinzugefügt werden:

1. In der Klasse Testdrive muss eine Funktion hinzugefügt werden, die den gewünschten Test ausführt.
2. Die Case-Anweisung in der Funktion Testdrive::fahre() muss entsprechend erweitert werden.
3. Die Listbox auf der GUI muss um einen Eintrag erweitert werden (Piwo/Hauptfenster.cpp).

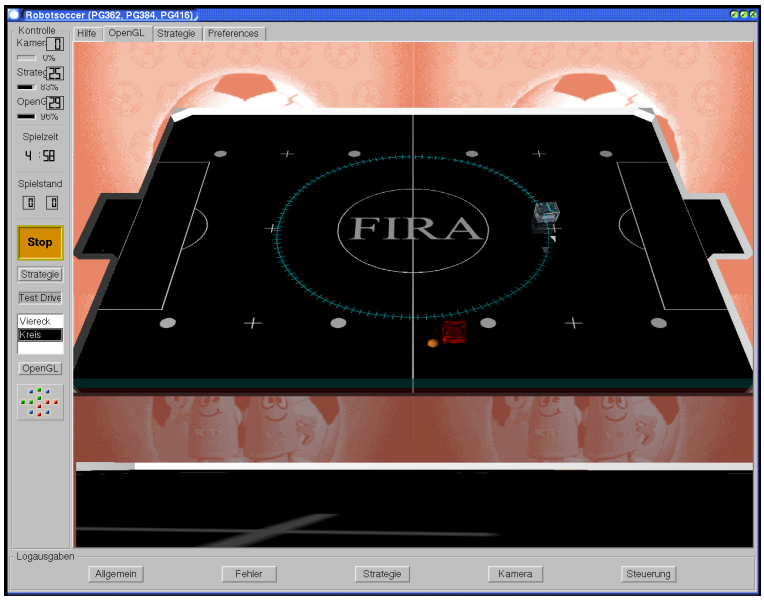


Abbildung 2.18: Fahrt auf einer Kreisbahn

2.5 Allgemeine Probleme

2.5.1 Restrukturierung der Headerdateien

2.5.1.1 Ausgangslage

Im Robotsoccerprojekt befindet sich in einigen Teilen ein eher unkonventionelles System, was die Bezeichung der Headerdateien angeht. Damit einher geht vielfach eine ebenfalls kritisierte Benutzung von Namespaces.

Der konventionelle Aufbau eines C++ Programms sieht - ab einer gewissen Projektgrösse - die Koexistenz von einerseits Headerdateien und andererseits Implementierungsdateien vor. Damit liegen dann zumeist Paare in Form von .h und .cpp Dateien vor. Die Idee ist nun in den .h Dateien nur die Deklarationen aufzubewahren, also z.B. die Deklaration einer Klasse und in den .cpp Dateien die zugehörige Implementation. Das sieht dann ungefähr so aus:

1. Headerdatei:

```
#ifndef _KLASSEA_H

#define _KLASSEA_H
```

```
class KlasseA
{

  public:

    void methode1 ();

  protected:

  private:

}

#endif
```

2. Implementationsdatei:

```
#include "Klasse.h"

void KlasseA::methode1()
{
  doSomething ();
}
```

Der Sinn dieser Vorgehensweise ist es, Struktur in größere Projekte zu bringen. Man kämpft dort meist darum die Übersicht zu behalten und dieses Konzept soll einem dabei eine Hilfe sein.

Das Konzept welches sich zum Teil noch im Robotsoccerprojekt befindet, umfasst dagegen eine .cpp Datei und 3 Headerdateien (.h1, .h2, .h3). Der einzige Sinn dieser Aufteilung ist letztlich die Möglichkeit verschiedene Teile einer Klassendeklaration bzw. -definition einzuziehen. Das sind bezogen auf ein Projektmanagement jedoch eher Spitzfindigkeiten. Der Sinn einer Aufteilung in .h und .cpp Dateien wird damit verfehlt, denn sie sollen die Übersichtlichkeit erhöhen. Durch die Aufteilung in 3 Header wird jedoch genau das Gegenteil erreicht. Letztlich leistet der Header vom Typ .h1 nicht mehr als eine Vorwärtsdeklaration einer Klasse. Will man eine KlasseA jedoch derart deklarieren, kann man class

KlasseA; schreiben. Ein solcher Header erbringt da keinen Vorteil und die komplizierte Verknüpfung derselben führt die Idee ad absurdum.

Eine weitere Auffälligkeit war die Verwendung von Namespaces. Namespaces wurden eingeführt um Kollision im Namensraum vermeiden zu helfen. In lokal begrenzten Projekten hat man damit normalerweise keine echten Probleme, denn der Compiler weist auf derartige Kollisionen hin und um nach Dateien suchen zu können (was bei grösseren Projekten eine sehr häufige Aktion ist) macht es sowieso Sinn verschiedene Klassennamen zu verwenden. Der Einsatz von Namespaces kann dann bedeutsam werden, wenn zwei bislang parallel entwickelte Projekte zusammengeführt werden. Dann kann man die Projekte jeweils in einen eigenen Namensraum legen und Kollisionen verhindern. Die exzessive Schachtelung von Namespaces wie in Teilen des Robotsoccerprojektes scheint jedoch, da ein rein lokales Projekt vorliegt, zumindest übertrieben. Das ganze führt letztlich zu einem Mehraufwand, der sich wohl kaum bezahlt macht.

Weiterhin sollte jede Klasse ein eigenes .h und .cpp Paar besitzen, da man dann Klassen über den Dateinamen leicht finden kann. Im Robotsoccerprojekt finden sich dagegen in manchen Headerdateien gleich mehrere Klassendeklarationen. Teilweise ist die Deklaration einer Klasse umgekehrt auch auf verschieden benannte Header (bei gleicher .h Endung) verteilt.

2.5.1.2 Restrukturierung

Hier sollen Lösungen zu obigen Schwächen in der Strukturierung angeboten werden.

Obiges konventionelles Konzept der Aufteilung des Sourcecodes auf .h und .cpp Dateien wurde von der PG416 als sinnvoll erachtet. Also stellt sich die Frage: wie wandelt man die .h1, .h2 und .h3 Dateien in konventionelle .h Dateien um? Dies ist relativ leicht, allerdings, in Anbetracht der Dateianzahl nicht wenig Arbeit. Betrachtet man die genannten Headerdateien, stellt man fest, dass die .h2 Dateien den konventionellen .h Dateien sehr stark ähneln. Die Ähnlichkeit geht sogar so weit, dass bis auf die Endung und die seltsamen Includes von .h1 Dateien gar kein Unterschied besteht! Die überraschend einfache Lösung besteht also in einer Umbenennung der .h2 Dateien in .h Dateien. Die .h1 und .h3 Dateien fallen einfach weg. Wer sie einzieht muss jetzt .h einziehen. Da recht viele Dateien allerdings nach diesem Konzept aufgebaut sind und viele Abhängigkeiten bestehen, wird man recht lange benötigen, bis man wieder ein compilierfähiges System hat.

Das Problem überfrachteter Namespaces lässt sich genauso leicht lösen, also: löschen!

Die weitere Anpassung an die konventionelle Form ist auch nicht weiter schwer: man gebe jeder Klasse

ein eigenes .cpp und .h Paar, wie oben angedeutet und muss dann nur noch die Includes anpassen.

2.5.2 Compiler Warnungen, Debugging

2.5.2.1 Einleitung

Das Robotsoccersystem ist ein sehr komplexes in C/C++ geschriebenes Programm. Vorteile dieser Programmiersprache sind vor allem ihr hoher Verbreitungsgrad und die Laufzeiteffizienz der damit erstellten Programme.

Ein schwerwiegendes Problem bei dieser Sprache ist, dass die geringen Zwänge, die sie dem Programmierer auferlegt, eine Vielzahl von z.T. sehr schwer auffindbaren Fehlern ermöglicht. Bei einem Projekt dieser Größenordnung (ca. 2 MB Code) ist damit die manuelle Fehlersuche (z.B. mittels printf) einigermassen hoffnungslos. Das Hauptproblem ist die Lokalisierung des Fehlers. Wer schon einmal versucht hat, in einem größeren Programm einen Fehler zu lokalisieren, der sich später als uninitialisierter Zeiger herausstellte, weiß, dass diese Lokalisierung extrem schwer ist, da das Verhalten des Programms teilweise nichtdeterministisch erscheint.

Darum ist es ein Muss, Gebrauch von Werkzeugen zu machen, die speziell für diesen Zweck geschaffen sind. Da die Fehlersuche (das Debuggen) nicht zu den angenehmsten Aufgaben gehört, sind diese Werkzeuge nicht nur die praktisch einzige Chance (bei sehr großen Programmen), sondern auch eine willkommene Arbeitserleichterung.

2.5.2.2 Ausgangslage für die PG416

Die PG416 fand ein Programm vor, welches einige Laufzeitprobleme aufwies. Es gab immer wieder Abstürze aus heiterem Himmel, d.h. es war schwer ein bestimmter Zusammenhang erkennbar und es gab von der Programmseite her auch keinerlei nützliche Hinweise, die einem helfen könnten einen Fehler einzugrenzen. Solche “Hinweise” beschränkten sich dann auf “Segmentation fault” oder ähnliches. Weiterhin schien das Programm Speicherleaks zu haben, da mit der Laufzeit der Speicherbedarf kontinuierlich zunahm.

Auf Anfrage, was gegen diese Probleme bisher unternommen wurde, wurde klar, dass von oben genannten Werkzeugen noch wenig Gebrauch gemacht worden war.

Technisch erschwert war (und ist) das Debuggen durch die Existenz mehrerer Threads (Bildverarbeitung, Kamera und Strategie), wodurch viele Debuggingtools an Wert verlieren, da sie oft nicht threadsicher sind.

2.5.2.3 Folgerung

Aus dem Gesagten lässt sich leicht ableiten, dass die Nutzung von Softwarewerkzeugen zur Qualitätssicherung dringend nachgeholt werden musste.

Jedes Werkzeug, dass einem bei der Fehlervermeidung und -beseitigung weiterbringt, sollte genutzt werden. Man kann sich dies wie ein Filtersystem vorstellen, was die Aufgabe hat Fehler zu entfernen.

2.5.2.4 Durchgeführtes Debugging

Compiler nutzen

Einleitung Moderne C/C++ Compiler verfügen über die Möglichkeit sogenannte Warnings auszugeben. Diese sind generell als Hinweise auf Fehler(quellen) oder zumindest schlechten Programmierstil anzusehen. Der hier eingesetzte Compiler ist der Gcc. Die PG416 musste feststellen, dass von dessen Warnings bisher keinerlei Gebrauch gemacht wurde.

Es soll hier ausdrücklich betont werden, dass die Warnings des Compilers absolut ernst zu nehmen sind, d.h. man sollte den Code stets derart abwandeln, dass keine Warnings mehr auftauchen!

Im Falle des Gcc-Compilers gilt es als empfehlenswert vom Flag -Wall Gebrauch zu machen. Auf diese Art werden viele der häufigsten Programmierfehler erkannt.

Jener Tag, an dem einige Mitglieder der PG416 nun alle Warnings des Gccs aktivierten, sollte unterhaltsam werden. Der Compiliervorgang wurde wie immer gestartet, mit dem kleinen Zusatz des -Wall Flags. Nun erwies sich der Compiler als unerwartet redselig. Über viele Bildschirmseiten wies er seine geduldigen Nutzer auf verbesserungswürdige Stellen im Programm hin. Schnell fiel es den Nutzern schwer ihm noch zu folgen, denn allzu stürmisch prasselten seine Anmerkungen auf sie hinunter. In bewährter Manier wurde der Vorgang deshalb wiederholt um all die gutgemeinten Ratschläge in eine besser lesbare Datei umzuleiten. Dann hieß es warten...

Schließlich war das Dokument erstellt und rief ob seiner Größe Verwunderung bei seinen Lesern hervor: sein Umfang betrug 34,6 MB! Beachtlich, wenn man bedenkt, dass die zu kommentierende Quelle 17 Mal kleiner war.

Glücklicherweise waren viele der Warnings Resultate anderer Warnings.

Solche Fehler oder stilistisch schlechten Programmteile sind, wenn man die Bedeutung der Warnings kennt, leicht zu verbessern.

Deshalb folgt hier eine Auflistung der wichtigsten Warnings und der Angabe von Beseitigungsmöglichkeiten. Die meisten derartigen Probleme kann man damit praktisch sofort lösen.

Gcc Warnings und deren Beseitigung

1. unused variable

   Dies ist ein Warning der harmlosen Art. Eine Variable wurde deklariert, wird aber nicht benutzt. Man kann sich zwischen Löschen und Auskommentieren entscheiden.

   Die konservative Methode des Auskommentierens bietet sich an, wenn man fremden Code bearbeitet, denn manche Variable mag ihren Sinn vielleicht noch in der Zukunft erlangen.

2. comparison between signed and unsigned

   Zwei Variablen, von denen nur eine vorzeichenbehaftet ist, werden miteinander verglichen. Dies ist eine potentielle Fehlerquelle, denn wenn z.B. die vorzeichenbehaftete Zahl negative Werte annehmen kann, kann ein schwer auffindbarer Bug die Folge sein. Gleiches gilt insbesondere für sehr große positive Werte der unsigned-Zahl, denn ihr positiver Wertebereich ist größer als der der signed-Zahl.

   Am besten ist es deartige Vergleiche zu vermeiden, indem man z.B. nur signed-Zahlen verwendet. Ist dies nicht leicht möglich, kann man zumindest den Compiler mit einem Cast zufriedenstellen, also z.B. so:

   int i; unsigned int j;

   ...

   if (i > (int)j) {...}

   Dies sollte man so machen, wenn man sicherstellen kann, dass j nie größer als INT_ MAX ist. Eine andere Methode ist der Cast zum nächstgrößeren Datentyp, was aber beim größten nicht mehr geht, also so:

   if ((long long)i > (long long)j) {...}

3. char format, different type arg (arg #)

   Der Gcc überprüft Ein- und Ausgabefunktionen wie printf(), denen man Formatlisten übergeben kann, darauf, ob der Formatstring zu den übergebenen Parametern passt. Z.B. passt i.f. Fall der float-Parameter nicht direkt zum Formatstring, denn %d steht für die Ausgabe von Integerwerten.

float f;

printf (" % d " , f);

Man sollte dies zunächst durch den passenden Formatstring beheben (das wäre hier %f) oder, falls tatsächlich die Ausgabe in einem anderen Typ gewünscht ist, entsprechend casten.

4. In function 'func': warning: 'i' might be used uninitialized in this function

   Dieses Warning zählt zu denjenigen, die auf besonders unangenehme Fehler hinweisen können. Deshalb sehe man sich den entsprechenden Code recht genau an.

   Der Compiler will damit sagen, dass i in der angegebenen Funktion innerhalb eines bestimmten Programmablaufs benutzt werden könnte, ohne vorher initialisiert worden zu sein. Dies kann, besonders im Falle von Zeigern zu sehr schwer auffindbaren Fehlern führen.

   Meist wird die korrekte Initialisierung leicht ersichtlich sein und mehr oder weniger vergessen worden sein.

   Ist dies nicht der Fall oder erweist sich die Initialisierung als keineswegs leicht, so sollte man zumindest einen gezielten Programmabbruch sicherstellen, der dann zur Laufzeit eine leichte Lokalisierung der fehlerhaften Programmstelle ermöglicht.

   Man kann dies z.B. erreichen, indem man initial einen Wert zuweist, mit dem später, falls die eigentlich korrekte Initialisierung nicht gefolgt ist, nicht ohne Programmabsturz weitergearbeitet werden kann.

   Bei Zeigern könnte man zu diesem Zweck NULL zuweisen. Sollte es sich dabei um einen Zeiger auf ein Objekt handeln und wird später z.B. eine Methode des Objekts aufgerufen, dann wird das Programm genau an der Stelle des Aufrufs abstürzen, sofern es sich immer noch um einen Nullzeiger handelt. Das erleichtert dann die Lokalisierung.

   Besser ist natürlich eine Überprüfung direkt vor dem Aufruf darauf, ob es sich um einen Nullzeiger handelt und falls ja Programmabbruch mit entsprechend kommentierender Ausgabe. Das bedeutet aber oft einen wesentlichen höheren Aufwand...

5. warning: control reaches end of non-void function

   Dies ist auch ein häufiger und schwerer Fehler. Eine Funktion, die einen Wert gemäß Deklaration liefern soll, liefert in einem bestimmten Zusammenhang nichts.

   Da diese Funktion aufrufende Programmteile evtl. mit diesem Wert weiterarbeiten möchten, ist es möglich, dass sie völlig falsche Werte erhalten.

Meist ist dies leicht dadurch zu beheben, dass man ein ”return” hinzufügt, das den richtigen Wert liefert.

Ansonsten kann man sich hier ähnlicher Techniken wie im vorherigen Abschnitt bedienen. Man kann diese Fehlerart auf die vorherige reduzieren, indem man für alle return-Fälle die gleiche Variable verwendet, was vielleicht auch kein schlechter Programmierstil ist.

6. class X has virtual functions but non-virtual destructor

   Eine Klasse, die virtuelle Funktionen besitzt hat keinen Destruktor, der mit virtual deklariert ist.

   Dies ist einfach zu beheben, indem man einen (möglicherweise leeren) Destruktor schreibt und virtual davor setzt, bzw., falls existent nur virtual davorschreibt.

   Dies mag nicht unmittelbar einsichtig sein, darum eine kleine Erklärung. Funktionen die mittels virtual deklariert sind, sind grundsätzlich Kandidaten dafür, von Erweiterungen überschrieben zu werden. Damit nun die Destruktoren der Erweiterungen und der Basisklasse aufgerufen werden, muss der Destruktor der Basisklasse mit virtual deklariert werden. Ansonsten sind Speicherleaks recht wahrscheinlich.

Erfahrungen mit genutzten Debuggingtools Hier folgt eine kurze Beschreibung der genutzten Debuggingtools. Es soll hier mehr darauf eingegangen werden, was mit ihnen jeweils im praktischen Einsatz erreicht werden konnte. Für eine Anleitung, wie das jeweilige Tool auf das Robotsoccerprojekt anzuwenden ist, siehe man im Anhang unter Laufzeittools.

1. Mpatrol

   Mpatrol ist ein besonders mächtiges (hinsichtlich der Vielfalt der Optionen) Tool, welches vor allem zum Aufspüren von Fehlern bei dynamisch erzeugtem Speicher gedacht ist.

   Mpatrol hat keine Probleme mit Threads, was eine wichtige Voraussetzung war. Die Anwendung von Mpatrol führte dann recht schnell einige typische C-Fehler zu Tage, z.B. solche, die bei der Arbeit mit nullterminierten Characterstrings leicht auftreten. Mpatrol überschreibt C-Stringfunktionen wie strcpy() durch sichere Varianten, die dann in der Lage sind bestimmte Fehler abzufangen. Ein häufiger Fehler war dann genannte Strings nichtnullterminiert an strcpy() zu übergeben, was die zufällige Überschreibung von Speicherbereichen zur Folge hat. Grundsätzlich wird empfohlen, da C-Strings sich als Hauptfehlerquelle erwiesen haben, unter C++ Stringobjekte zu verwenden.

Mpatrol erwies sich als sehr gut konfigurierbares Tool, welches sinnvoll auf Robotsoccer angewendet werden kann. Das einzige Problem war dessen starke Verlangsamung des Programmablaufs. Speziell bis zum Aufbau der GUI konnte man kaum warten. Die Konsequenz ist, das dadurch die realistische Benutzung von Robotsoccer (sprich: ein Spiel durchführen) und die gleichzeitige Nutzung von Mpatrol nicht möglich war. Deshalb konnten so nicht alle Situationen geprüft werden.

2. ccmalloc

   Ccmalloc ist kein so mächtiges Tool wie Mpatrol, aber ist ist einfach in der Benutzung und seine Ausgaben sind gut lesbar, denn Fehler werden zeilengenau angegeben. Das Hauptproblem bei ccmalloc bis hierhin war die unzureichend Unterstützung von Threads. Es gab zwar Patches, die das Problem beheben sollten, aber bis heute (Februar 2003) ist das noch nicht vollständig geschehen.

   Deshalb konnten nur Fehler in der Startphase des Programmes entdeckt werden, in denen die Threads noch nicht aktiv waren.

3. LeakTracer

   Auch LeakTracer ist einfach in der Benutzung und wird im Zusammenhang mit größeren Projekten (z.B. KDE oder Mozilla) eingesetzt. Die Erfahrungen beim Robotsoccerprojekt waren jedoch vergleichbar zu denen von ccmalloc, d.h. LeakTracer war keine große zusätzliche Hilfe.

4. valgrind

   Valgrind ist ein relativ neues und sehr vielversprechendes Tool, das mehrere andere Debugtools in sich vereint. Es ist besonders einfach in der Anwendung, denn man kann ihm direkt Binaries übergeben ohne neu compilieren zu müssen.

   Valgrind entdeckt Bereichsüberschreitungen bei Arrays sofort und hält das Programm dann genau an der betreffenden Stelle an. Im Falle von Speicherleaks kann es diese nicht immer zeilengenau zuordnen, es kann aber zumindest recht präzise angeben, ob Leaks vorliegen oder nicht.

   Auch Threads stellen kein Problem für valgrind dar. Das einzige und leider schwerwiegende Problem ist, dass es bis dahin noch nicht mit NVidia's OpenGL - Treibern arbeiten konnte.

   Deshalb konnten damit Bugs nur außerhalb der GUI entdeckt werden. Da dieses Tool jedoch im Vordergrund der OpenSource-Entwicklung zu stehen scheint, sollte man es unbedingt im Auge behalten.

Debuggingtechniken An erster Stelle sollte hier die Benutzung von oben genannten Debuggingtools genannt werden. Mit ihnen entdeckt man Fehler, die man sonst vielleicht niemals bemerken würde.

Darüberhinaus war es aber auch nützlich vom CVS-System Gebrauch zu machen. Nachdem das Flag -Wall zahlreiche Warnings hervorrief, musste man sich überlegen, wie bei einer so großen Anzahl zu beseitigender Fehler vorzugehen ist.

Dabei erwies sich folgende Methode als geeignet: man erstellt zunächst bevor man irgendeine Änderung vornimmt einen Schnappschuß des aktuellen Standes im CVS-System. Das führt dazu, dass man unter einem bestimmten Namen immer wieder den derzeitigen Stand abrufen kann.

Nun nimmt man alle nötigen Veränderungen vor, beseitigt alle Fehler. Da man hierbei dazu geneigt ist, ein paar neue Fehler zu produzieren, markiere man jede Veränderung durch einen kurzen einheitlichen Kommentar im Quellcode um diese Stellen später leicht finden zu können.

Sind auf diese Weise alle Fehler entfernt, fügt man den Stand wieder in das CVS-System als aktuellen Stand ein und erzeuge wiederum einen Schnappschuß. Aus seinem Namen sollte hervorgehen, dass es der Stand nach der größeren Veränderung war.

Treten nun in der Benutzung der neuen Version bestimmte Fehler auf, die in der alten nicht vorhanden war, hat man die Möglichkeit mit der alten zu vergleichen, was sehr vorteilhaft bei der Fehlerlokalisierung sein kann.

Eine Technik (die nicht immer anwendbar ist), die hier konkret zum Einsatz kam war die folgende: aufgrund zahlreicher Veränderungen brachte die neue Version auch einen neuen Fehler mit sich. Nun wurden .cpp Dateien der neuen Version systematisch in Manier der binären Suche mit dem Gegenstücken der alten Version ausgetauscht. Sobald das Programm dann den Fehler nicht mehr aufweist ist klar, dass man soeben den Teil ausgetauscht hat, der den Fehler enthält. Nun beschränkt sich die Suche auf diesen Teil und man fährt rekursiv fort indem man davon wieder die Hälfte austauscht. So konnte dann in logarithmischer (Compilier- !) Zeit der neue Fehler lokalisiert werden.

Kapitel 3

Hardware

3.1 Roboter

Die eingesetzten Roboter basieren auf dem Entwurf von Thomas Klute [Klu01]. Die Roboter der zweite Baureihe unterscheiden sich in zwei Punkten von denen, der ersten: Zum einen stieg die Anzahl der Impluse pro Motorumdrehung von 16 auf 512, zum anderen sind die Roboter aufklappbar (s. Abb. 3.1, Seite 82).

3.1.1 Aufbau

Die Software besteht aus mehreren Modulen (s. Abb. 3.2, Seite 83), die man in 5 Bereiche einteilen kann, wobei bestimmte Module zu mehr als einem Bereich gehören und deshalb in der folgenden Übersicht doppelt auftauchen. Die einzelnen Module werden im folgenden Kapiteln erläutert.

1. Ablaufsteuerung
    - Main
    - Scheduler
2. Steuerung des Roboters
    - Control
    - Winkel
    - Rampe

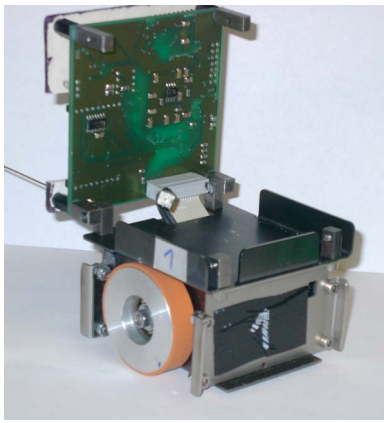


Abbildung 3.1: Roboter der zweiten Baureihe.

- Speed
- Pid

3. Funk
   - Uart
   - Dfrsm
4. Interrupt Verarbeitung
   - Rtint
   - Uart
5. Ansteuerung und Abfrage der Hardware
   - Quadinp
   - Pwmmot
   - Led
   - Adc
   - digiin

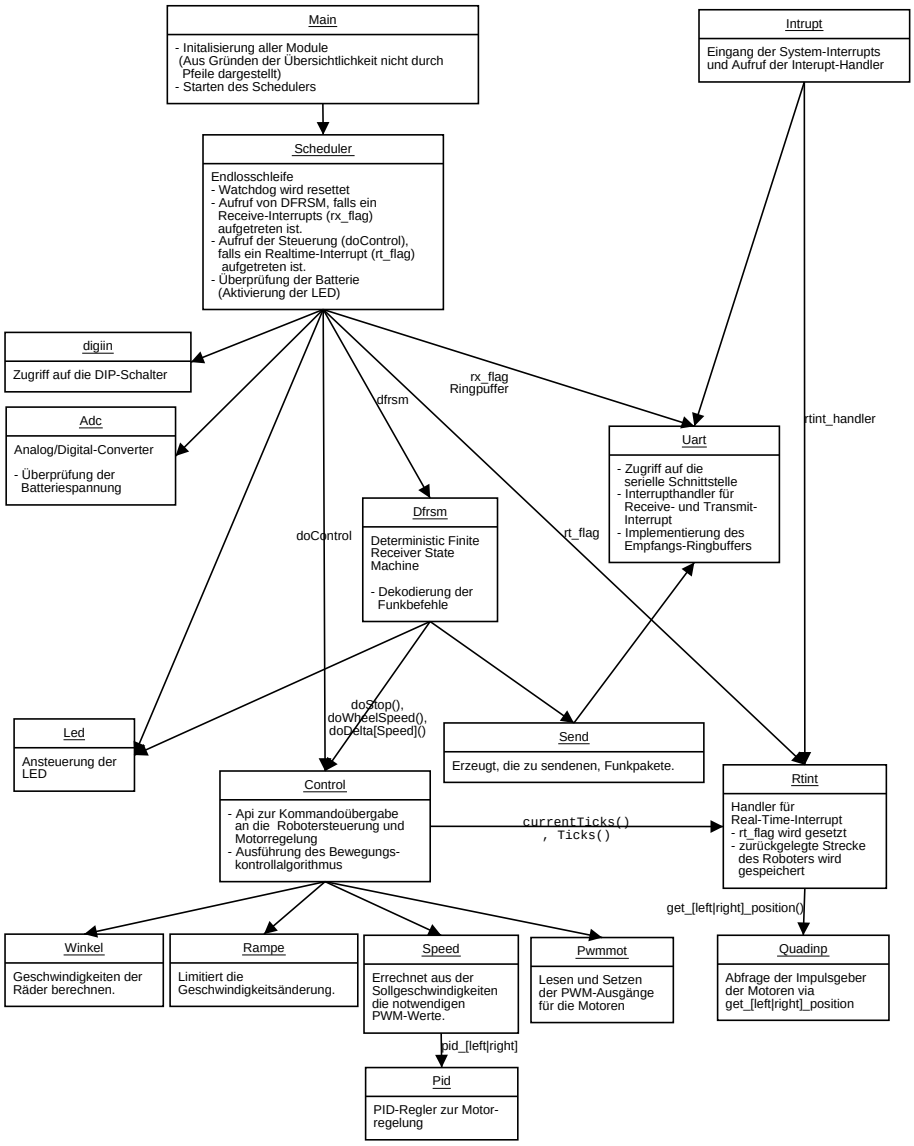


Abbildung 3.2: Struktur der Software auf dem Roboter.

3.1.2 Ablaufsteuerung

3.1.2.1 Main

Das Modul Main sorgt dafür, dass die Initialisierungsroutinen aller Module aufgerufen werden und dass der Scheduler gestartet wird.

3.1.2.2 Scheduler

Der Scheduler besteht aus einer Endlosschleife, die folgende Aufgaben durchführt:

- Falls das Flag rx_flag (s. 3.1.5.3) gesetzt ist:
  Aufruf des Moduls Dfrsm.
- Falls das Flag rt_flag (s. 3.1.5.2) gesetzt ist:
  Aufruf der Funktion doControl aus dem Modul Control.
- Falls keiner der beiden oberen Punkte zutrifft:
  Ladezustand der Batterie überprüfen.

3.1.3 Steuerung / Regelung

3.1.3.1 Control

Die Funktion doControl wird 108 mal in der Sekunde von dem Modul Scheduler aufgerufen und ist das Kernstück der Steuerung des Roboters. Aus dieser Funktion werden sämtliche anderen Module dieses Bereiches aufgerufen. Die Abbildung 3.3 auf Seite 85 verdeutlicht, wie die momentan vorhandenen Module Winkel (Winkelsteuerung), Rampe (Beschränkung von Geschwindigkeitsänderungen) und Speed (Geschwindigkeitsregelung) und die zukünftig zu entwickelnden Module Pathfinder (Wegberechnung) und Zielansteuerung (Zielpunkt anfahren) ineinander greifen. Desweiteren wird verdeutlicht, an welchen Stellen die per Funk übermittelten Befehle ansetzen.

3.1.3.2 Winkel

Dieses Modul basiert auf den Ideen der PG 362 [PG362, S. 18].

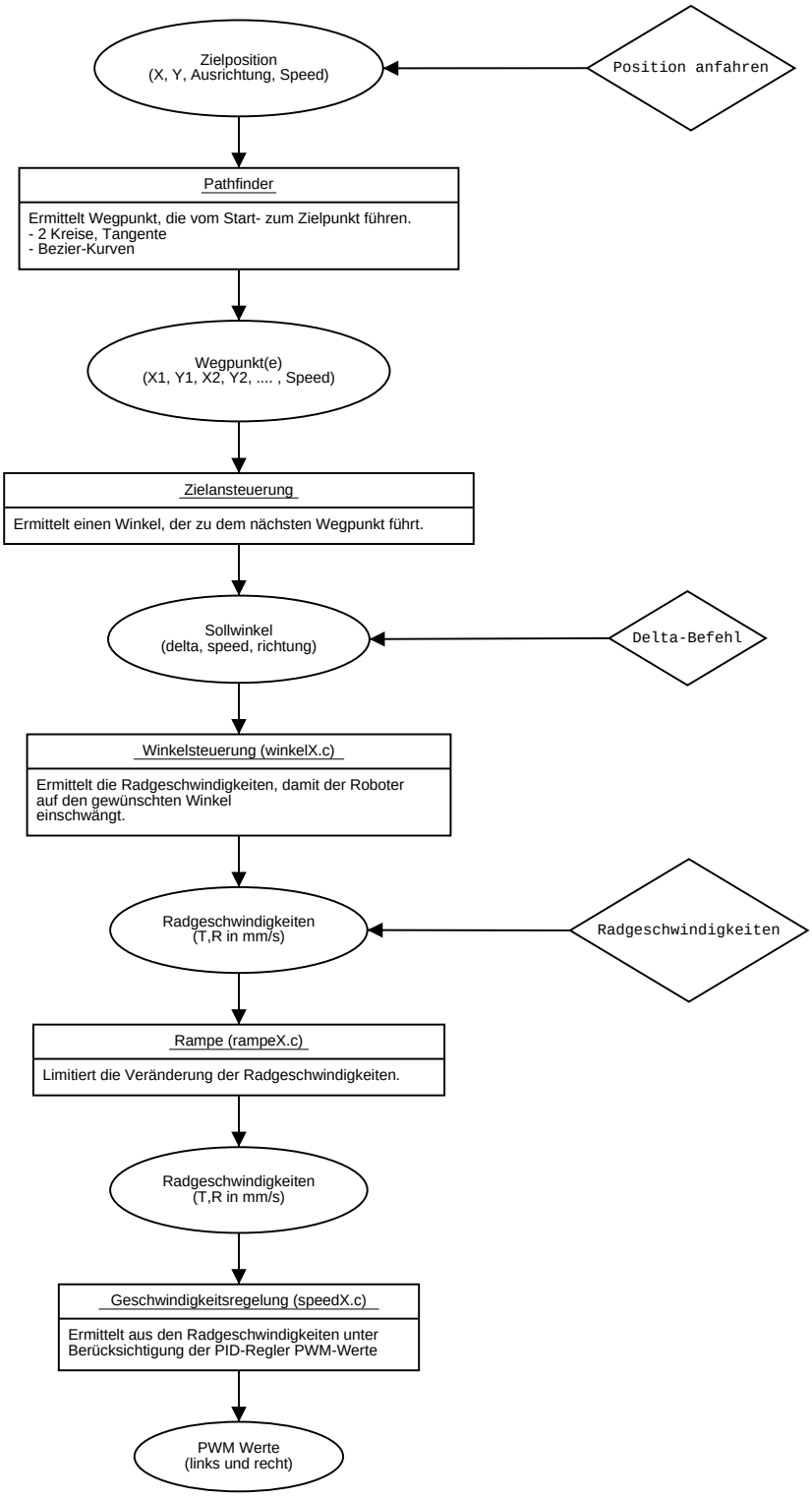


Abbildung 3.3: Abhängigkeiten im Bereich Steuerung

Durch den Delta-Befehl ist ein Winkel gegeben, der die Richtungsänderung während der Fahrt vorgibt. In diesem Modul werden nun TR-Werte berechnet, die dafür sorgen, dass der Roboter die gewünschte Richtungsänderung ausführt.

Die von der PG 362 gewählte Implementierung war ungünstig, wie ein kleines Beispiel verdeutlichen soll.

Ausgangssituation: Ein Roboter fährt mit 1000 mm/s und bekommt den Befehl, sich um 90° nach links zu drehen. Turn_Max ist auf 200 eingestellt.

Die alte Funktion berechnete in diesem Fall für das linke Rad eine Geschwindigkeit von 200 mm/s und für das rechte Rad eine Geschwindigkeit von -200 mm/s. Dies würde einer Drehung auf der Stelle entsprechen, wenn der Roboter nicht bereits in Bewegung wäre.

Dieser Ansatz hat zwei gravierende Nachteile:

- verlangsamte Fahrt
  Auch wenn der Roboter mit 1000 mm/s fahren soll, so verringert sich bei jeder Kurve die Geschwindigkeit.
- Traktionsverlust
  Diese abrupte Geschwindigkeitsänderung kann zum Verlust der Traktion führen, da die Rampen für solche extremen Manöver nicht ausgelegt sind.

Aus diesem Grund wurde eine einfachere Variante implementiert:

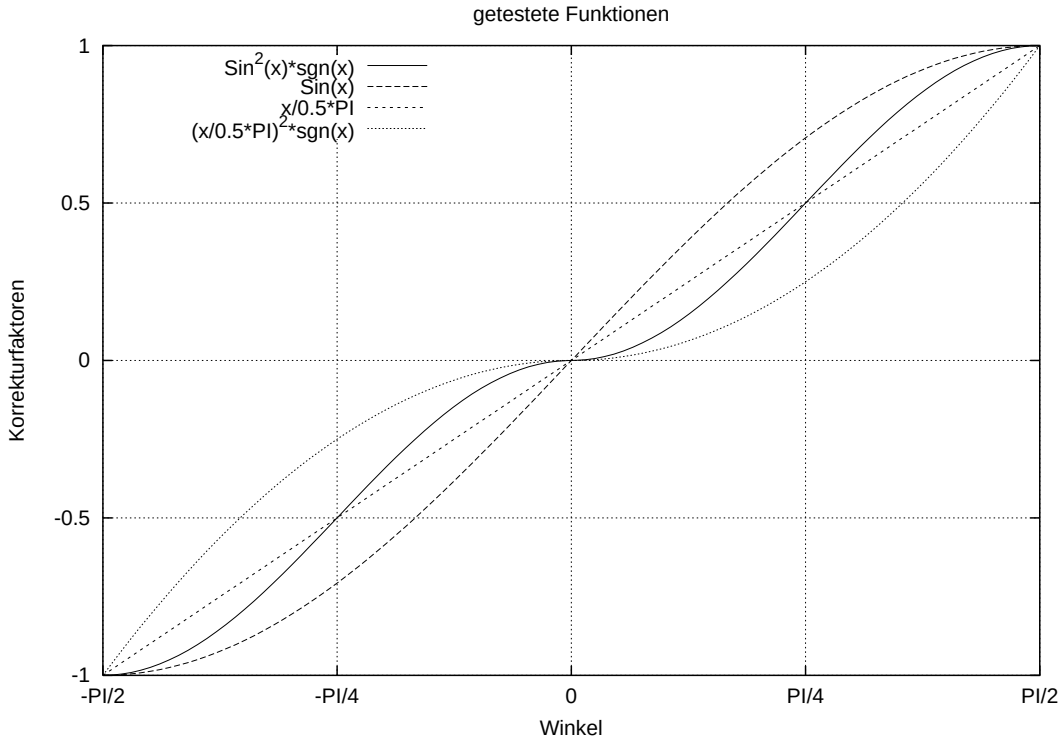
Mit Hilfe der Funktion $f(\alpha)$, die den Wertebereich $[-1:1]$ hat, wird ein Korrekturwert für die Räder ermittelt.

Das linke Rad wird mit einer Geschwindigkeit von $\mathrm{t} + f(\alpha) * \mathrm{TurnMax}$ und das rechte mit $\mathrm{t} - f(\alpha) * \mathrm{TurnMax}$ angesteuert.

An die Funktion $f(\alpha)$ werden zwei wesentliche Voraussetzungen gestellt: Die Steigung darf zum einen nicht so groß sein, dass der Roboter permanent übersteuert, zum anderen muss sie jedoch groß genug sein, dass der Roboter schnelle Drehungen ausführen kann.

Die PG 362 entschied sich in ihrer Implementation für $f(\alpha) = sin^2(\alpha) * sgn(\alpha)$. Tests zeigten jedoch, dass diese Funktion für die Neuimplementierung suboptimal war, weshalb weitere Funktion getestet wurden.

Die nachfolgende Grafik zeigt einige der getesteten Funktionen:



Tests

- Der Roboter bekommt die Anweisung, sich während der Fahrt um 15°, 30°, 45°, 60°, 75° und 90° zu drehen.

  Mit Hilfe dieser Testreihe kann überprüft werden, ob der Roboter sich schnell genug drehen kann und ob er bei den Drehungen nicht überschwingt.

- Der Roboter bekommt in kurzen Abständen Anweisungen, sich abwechselt um 5° nach links und rechts zu drehen.

  Mit diesem Test wird die Ungenauigkeit der Positionserkennung der Roboter simuliert, die den Roboter zu solchen Kurskorrekturen veranlasst.

Die durchgeführten Tests zeigen, dass keine der getesteten Funktionen die Erwartungen erfüllt und hier weitere Versuche notwendig sind.

3.1.3.3 Rampe

Auf die Rampe der PG 362 [PG362, S. 27ff] wurde zu Gunsten der von Norman Weiss implementierten Rampe verzichtet. Diese Rampe hatte den Vorteil, dass sie bei gleicher Güte, sowohl einfacher zu ver-

stehen als auch zu warten war. Da alle bisher implementierten Rampen nicht die physikalischen Kräfte des Roboters berücksichtigten und deshalb in verschiedenen Situationen nicht optimal funktionieren, wurde ein physikalisches Robotermodell entworfen (s. 5.1), auf dessen Grundlage eine neue Rampe implementiert werden kann. Die abschließende Implementierung wurde noch nicht durchgeführt.

3.1.3.4 Speed, PID

Das Modul Speed errechnet aus der gewünschten Geschwindigkeit unter Benutzung der PID-Regler neue PWM-Werte für die beiden Räder.

3.1.3.5 simulateMovement

Dieses Modul approximiert unter Berücksichtigung der Radencoder den vom Roboter gefahrenen Weg. Das Verfahren basiert auf der Implementierung der PG 362 [PG362, S. 33ff], wobei bei der damaligen Implementation fälschlicherweise keine Fallunterscheidung zwischen einer Links- und Rechtskurve gemacht wurde, sondern die Bewegung immer so berechnet wurde, als ob es eine Rechtskurve wäre. Dieser Fehler wurde korrigiert.

3.1.4 Ansteuerung und Abfrage der Hardware

3.1.4.1 Quadinp

Mit Hilfe dieses Moduls ist der Zugriff auf die Radencoder [Fau00] möglich, die in jedem der beiden Motoren eingebaut sind und mit deren Hilfe die Anzahl der Radumdrehungen ermittelt werden kann. Durch den Test der Radencoder (siehe Seite 93) wurde ein weiterer Fehler gefunden. Nach einer gewissen Zeit gibt es in den Zählern für die Radumdrehungen einen Überlauf, der nicht abgefangen wurde. Um diesen Überlauf zu vermeiden, setzen die Routinen nun nach dem Auslesen der Zähler, diese wieder auf 0 zurück. Da die Zähler in Abständen von 10 ms ausgelesen werden, ist ein Überlauf nun nicht mehr möglich.

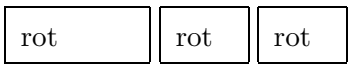
3.1.4.2 Pwmmot

Mit Hilfe dieses Moduls können die PWM-Ausgänge, die die Ansteuerung der beiden Motoren kontrollieren, gesetzt und ausgelesen werden.

3.1.4.3 Led

Dieses Modul dient zur Ansteuerung der Led, die auf der Hauptplatine sitzt. Leider kam es in der Vergangenheit des öfteren vor, dass Unklarheit über die Interpretation der Lichtsignale herrschte. Aus diesem Grund wurden die folgenden neuen Lichtsignale eingeführt:

Ein gültiger Funkbefehl wurde empfangen.



Ein Fehler ist aufgetreten. Die Anzahl der kurzen Impulse gibt den Fehlercode an.

Fehlercode	Beschreibung
1	Funk-Timeout In den letzten zwei Sekunden wurde kein gültiges Funkpaket empfangen, weshalb der Roboter gestoppt wurde.
x	Auf eine Auflistung der weiteren Fehlercodes wurde verzichtet, da diese nur während der Entwicklung benutzt werden und sich somit relativ häufig ändern.

Batteriewarnung.

3.1.4.4 Adc

Zugriff auf die Analog/Digital-Konverter. Dieser Konverter wird z.Z. nur zur Ermittlung der Batteriespannung benutzt.

Die Schwelle, ab der die Batteriewarnung aktiviert werden soll, konnten wir nicht ermitteln, da uns kein Netzteil mit geregelter Spannungsversorgung zur Verfügung stand.

3.1.4.5 digiin

Zugriff auf die DIP-Schalter, die zur Einstellung der RoboterID dienen. Die alte Implementation sah in Abhängigkeit vom Spielmodus (3 gegen 3 oder 5 gegen 5) verschiedene Stellungen der DIP-Schalter für die gleiche ID vor. Dieses wurde vereinheitlicht:

ID	DIP 2	DIP 3	DIP 4
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	1	1	1

Dieses Verfahren zur Einstellung der RoboterID ist nur für bis zu 6 Roboter (ID 0 ist für das Hostsystem reserviert) praktikabel und sollte durch ein automatisiertes Verfahren ersetzt werden (s. [Wei03]).

3.1.5 Interrupt Verarbeitung

3.1.5.1 Intrupt

Dieses Modul stellt einen globalen Interrupt-Handler zur Verfügung, der in Abhängigkeit vom eingegangenen Interrupt Funktionen aus den Modulen Rtint und Uart aufruft.

3.1.5.2 Rtint

Dieses Modul stellt die Funktion rtint_handler zur Verfügung, die aufgerufen wird, wenn ein Realtime-Interrupt vom DSP generiert wurde. Dieser Realtime-Interrupt wird in der aktuellen Implementierung 108-mal in der Sekunde generiert. Es werden folgende Funktionen ausgeführt:

- Momentangeschwindigkeit bestimmen

  Durch Auslesen der Radencoder wird auf die Momentangeschwindigkeit geschlossen.

- rt_counter inkrementieren

  rt_counter wird von verschiedenen Modulen für die Zeitmessung genutzt.

- rt_flag setzen

  Das Setzen von rt_flag führt dazu, dass die Routine do_control vom Scheduler aufgerufen wird.

3.1.5.3 Uart

Je nachdem, ob sich der Roboter im Empfangs- oder Sendemodus befindet, werden die Funktionen uart1_rxint oder uart1_txint vom globalen Interrupt-Handler aufgerufen.

uart1_rxint Diese Funktion hat die Aufgabe, das empfangene Zeichen in den Empfangsbuffer (ringbuffer) zu schreiben und die Variable rx_int zu setzen.

uart1_txint Schreibt das nächste Zeichen aus dem Sendebuffer (uart1_sbuffer) in das Senderegister.

3.1.6 Funk

Anders als bei der Implementierung auf der Hostseite, gibt es auf der Roboterseite keine Schwierigkeiten mit dem Emfang und dem Senden von Daten.

Sowohl der Empfang als auch das Senden sind komplett interruptgesteuert, so dass nur sichergestellt werden muss, dass zum einen die Daten schnell genug in das Senderegister geschrieben werden (uart1_txint) und zum anderen die Daten schnell genug aus dem Emfangsregister gelesen werden (uart1_rxint).

Weiterführende Informationen zur Programmierung der Uart sind im Datenblatt [Tex01, Kap 8-2] zu finden.

3.1.6.1 Uart

Dieses Modul stellt neben den in Kapitel 3.1.5.3 erwähnten Funktionen noch die Funktion uart1_put_string zur Verfügung, die den übergebenen String in den Sendebuffer schreibt und danach in den Sendemodus umschaltet.

3.1.6.2 Dfrsm

Dieses Modul stellt einen endlichen Automaten zur Verfügung, mit dessen Hilfe die eingehenden Funkpakete (s. 2.3.1) verarbeitet werden.

Der endliche Automat wartet auf das Startbyte (0x8E) und legt danach die CRC sowie die RoboterID in einem Zwischenspeicher ab. In Abhängigkeit von dem empfangenen Befehl verzweigt der endliche

Automat in verschiedene Zustände, die für die Speicherung der befehlspezifischen Daten sorgen. Nach dem kompletten Empfang des Datenpakets wird die errechnete CRC mit der übermittelten CRC verglichen. Bei einer fehlerlosen Übertragung wird die zu dem Befehl passende Funktion (doStop, doDeltaSpeed, doWheelSpeed,...) aus dem Modul Control aufgerufen, um die neuen Daten an die Steuerung zu übergeben.

3.1.7 Durchgeführte Tests

3.1.7.1 Bestimmung der Reibungskoeffizienten

Testumgebung verwendete Hilfsmittel:

- Federwaagen
  - Messbereich: 0 - 300 p
    Skala: 5 p / Strich
  - Messbereich: 0 - 1 kp
    Skala: 10 p / Strich

Ein Pond sind $9,80665 * 10$ Newton.

Zur Dürchführung der Messsung wird eine Hälfte des Spielfelds gesaugt, die andere Hälfte ungereinigt belassen. Die Räder der Roboter werden mit Hilfe von Klebeband gereinigt (Klebeband falschrum auf einen Tisch kleben und mit den Roboterreifen über die klebende Seite fahren). Der Roboter wird mit der Wettkampfausstattung (Akku, Funkmodul und Patch) versehen. Anschließend werden mit Klebeband die Räder fixiert. Möglichst nah in Höhe des Schwerpunktes des Roboters werden zwei Drahtschlaufen parallel zur Laufrichtung und orthogonal zur Laufrichtung befestigt.

Die Masse der Roboter in dieser Konfiguration:

- Roboter der alten Baureihe: ca. 595g
- Roboter der neuen Baureihe: ca. 545g
- Korea-Roboter: ca. 555g

Durchführung Der Roboter wird innerhalb des gesaugten Bereichs des Spielfeldes platziert. Mit den Federwaagen in der Vorwärtsschlaufe wird nun gemessen, wieviel Kraft benötigt wird, um den Roboter ins Gleiten zu versetzen. Dieses wird mit der orthogonalen Schlaufe wiederholt, mit dem Unterschied, dass der Roboter unter Umständen auch mit einem Rad vom Boden abheben könnte. In diesem Fall wird die Kraft zum Abheben gemessen.

Das ganze wird anschließend auf der ungereinigten Seite des Spielfelds wiederholt. Dazu wird der Roboter auch ein paarmal mit den Rädern durch den Staub gezogen. Es sollten vor dieser Wiederholung des Tests auf der staubigen Seite Räder und Spielfeld mit Staubpartikeln verschmutzt sein.

Der Test wird mit jeweils einem Modell der unterschiedlichen Roboterbaureihen und auf allen Spielfeldern durchgeführt.

Messung

- Platte 1: Professionell erstelltes Spielfeld 2,2 x 1,8 m, Oberfläche: Parkettlack
- Platte 2: Eigenbau Spielfeld 2,2 x 1,8 m, Oberfläche: Tafellack

Robotertyp	Richtung	Platte 1	Platte 1	Platte2	Platte 2
		gesaugt	nicht gesaugt	gesaugt	nicht gesaugt
alte Baureihe	vor.	225p	150p	250p	230p
	ort.	300p	210p	350p	350p
neue Baureihe	vor.	1,42N (145p)	1,07N (110p)	225p	190p
	ort.	2,70N (275p)	1,47N (150p)	275p	275p
Korea-Roboter	vor.	205p	85p	205p	135p
	ort.	320p	150p	330p	275p

3.1.7.2 Radencoder

Testumgebung Der Roboter ist auf einer Kombizange aufbockt, mit dem Parallel-Jtag (Debug) an Carstens Notebook (Celeron 600MHz) angeschlossen und auf Debugmodus gestellt. Als Testprogramm dienen die herkömmlichen Robotersourcen in der aktuellen Version (es sind keine Modifikationen für den Test notwendig). Die Werte der Radencoder werden mittels eines im Code-Composer integrierten Watch-Windows ausgegeben (Datei z.B. quadinp.c) Variablen:

- evptr_cap->T2CNT für linker Motor

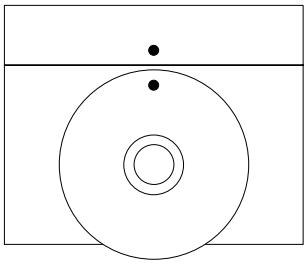


Abbildung 3.4: Ausrichtung des Rades zum Testbeginn

- evptr_cap->T3CNT für rechter Motor

Durchführung Jedes der beiden Räder ist mit einem Punkt versehen. Auch an beiden Seiten des Chassis ist ein Punkt angebracht. Die Punkte werden wie in der Skizze (Abb. 3.4) dargestellt ausgerichtet. Das Programm wird auf den Roboter übertragen und das Watch Window eingestellt. Anschließend wird das Programm gestartet. Vom Programm werden hier die beiden Variablen jetzt auf 0 initialisiert. Man dreht die beiden Räder manuell jeweils eine Vierteldrehung vorwärts bzw. rückwärts (am besten ein Rad vor, das andere zurück). Danach hält man das Programm an und notiert die beiden Werte im Watch-Fenster. Nun wird eine weitere Vierteldrehung weiter gedreht, das Programm kurz weiter laufen gelassen (nicht neu starten!), wieder angehalten und die Werte notiert. Dies kann jetzt jeweils in Viertelschritten immer so weiter gemacht werden. Nach einer vollen Drehung wird hier abgebrochen und danach nur noch die Werte abgenommen, die nach einer weiteren vollständigen Drehung des Rades erreicht wurden.

Die Räder sollten jetzt wieder in "Punktdeckung" stehen. In dieser Position das Roboter-Programm wieder von vorne starten (nicht nur weiter laufen lassen!). Der Test wird jetzt unter Umkehrung der Drehrichtung der Räder wiederholt.

Dieser Test wird mit jedem Roboter durchgeführt: zweimal pro Baureihe in dieser ausführlichen Form, die restlichen Roboter der gleichen Baureihe aber nur noch mit einer Volldrehung und einer zweiten Volldrehung der Räder in beide Richtungen (aus Zeitgründen). Nur wenn sich dort auffällige Abweichungen ergeben, wird der betreffende Roboter auch mit den Viertel-Werten getestet.

zu erwartende Ergebnisse

- für die Roboter der alten Baureihe (Roboter 1-7):

Die Motoren liefern 64 Ticks pro Umdrehung (internes Getriebe mit eingerechnet). Bei einer Umsetzung von 8/1 ergeben sich also $64 * 8 = 512$ Ticks pro Radumdrehung

- für die Roboter der neuen Baureihe (Baujahr 2002, Roboter 10-16):

  Hier liefern die Motoren 2048 Ticks pro Motorumdrehung (internes Getriebe mit eingerechnet). Die Umsetzung hier beträgt 10/1, also werden 20480 Ticks/Radumdrehung erwartet.

Bei der rein manuellen Bewegung der Räder können sich noch recht große Abweichungen ergeben. Bei einer Abweichung von etwa fünf Grad z.B. $(512/360) * 5 = 7,11$ bzw. $(20480/360) * 5 = 284,44$ Ticks Abweichung in den Meßwerten.

Messung

Robo.	Rad	rückw.										vorw.
		-2	-1	-0,75	-0,5	-0,25	0	0,25	0,5	0,75	1	2
01	r	64516	65025	65152	65283	65409	0	125	258	394	515	1026
	l	1022	509	398	258	127	0	65423	65283	65150	65023	64511
02	r	64513	65024	65147	65278	65409	0	126	256	386	513	1025
	l	1023	512	396	261	124	0	65418	65284	65149	65024	64510
03	r	64514	65026				0				512	1023
	l	1021	510				0				65024	64508
04	r	64511	65023				0				511	1023
	l	1020	512				0				65022	64511
05	r	64514	65026				0				514	1024
	l	1018	510				0				65019	64509
06	r	64512	65023				0				514	1026
	l	1020	507				0				65022	64509
07	r	64520	65029				0				510	1024
	l	1024	512				0				65023	64512
10	r	24557	45036				0				20500	41014
	l	40882	20453				0				45170	24667
11	r	24482	45108				0				20527	40950
	l	40968	20488				0				45008	24484
12	r	24545	45053				0				20432	40954
	l	41000	20471				0				45099	24545
13	r	24670	45019	50117	55369	60575	0	5130	10274	15458	20451	40928
	l	40982	20520	15512	10623	5145	0	60741	55460	50085	45119	24598
14	r	24565	45041	50144	55288	60784	0	5122	10346	15446	20481	40965
	l	40874	20487	15118	10200	5112	0	60326	55062	49957	45105	24745
15	r	24597	45070				0				20414	40865
	l	41004	20453				0				45019	24574
16	r	24560	45090				0				20528	40999
	l	40912	20431				0				44945	24600

Bewertung Als erstes fallen die unterschiedlichen Richtungen der Zählung auf. Rechte Räder werden bei einer Vorwärtsfahrt immer positiv gezählt, linke Räder immer negativ. Dies ist bei allen Robotern gleich, eine Einstellung “Verpolt”, wie sie in den Robotersourcen Verwendung findet, ist also hier nicht notwendig.

Die neue Baureihe hat in diesem Test bereits einen Überlauf erzeugt: z.B. rechtes Rad des Roboter 13: vorwärts wird von 0 über 5130 usw. bis 40928 gezählt, rückwärts über 60575 usw. bis 24670. Somit hat sich bei der Fahrt rückwärts ein Wert kleiner als bei der Fahrt vorwärts eingestellt. Wenn

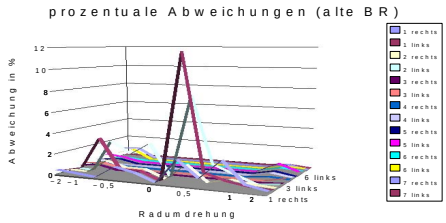


Abbildung 3.5: Abweichungen von den erwarteten Werten, alte Baureihe

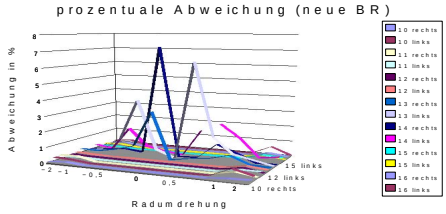


Abbildung 3.6: Abweichungen von den erwarteten Werten, Baureihe 2002

man die Werte auf positiv und negativ umrechnet, heißt das, das Intervall, aus dem die Meßwerte hier stammen, ist [-40866, 40928], also insgesamt 81 732 Werte. Das sind mehr, als die Datenstruktur unsigned int (mit 16 Bit) aufnehmen könnte. Also ist ein Überlauf aufgetreten. In den Diagrammen wurde diese Abweichung bereits herausgerechnet.

Alle Abweichungen, die sich aus den Messungen ergeben, sind durch die schon oben angesprochenen Meßungenauigkeiten erklärbar und liegen maximal bei etwa 7% des Optimalwertes.

Schlußfolgerungen Aufgrund der wesentlich häufigeren Überlaufsituationen der Wertebereiche sollten die Sourcen der Roboter diesbezüglich auf etwaige Anfälligkeiten genau geprüft werden.

Auf Verpolung braucht in den hier getesteten Baureihen nicht geachtet zu werden, jedoch kann es passieren, daß bei Reparaturen die Motoren nicht wieder korrekt angeschlossen werden. Daher ist zu empfehlen, die verkürzten Radencodertests nach jeder Roboter-Reparatur durchzuführen.

3.1.8 offene Tests

3.1.8.1 PWM

Anforderungen an diese Komponente Es sollen die in mm/s gegebenen Radgeschwindigkeiten in passende PWM-Werte umgerechnet werden, damit sich der Roboter mit der gewünschten Geschwin-

digkeit fortbewegt.

Mögliche Fehler

- Die Roboter fahren nicht mit der gewünschten Geschwindigkeit, weil z.B. die Umrechnung falsch ist.

Abhängigkeit zwischen Geschwindigkeiten und PWM ermitteln

- Testverfahren
  vollständige Messkurve über den gesamten Messbereich erstellen.
  Testen, ob die Spannung der Batterie Einfluss auf das Verhältnis zwischen Geschwindigkeit und PWM-Wert hat.
- Durchführbarkeit
  Messgerät für Geschwindigkeiten und genügend Platz sind notwendig.

3.1.8.2 PID-Regler

Anforderungen an diese Komponente "Der Roboter soll einer Strecke geradlinig folgen können, so dass sich während der Fahrt keine signifikanten Änderungen in der Ausrichtung ergeben und der Roboter seitlich nicht signifikant von der Strecke abweicht." [Klu01, S. 104]

Mögliche Fehler

- Roboter fährt nicht geradeaus, sondern fährt eine "Schlängellinie" oder eine Kurve.

Geradeaus Fahrt

- Testverfahren
  Roboter mit verschiedenen Geschwindigkeiten geradeaus fahren lassen. Um Fehler, z.B. durch unsymmetrische Radbereifung auszuschließen, sollten die Tests sowohl für die Vorwärtsfahrt als auch für die Rückwartsfahrt durchgeführt werden.
- Durchführbarkeit
  möglich

3.1.8.3 Rampe

Anforderungen an diese Komponente

1. Die Rampe soll dafür sorgen, dass der Roboter seine Geschwindigkeit verändern kann, ohne dass er außer Kontrolle gerät oder seine Räder durchdrehen.
2. Die Rampe soll die Differenz zwischen dem linken und dem rechten Rad so anpassen, dass er bei Kurvenfahrten nicht außer Kontrolle gerät.

Mögliche Fehler

- durchdrehende Räder
- Roboter gerät außer Kontrolle

Güte

1. $\text{Güte}(v) = \frac{1}{\text{Zeit bis zum Erreichen der Geschwindigkeit v}}$
2. $\text{Güte}(v) = \frac{1}{\text{minmal zu fahrender Radius bei Geschwindigkeit v}}$

Beide Gleichungen sollen auf ihrem Definitionsbereich maximiert werden.

gerade Fahrt

- Testverfahren
  Zeit bis zum Erreichen von verschiedenen Geschwindigkeiten messen.
- Durchführbarkeit
  Geschwindigkeitsmessgerät wird benötigt.

Kurvenfahrt

- Testverfahren
  ermitteln, welche Kurvenradien der Roboter bei verschiedenen Geschwindigkeiten fahren kann, ohne außer Kontrolle zu geraten.
- Durchführbarkeit
  Geschwindigkeitsmessgerät wird benötigt.

3.1.8.4 Radencoder und SimulateMovement

Anforderungen an diese Komponenten Pro Radumdrehung werden eine bestimmte Menge von Impulsen zurückgeliefert. Daraus kann berechnet werden, wie weit sich der Roboter bewegt hat, falls die Räder zwischenzeitlich nicht durchgedreht haben.

Mögliche Fehler

- Die angegebene Menge an Impulsen pro Radumdrehung stimmen nicht mit der tatsächlichen Menge überein.
- Die Berechnung der zurückgelegten Strecke ist nicht korrekt.

Bewegungstest

- Testverfahren
  Roboter gewisse Strecken fahren lassen oder ihn diese Strecken schieben und danach die tatsächliche mit der errechneten Bewegung vergleichen.
- Durchführbarkeit
  bidirektionales Funkprotokoll ist notwendig, damit die errechnete Bewegung übertragen werden kann.

3.1.8.5 Delta-Befehl

Anforderungen an diese Komponente Der Roboter soll sich um den gewünschten Winkel drehen.

Mögliche Fehler

- Roboter dreht sich nicht um den gewünschten, sondern um einen anderen Winkel.
- Roboter schwenkt über

Drehtest

- Testverfahren

  Den Roboter um verschiedene Winkel bei verschiedenen Geschwindigkeiten drehen lassen und messen, ob die Drehung stimmt.

- Durchführbarkeit

  möglich

3.2 Inbetriebnahme der Koreanischen Roboter

Das koreanische Roboterfussball System besteht aus 5 Robotern, einer Kamera, sowie der Steuersoftware (für Windows). Da die koreanischen Roboter über keinen DSP verfügen, werden hier die Radgeschwindigkeiten direkt gesetzt. Die Integration in *robosoccer* setzt daher ein Funkmodul voraus, das eventuelle DSP Befehle direkt in Radgeschwindigkeiten für diese Roboter umsetzt, sprich auf der Hostseite dekodiert werden.

Ein Datenpaket für die koreanischen Roboter sieht wie folgt aus:

Startbyte	Roboter 1	Roboter 2	Roboter 3	Roboter 4	Roboter 5
1 Byte	2 Byte	2 Byte	2 Byte	2 Byte	2 Byte

Neben dem Startbyte wird also jeder Roboter mit 2 Bytes angesteuert. Für jeden Roboter setzt das erste Byte die Radgeschwindigkeit des linken Rads, das zweite Byte die Geschwindigkeit des rechten.

Die Räder sind dabei mit folgenden Werten anzusteuern:

Wert	Wirkung
0x02 = 2	max Rückwärts
⋮	⋮
0x7e = 126	min Rückwärts
0x7f = 127	Stop
0x80 = 128	min Vorwärts
⋮	⋮
0x7e = 254	max Vorwärts

Empirische Versuche haben gezeigt, das der Geschwindigkeitsanstieg und -abfall linear ist. Ein Schritt nach obiger Tabelle entspricht damit einem Geschwindigkeitsanstieg (-abfall) um ca. $16.8mm/s \sim 17.0mm/s$[1].

Da für die Rückwärtsfahrt nur 125 Einheiten zur Verfügung stehen, für die Vorwärtsfahrt jedoch 127, ergibt sich eine maximale Geschwindigkeit nach $125 * 17mm/s = 2125mm/s$ gesetzt, um die Ansteuerung in beide *Richtungen* transparent zu halten.

[1] empirisch ermittelte Werte

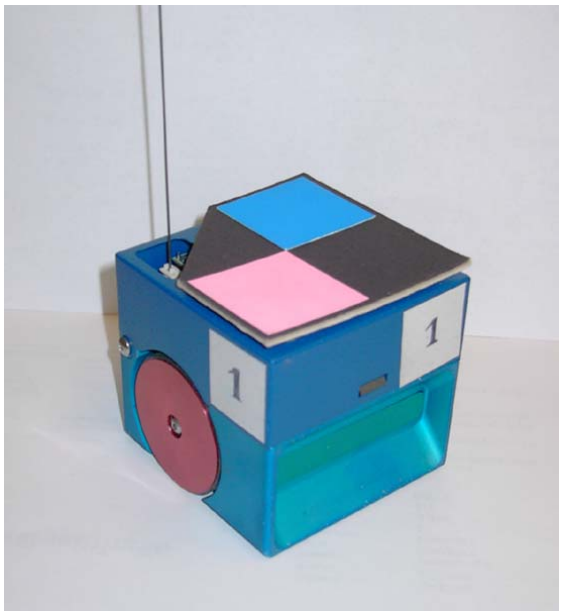


Abbildung 3.7: Korea Roboter

Der Modus *DeltaSpeed* musste angepasst werden, da die koreanischen Roboter über keinen DSP verfügen, der solche Befehle auswerten kann. Hier muss der Befehl direkt in Radgeschwindigkeiten umgewandelt werden und diese dann explizit gesetzt werden. Die Umrechnung erfolgt analog zum Verfahren, wie es auf den DSPs eingesetzt wird, nachzulesen in [PG362].

Die Rampen werden im aktuellen System des LS 1 auf den Robotern gesetzt. Dies ist wegen nicht vorhandener DSPs bei den koreanischen Robotern nicht möglich, und muss daher auch im Funkmodul berücksichtigt werden. Diese Problemstellung ist allerdings noch offen, da sich Aufgaben mit höherer Priorität im Verlauf der Projektgruppe ergeben haben.

3.3 Kamera

3.3.1 Ausgangssituation

Bereits zu Beginn der PG 416 wurde das bestehende Roboterfussball System von 3-gegen-3 auf 5-gegen-5 umgestellt. Die Umstellung brachte einige Schwierigkeiten mit sich, darunter Probleme mit der Strategie, sowie mit der Bildverarbeitung, bzw. der Kamera.

Die zwei Kameras, die am LS 1 eingesetzt werden, sind derzeit die *JAI MCL 1500*, und die *SDC-410ND*. Letztere Kamera lag dem koreanischen Roboterfussball System bei, das dem LS 1 zur Verfügung steht. Beide Kameras sind analog und liefern ein NTSC Signal. Die Digitalisierung des Bildes erfolgt durch eine Framegrabber Karte vom Typ *Meteor I* der Firma Matrox.

Die Bildverarbeitung, wie sie in *robosoccer* eingesetzt wird, wertet aus Effizienzgründen nur jedes zweite Halbbild aus. Dadurch werden Kammeffekte vermieden, die Auflösung jedoch halbiert. Durch die Umstellung auf 5-gegen-5, und die damit gewachsenen Spielfeldmaße, ergeben sich damit einige Probleme bei der Bildanalyse. Durch einen Parallaxeneffekt kann eine Positionszuordnung durch die Bildverarbeitung um bis zu 4 cm von der tatsächlichen Position abweichen. Selbst wenn ein Roboter direkt unter der Kamera steht, und somit kein Parallaxeneffekt auftritt, ergibt sich eine Ungenauigkeit durch die Tatsache, das auf mehrere Zentimeter nur ein Pixel zur Verfügung steht. Nicht zuletzt hat sich der Fischaugeneffekt durch das größere Spielfeld verstärkt.

Neben diesen Problemen ergeben sich noch spezielle Probleme mit den jeweiligen Kameras. Die *JAI MCL 1500* ist nicht in der Lage, das komplette Spielfeld bei 5-gegen-5 aufzunehmen. Die *SDC-410ND* leidet an Farbverfälschungen und Farbmusterung.

Für weitere Informationen zu den Problemen der Bildverarbeitung verweisen wir auf [Wei03].

3.3.2 Eigenschaften einer neuen Kamera

Einige Probleme der aktuellen Kameras ergeben sich also, wie besprochen, vor allem aus der schlechten Auflösung (bedingt, unter anderem, durch die Auswertung von Halbbildern). Die PG 416 ist mit folgenden Anforderungen auf die Suche nach einer neuen Kamera gegangen:

- Auflösung mindestens 640×480
- Progressive Scan

Durch den Progressive Scan Modus liefert die Kamera also Vollbilder. Damit kann die Bildverarbeitung also Vollbilder auswerten, ohne das etwa Kammeffekte auftreten würden. Zudem steht dann die volle Auslösung der Kamera zur Verfügung.

- Digitaler Anschluß

  Der Anschluß an das Hostsystem sollte über eine digitale Schnittstelle erfolgen, etwa Firewire (*IEEE-1394*) oder andere gängige Standards (*RS-644 LVDS*).

- 30 Bilder/Sekunde

  Da die Roboter eine Geschwindigkeit von bis zu $2m/s$ erreichen können, ist die Tauglichkeit einer Kamera auch davon abhängig, wieviele Bilder pro Sekunde geliefert werden können.

- Keine komprimierte Datenübertragung

  Die Dekompression von Daten, wie sie etwa von DV Kameras geliefert wird, erfordert ein zusätzliches Maß an Zeit. Zudem erschwert dies die Anpassung an das bestehende System des LS 1.

Unter Linux ist der Betrieb einer Kamera über eine andere Schnittstelle als Firewire schwierig, da keine Treiber zur Verfügung stehen. Konsequenterweise kam nur eine Firewire Kamera in Frage. Weiter eingeschränkt wurde die Auswahl der Kamera durch das Budget. Kameras ohne Komprimierung mit YUV 4:4:4 liegen dabei deutlich über dem Budget, es kommen also nur Kameras mit YUV 4:2:2 in Frage. Diese Einschränkung setzt sich fort im Hinblick auf die Auflösung: Kameras mit 640×480 und 30 Bildern pro Sekunde liegen in der Regel oberhalb der 2000 Euro Marke.

Als Ergebnis der Recherchen kommt zunächst nur die *DFW-V500* von Sony in Frage. Diese erfüllt, nach ersten Tests, die gegebenen Anforderungen.

3.3.3 Kamera- und Adapterkartentest

3.3.3.1 Umgebung

In folgender Umgebung wurde dieser Test durchgeführt:

Rechner: fira, Vergleichsplattform: soccer

Adapterkarte: Digitus Connect IEEE 1394 PCI Add-on Card (siehe [Dig]), Vergleichsplattform: Meteor I

Kamera: Sony DFW-V500 (siehe [Son99]), Vergleichsplattform: Samsung-SDC-410ND

[Bemerkungen:] zusätzlich installierte Software: Linux Kernel 2.4.20, libdc-1394-0.9.1, coriander-0.27.1, libraw1394-0.9.0

3.3.3.2 Durchführung

Nach Installation der Software wurde die Kamera an der Deckenhalterung befestigt und mit dem Rechner verbunden. Auf das Spielfeld wurden alle uns zur Verfügung stehenden Patches verteilt, auch der Ball. Ähnliche Farben wurden in kurzen Abständen voneinander abgelegt, um eine bessere Vergleichsmöglichkeit zu erreichen. Mit dem Programm *coriander* wurde das Kamerabild zunächst auf dem Bildschirm ausgegeben und die Kamera so eingestellt, daß sie ein optimales Standbild wiedergibt. Anschließend wurde auf soccer (ohne irgendwelche zusätzliche Software) mit *mvid* ebenfalls ein Standbild aufgenommen und die Samsung Kamera (angeschlossen an soccer) so gut wie möglich eingestellt. Von beiden Systemen wurden anschließend Bildschirmfotos geschossen – bei *mvid* mit *xv* screenshot-Funktion und bei *coriander* via integriertem raw-Speichermodus (Datei kann mittels *yuvtoppm* in ein von *xv* lesbares Format umgewandelt werden).

Ein zweiter Test wurde anschließend mit den eigenen Patches gemacht. Es wurde eine Spielsituation nachgestellt, in der ein Roboterpaar Patch an Patch und die anderen Roboter über das Spielfeld verteilt liegen. Wiederum wurden Bildschirmfotos angefertigt.

Abschließend wurde noch ein dritter Test durchgeführt, bei dem ein Roboter von Hand über das Spielfeld gestoßen wurde, während die Aufzeichnung lief.

Schließlich wurden sämtliche Bilder zur Auswertung an Norman Weiß, den Experten für die Bildverarbeitung, übersandt und um eine Stellungnahme gebeten.

3.3.3.3 zu erwartende Ergebnisse

Die angeschlossene Kamera sollte zumindest keine schlechteren Bilder liefern als die Samsung-Kamera.

3.3.3.4 Auswertung

Eine Auswahl der geschossenen Bilder, links jeweils die neue Sony-Kamera, rechts die Samsung. Schon die ersten Bilder der Sony-Kamera zeigten sehr klare Farben auf den Patches. Bei der Bewegung des

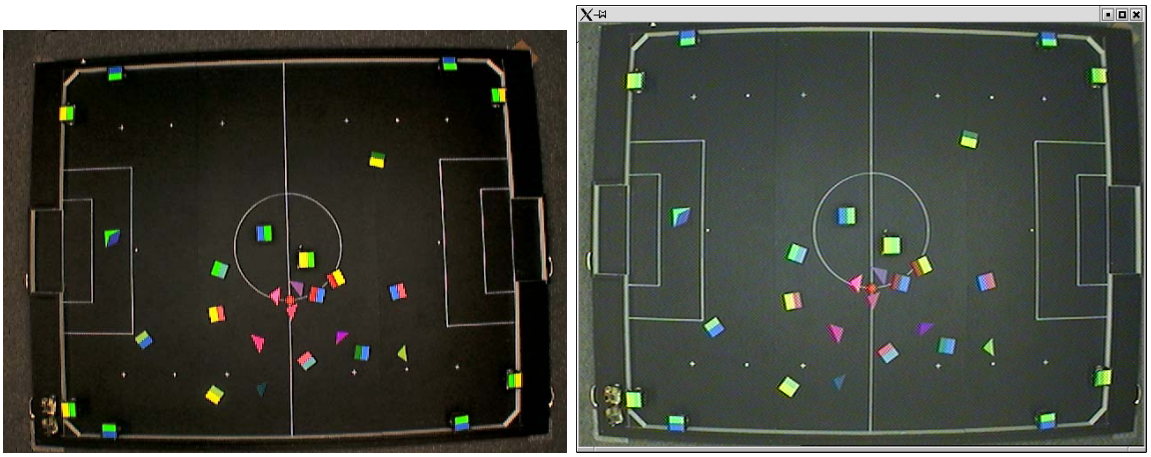


Abbildung 3.8: Test 1

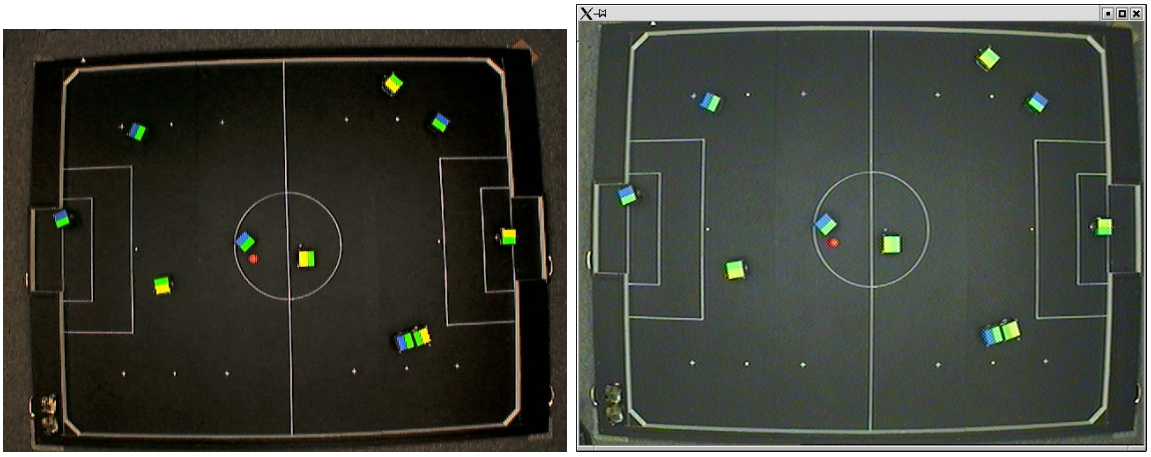


Abbildung 3.9: Test 2

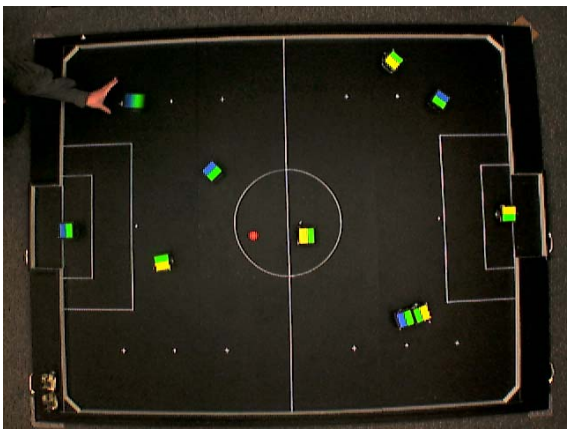


Abbildung 3.10: Test 3

Roboters auf dem Spielfeld zog das Bild jedoch deutliche Schlieren. Die Vermutung liegt nahe, daß die Belichtungszeit für den Test zu groß eingestellt war.

Als Experte in der Bildverarbeitung beurteilte Norman Weiss die Kamera so:

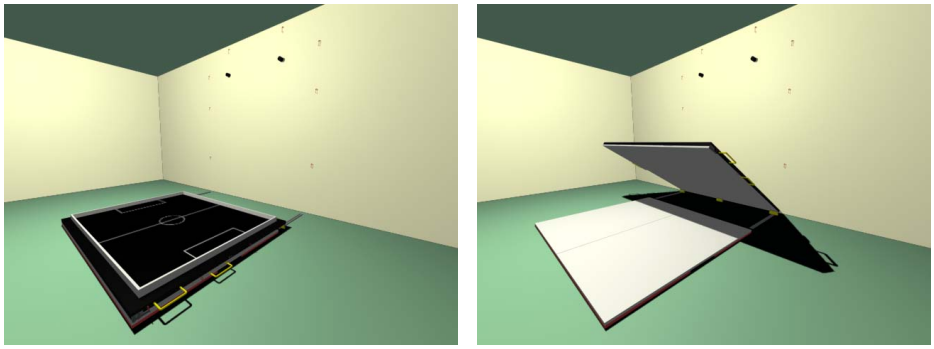
> „Die Kamera ist qualitätsmäßig auf jeden Fall das Beste, was wir bisher hatten. Die Farben sind wirklich kräftig (Die Farbflächen sind fast vollständig S=255!), homogen und verlaufen nicht ineinander. Die Kamera ist sehr kontraststark, d.h. Farben und Spielfeld heben sich sehr gut voneinander ab. Ich habe die Bilder auch mal mit meiner BV getestet, die Auswertung dürfte problemlos möglich sein, im Vergleich zur Samsung ist diese Kamera auf jeden Fall eine Wohltat!
>
> [Ein] paar kleine Punkte bleiben trotzdem:
>
> - Bei Bewegung der Roboter verwischen die Farben ineinander. Da ich nicht weiß, wie schnell der bewegte Roboter bei der Aufnahme war, kann ich auch nicht sagen, ob das noch ein Problem werden wird. Habt ihr irgendeine Möglichkeit, die Einstellwerte der Kamera auszulesen? Ich würde mal schätzen, die Aufnahmen wurden mit einer sehr kleinen Blende und langer Belichtungszeit gemacht, was natürlich nicht sinnvoll ist. Läßt sich das Problem durch Verkürzen der Belichtungszeit nicht in den Griff bekommen, heißt das, daß die Kamera Halbbilder aufnimmt, mittelt und als Vollbild an den Rechner schickt, was aber hoffentlich nicht der Fall ist.
> - Der Ball sieht - hmmmm - komisch aus! Ein sehr lustiges Muster, das die BV hoffentlich nicht stört (wird alles als rot erkannt), aber trotzdem nicht schön ist. Liegt das an der Oberfläche des Balls?
> - Wenn man die drei Bilder mal hintereinander auflaufen läßt (als Daumenkino) sieht man, daß die dunklen Flächen ein nicht unerhebliches Rauschen aufweisen. Das betrifft zwar nicht die Farbflächen, rein visuell scheinen die rauschfrei zu sein, aber trotzdem.
>
> Fazit: Scheint eine sehr gute Kamera zu sein (besser als alles, was wir bisher hatten!), trotzdem sollte sie nochmal im Echtbetrieb (d.h. BV mit laufender Kamera, am Besten während eines Spieles) getestet werden, bevor wir sie kaufen!“

3.3.3.5 Schlußfolgerungen

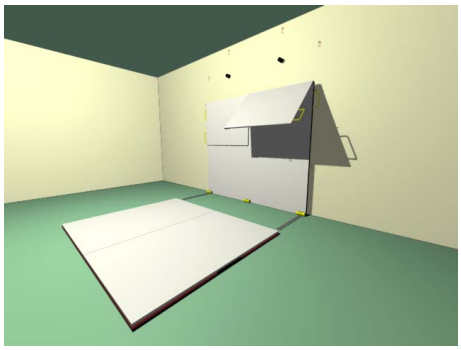
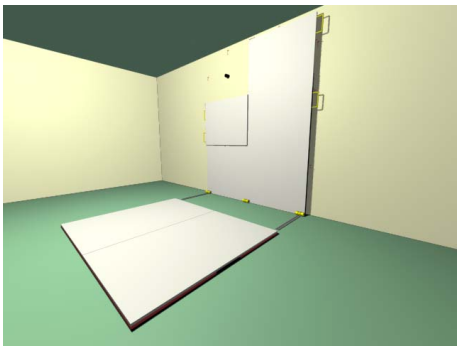
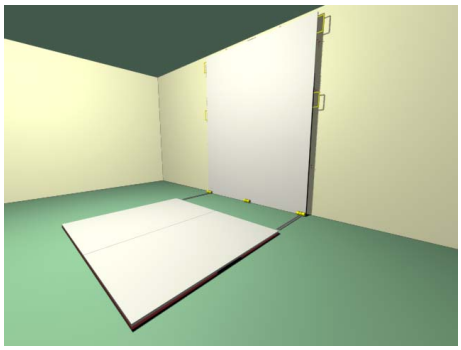
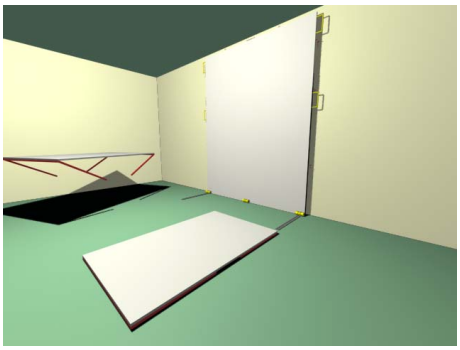
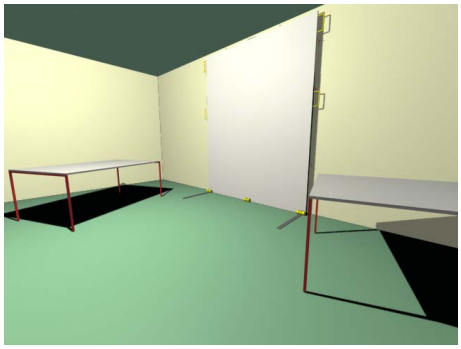
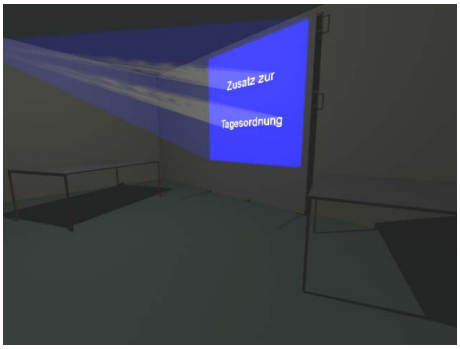
Gemäß dem Expertenrat kann bis jetzt nur eine vorläufige Bewertung abgegeben werden, es fehlen noch einige Tests. Man kann aber schon so viel sagen: Bis jetzt sieht alles sehr gut aus.

3.4 Spielfeld und Beleuchtung

Im Projektgruppenraum für Roboter-Fußball befinden sich vier einzelne Spielfelder; zwei kleine für die Spielvariante 3-gegen-3 und zwei große für 5-gegen-5. Die beiden kleinen Spielfelder unterscheiden sich kaum, sie lehnen an der Wand, da der größte Teil der Entwicklungsarbeit sich derzeit auf die Spielversion mit mehr Robotern konzentriert. Die beiden Spielfelder hierfür unterscheiden sich in der Tatsache, daß das derzeit benutzte, selbstgemacht ist und zu Trainingszwecken dient. Das andere ist das eigens für die GI-Tagung angefertigte Turnierspielfeld, welches zusätzlich einen separaten Unterbau zum Ausgleich von Bodenunebenheiten sowie vier abgeschrägte Umrandungselemente zur besseren Optik oder wahlweise auch zur Anbringung von Werbung besitzt. Eine detaillierte Aufbauanleitung für das komplette Turnierspielfeld inkl. Überbau und Beleuchtung sowie eine genaue Maßzeichnung der Spielfelder für den Modus 5-gegen-5 befinden sich im Anhang dieser Ausarbeitung. Das Trainigsspielfeld befindet sich meist auf dem Boden des Raumes und dient Spielsimulationen und Testzwecken. Da hierdurch nur noch wenig Platz zum Laufen bleibt, wurde eine Idee durchdacht, die es ermöglicht den gesamten Raum in kurzer Zeit in einen Seminarraum mit einer beschreibbaren Projektionswand und zusätzlichen Tischen zu verwandeln. Die folgenden Bilder zeigen eine Simulation dieses Vorganges. Da jedoch ein separater Seminarraum für die regelmäßigen Projektgruppensitzungen zur Verfügung stand wurde diese Idee nur dahingehend verwirklicht, daß das Trainingsspielfeld mit Griffen versehen wurde, die ein einfaches Hochklappen an die Wand bzw. an das sich bereits dort befindende Turnierspielfeld ermöglichen.



Zur genaueren Positionierung des Spielfeldes wurden auf dem Boden Klebemarkierungen angebracht. Somit ist gewährleistet, dass sich das Spielfeld stets zentriert unter der Kamera befindet. Die Kamera selbst hängt an einer neuen, eigens dafür hergestellten Halterung unter der Decke. Diese Halterung ermöglicht eine Verschiebungen in alle Dimensionen wie auch die Anbringung von zwei Kameras. Als Adapter für die Kameras dienen zwei hierfür angebrachte 3D-Köpfe, die weitere vielfältige Justie-







Zusatz zur
Tagesordnung

rungsmöglichkeiten bieten und nach einfacher Demontage auch über dem Turnierspielfeld zum Einsatz kommen. Als Stromversorgung dient eine auf der Zwischendenke befindliche Mehrfachsteckdose. Zusätzlich zu den genannten Umbaumaßnahmen kam noch die Anbringung einer Beleuchtung. Über dem Spielfeld befinden sich nun vier Doppelneonröhren-Leuchten, die jeweils mit Kaltlicht-Röhren zur besseren Farbreflexion bestückt wurden. Um eine ausreichende Beleuchtung zu erzielen sollten für ein Testspiel jedoch sämtliche Leuchten im Raum angeschaltet werden. Die Spielfeldleuchten wie auch die Stromversorgung der Kameras können über einen separaten Wandschalter betätigt werden. Eine Vielzahl von provisorischen Einrichtungen wurde somit durch praktikable Versionen ersetzt.

Kapitel 4

Veranstaltungs- und Erfahrungsberichte

4.1 Auftritt im Rahmen des JOY-Wettbewerbs

Am 17. Juli 2002 wurden im Depot an der Immermannstraße vor rund 300 Gästen die Gewinner des IT-Azubi-Wettbewerbs JOY 2002 bekannt gegeben. JOY steht für "Junior of the year" und ist eine Initiative des Dortmund-Project [1]. Bei diesem Wettbewerb hatten seit Februar 58 IT-Azubis aus 27 Unternehmen aus Dortmund und Umgebung mit Projekten aus ihrem Arbeitsalltag um den Titel Junior of the Year gerungen.

Abbildung 4.1: Die Joy-Trophäe

[1] http://www.dortmund-project.de

Im Rahmen der Veranstaltung sorgte die Projektgruppe mit einer Demonstration des Robosoccer-Systems für Unterhaltung und Abwechslung. So konnte man Roboterfußball einem fachlich interessierten Publikum nahebringen. Außerdem war die Einladung eine gute Gelegenheit, das gesamte System für den Ernstfall zu testen. So konnte man ohne großes Risiko sehen, ob z.B. die erstellten Checklisten den Praxistest bestehen.

Das System arbeitete praktisch einwandfrei und auch die Projektgruppe zeigte sich in der Lage, Robosoccer in Betrieb zu nehmen und zu bedienen. In dieser Hinsicht war der Auftritt also gelungen.

4.2 Auftritt im Rahmen der Aktionstage Dortmund-Project

Vom 23.09.02 bis zum 27.09.02 fanden auf dem Reinoldikirchplatz die Aktionstage des Dortmund-Project (www.dortmund-project.de) statt. Auf Einladung des Dortmund-Project nahm die Projektgruppe an der Veranstaltung teil. Einerseits konnte man sich im Rahmen einer IT-orientierten Veranstaltung präsentieren, andererseits wollte man die Gelegenheit nutzen, kurz vor der GI-Tagung am 30. September nocheinmal das Gesamtsystem einem Praxistest zu unterziehen.

Am 25. September 2002 gab es zwei Blöcke, in denen die Projektgruppe jeweils das Robosoccer-System vorführte:

11:00 - 12:00 Uhr: IT Center Dortmund, Der schnellste IT-Studiengang Deutschlands

12:00 - 12:30 Uhr: Musik, Gespräche am Infozentrum

12:30 - 13:30 Uhr: Dortmund - ein internationaler Standort?

13:30 - 14:00 Uhr: Musik, Gespräche am Infozentrum

14:00 - 15:00 Uhr: Roboter-Fußball am Infozentrum

15:00 - 16:00 Uhr: Talkrunde - start2grow Der IT-Gründungswettbewerb

16:00 - 16:30 Uhr: Musik, Gespräche am Infozentrum

16:30 - 17:30 Uhr: Talkrunde - JOY 2002 Dortmunds großer Azubi-Wettbewerb

17:30 - 18:30 Uhr: Roboter-Fußball am Infozentrum

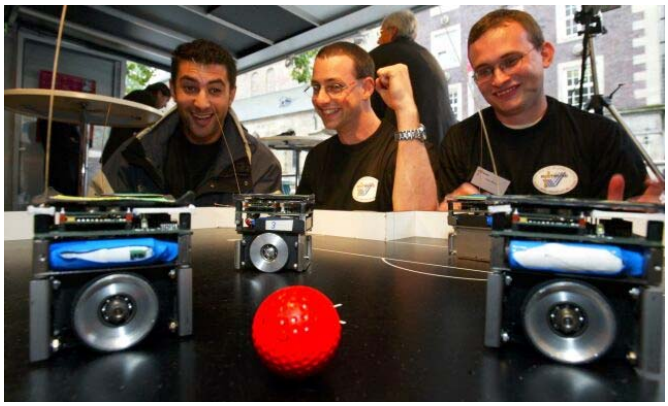


Abbildung 4.2: Die Roboter im Einsatz

Leider war die Veranstaltung nicht besonders gut besucht - genauer gesagt, war das Publikum wohl nicht ganz das, was sich das Dortmund-Project vorgestellt hatte. Aber vielleicht ist der Platz vor Kentucky Fried Chicken auch nicht die richtige Wahl gewesen, um mögliche IT-Unternehmensgründer in großer Zahl anzutreffen. So musste man größtenteils mit fachlich eher anders orientiertem Publikum Vorlieb nehmen. Leider war somit - zumindest für die PG - das Ziel, Interessierte für den Roboter-Fußball zu gewinnen, zum Scheitern verurteilt.

Aber trotzdem kann und sollte man Lehren aus dem Autritt ziehen: Zukünftige Projektgruppen sollten sich vorher informieren, in welchem Rahmen und vor welchem Publikum der Auftritt stattfinden soll, bevor sie eine Entscheidung für oder gegen eine Teilnahme treffen. Sicherlich ist es wichtig (gerade, wenn ein so großes Event wie in diesem Fall die GI-Tagung unmittelbar bevorsteht) jede sich bietende Gelegenheit wahrzunehmen, um das System unter verschiedenen Bedingungen zu testen und den Ernstfall zu proben.

Allerdings bringt die Teilnahme an so einer Veranstaltung auch immer sehr viel Arbeit mit sich. In diesem Fall handelte es sich insgesamt um einen Aufwand von mindestens 100 Arbeitsstunden, und herausgekommen ist leider nicht viel. Das soll nicht bedeuten, dass angebotene Gelegenheiten, vor Publikum aufzutreten in Zukunft abgelehnt werden sollten - man sollte vorher nur Aufwand und Nutzen kalkulieren!

4.3 GI-Tagung

Vom 30.09. bis zum 03.10.2002 fand in den Dortmunder Westfalenhallen die 32. Jahrestagung der Gesellschaft für Informatik, kurz GI, statt. Die GI informiert an diesen Tagen in Workshops und

Abbildung 4.3: Das Dortmund-Project Maskottchen

Vorträgen über aktuelle Themen der Informatik und der damit verbundenen Forschung. Im Rahmen dieser Tagung veranstalteten die "Dortmund Droids" ein Einladungsturnier der FIRA MiroSot Middle League unter dem offiziellen Titel "FIRA Invitation Championship Dortmund". Hierzu wurden, neben dem Dortmunder Team, fünf weitere internationale Mannschaften eingeladen. Im einzelnen befanden sich unter den Teilnehmern 2 Mannschaften aus Korea, darunter der amtierende Weltmeister, ein Team aus Österreich, eins aus Slowenien und eine weitere Mannschaft aus China.

Zu Beginn war es Aufgabe der Projektgruppe die nötigen Vorbereitungen zu treffen. Bereits ein Tag vorher wurde in einem Saal der Westfalenhallen das komplette und extra hierfür angefertigte Turnierspielfeld inklusive Traversenüberbau aufgebaut.

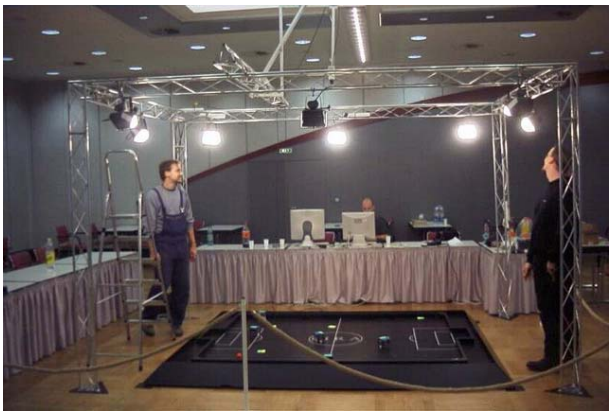


Abbildung 4.4: Aufbau Turnierspielfeld

Weiterhin wurden Vorbereitungen getroffen, um den internationalen Gästen ihren Aufenthalt so angenehm wie möglich zu gestalten. Als alle Gäste eingetroffen waren und ihr gesamtes Equipment installiert hatten, konnte die Auslosung der beiden Vorrunden-Gruppen erfolgen, die somit den Beginn des Turniers einläutete. An den ersten beiden Tagen wurden die Halbfinal-Teilnehmer ermittelt; der koreanische Weltmeister zog problemlos in die Endrunde ein, weiterhin konnte sich das chinesische und das slowenische Team durchsetzen; auch die Dortmunder Mannschaft erreichte zur Erleichterung aller ortsansässigen Beteiligten die Endrunde.

Diese wurde in einem größeren Saal abgehalten, was einen weiteren Umzug des gesamten Equipments mitsich brachte, von den Teilnehmern aber gern hingenommen wurde. Am Ende setzte sich der Weltmeister aus Korea im Finale der beiden überragenden Mannschaften gegen die starke slowenische Mannschaft durch, die Dortmunder belegten mit dem vierten Platz ein zufriedenstellendes Ergebnis. Schließlich konnte nach dem Turnier jede Mannschaft einen eigens angefertigten Pokal in den Händen halten.

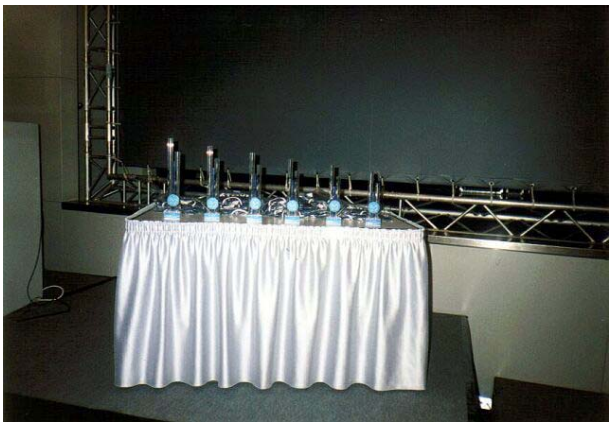


Abbildung 4.5: Pokale

Viel wichtiger als der Turnierverlauf selbst waren aber andere Dinge. So zeigte sich während des Turniers eine immer stärker werdende Vertrautheit zwischen den eigentlich so unterschiedlichen Menschen aus aller Welt. Gemeinsame abendliche Unternehmungen während dieses Turniers trugen dazu bei, dass sich alle Teilnehmer in kurzer Zeit zu einer “großen Familie” zusammenschlossen. Das gemeinsame Interesse am Roboterfußball wie auch daran andere Menschen kennenzulernen förderte das Zusammenfügen dieser Gemeinschaft.

Als am abschließenden Turniertag sich die ohnehin schon zahlreiche Zuschauerzahl nochmals erhöhte war die Stimmung an ihrem Höhepunkt angekommen. Beifall für erzielte Tore und andere spontane

Reaktionen ließen die Teilnehmer der Projektgruppe den Eindruck eines großen Turniers gewinnen. Es erschien zeitweise so als hätten die Zuschauer vergessen, dass sie hier “nur” das Fußballspiel kleiner Maschinen betrachteten, ein schön anzusehendes Phänomen.

Abbildung 4.6: Zuschauer

Auch dies ist eines der eigentlichen Ziele, die diese Veranstaltung erreicht hatte; die PG-Mitglieder hatten hierdurch einen kompletten Turnierablauf kennengelernt, mit allem was dazu gehört. Dazu zählten Aspekte wie Stimmung unter den Teilnehmern, Zuschauerreaktionen, Interviews für die Presse, Informationsvermittlung für interessierte Zuschauer, aber auch Probleme die kurzfristig auftreten können und schnell behoben oder umgangen werden müssen. Alles in allem war diese Veranstaltung ein voller Erfolg und wurde von allen Beteiligten auch als solcher angesehen und gefeiert.

Abbildung 4.7: Eine große Familie

4.4 Workshop

Am 21. und 22. November 2002 fand an der Universität Dortmund ein internationaler Roboterfussball Workshop statt. Unter dem Titel “Autonomous Intelligent Agents and Robot Technologies” sollten Erfahrungen ausgetauscht und mögliche Ziele der Forschung rund um den Roboterfussball und autonomer Agentensysteme diskutiert werden.

Folgende Personen haben am Workshop teilgenommen:

- Hildebrand, Lars, Universität Dortmund, LS1, Deutschland
- Wickrath, Marco, Universität Dortmund, LS1, Deutschland
- Bursch, Sven, Universität Dortmund, LS1, Deutschland
- Bunzel, Michael, Universität Dortmund, LS1, Deutschland
- Michalski, Christoph, Universität Dortmund, LS1, Deutschland
- Soelter, Marc, Universität Dortmund, LS1, Deutschland
- Honzik, Bohumil, Brno University of Technology, Tschechische Republik
- Solc, Frantisek, Brno University of Technology, Tschechische Republik
- Man-Wook, Han, TU Wien, Österreich
- Kopacek, Peter, TU Wien, Österreich
- Gruer, Pablo, UTMB, Frankreich
- Simonin, Olivier, UTMB, Frankreich
- Birk, Andreas, IUB, Deutschland
- Goerick, Christian, Honda R & D Europe, Deutschland
- Poel, Mannes, Uni-Twente, Niederlande
- Schoute, Albert, Uni-Twente, Niederlande
- Matko, Drago, F. E. Ljubljana, Slowenien
- Dutkiewicz, Piotr, ICSI PUT-Poznan, Polen

- Kozlowski, Krysztof, ICSI PUT-Poznan, Polen
- Klute, Thomas, Universität Dortmund, LS1, Deutschland
- Jesse, Norbert, Universität Dortmund, LS1, Deutschland
- Heuser, Thomas, Dr. Heuser AG, Dortmund, Deutschland

In der ersten Phase des Workshops berichteten Teilnehmer aller vertretenen Institute über bisher gemachte Erfahrungen, aufgetretene Probleme und Forschungserfolge. Diese Phase gestattete allen Teilnehmern einen besseren Einblick in die Arbeit der anderen Institute und skizzierte bisherige Forschungsschwerpunkte. Detailfragen ermöglichten zudem einen Erfahrungsaustausch, der eventuell neue Impulse für die eigene Forschungsarbeit liefert.

In der zweiten Phase wurde in einer offenen Diskussion über die nächsten möglichen Forschungsschwerpunkte diskutiert. Dabei kristallisierten sich folgende Schwerpunkte heraus:

- verbesserte Wiederverwertung von Komponenten (Hardware, Software, Wissen)
- Robotersteuerung auf Basis von KI-Techniken
- verbesserte Kollisionsvermeidung
- Fahrtplanung in nicht kooperativen Umgebungen
- dynamische Analyse des Verhaltens gegnerischer Roboter / Predicting
- verbesserte Kommunikationsprotokolle und direkte Kommunikation zwischen den Robotern
- weitere Verlagerung der Intelligenz vom Hostsystem zu den Robotern
- Entwicklung einer Plattform für Multi Agenten Systeme
- verbesserte Bildverarbeitung
- verbesserte Programmlaufzeiten / Optimierungen
- Roboter als Edutainment-Plattform

Gleichzeitig wurde ebenfalls die Art des weiteren Vorgehens diskutiert. Zur Bündelung der vorhandenen Ressourcen und zur Nutzung spezifischer Fachgebiete wurde über die Vorzüge und die mögliche Art eines Forschungsnetzwerks diskutiert.

Neben der Teilnahme am Workshop wurde von den Projektgruppenmitgliedern eine Präsentation der am Lehrstuhl eingesetzten Roboterfussball Hardware durchgeführt. In diesem Rahmen fand ebenfalls ein Demonstrationsspiel unter den eigenen Robotern statt. So bot sich gerade für Teilnehmer, welche keinen direkten Kontakt zum Roboterfussball besaßen, die Möglichkeit einem Live ausgetragenem Spiel zuzusehen.

Kapitel 5

Offene Ideen und Konzepte

5.1 Physikalisches Modell

Für eine optimale Fahrt der Roboter ist es wesentlich, die auf die Roboter einwirkenden Kräfte abschätzen zu können. Kennt man diese Kräfte, so ist es z.B. möglich, eine Kurve mit minimalem Radius bei maximaler Geschwindigkeit zu fahren, ohne dass die Roboter die Traktion verlieren.

Im folgenden Kapitel wird ein Modell vorgestellt, das die auf den Roboter wirkenden physikalischen Kräfte näherungsweise berechnet.

Nach einer kurzen Übersicht über die benutzten Abkürzungen und Konstanten werden Formeln für Reibungskraft, Zentripetalkraft und Beschleunigung aufgestellt. Diese Kräfte dienen abschließend als Grundlage für den Entwurf einer neuen Rampe.

5.1.1 Abkürzungen und Konstanten

F_R	Reibungskraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_N	Normalkraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_{ZP}	Zentripetalkraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_B	Beschleunigungskraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_{B_R}	Beschleunigungskraft pro Rad	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_{Rot}	Rotationskraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_{QR}	Querreibungskraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
F_{LR}	Längsreibungskraft	$\left[\mathrm{N}=\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
μ	Reibungszahl, Reibbeiwert	
m	Masse	[kg]
a	Beschleunigung	$[\frac{\mathrm{m}}{s^2}]$
a_l	Beschleunigung des linken Rades	$[\frac{\mathrm{m}}{s^2}]$
a_r	Beschleunigung des rechten Rades	$[\frac{\mathrm{m}}{s^2}]$
r	Radius	[m]
v	Geschwindigkeit	[m]
h	Höhe des Schwerpunktes	[m]
b	Spurbreite	[m]
g	Erdbeschleunigung	9.81 $\frac{\mathrm{m}}{\mathrm{s}^2}$

5.1.2 Reibungskraft

5.1.2.1 Coulombsche Festkörperreibung

Die Coulombsche Festkörperreibung unterscheidet zwei Zustände: Haftreibung und Gleitreibung.

Reibungskraft = Reibungszahl * Normalkraft

$F_R = \mu * F_N$

μ kann auf einer schiefen Ebene ermittelt werden. Für die Haftreibung misst man, ab welchem Winkel α das Objekt zu rutschen beginnt und für die Gleitreibung, ab welchem Winkel α das Objekt wieder stehenbleibt.

$\mu = tan(\alpha)$

Die Festkörperreibung hat folgende Eigenschaften:

- Haftreibung ist größer als die Gleitreibung
- Reibungskraft F_R ist nur abhängig von der Normalkraft F_N, nicht von der Auflagefläche
- Die Gleitreibung ist geschwindigkeitsunabhängig

5.1.2.2 Reibbeiwert

Da der Belag der Roboterräder aus Gummi ist und Gummi kein Festkörper ist, gilt die Festkörperreibung nicht. In diesem Fall wird jedoch ein sogenannter Reibbeiwert μ definiert, der nicht mehr konstant ist, sondern von verschiedenen Faktoren, wie z.B. Geschwindigkeit, Temperatur, Druck und Auflagefläche abhängig ist

Mit gegebenen μ ist die Berechnung der maximalen Beschleunigung möglich:

$$\begin{aligned} F_R &= \mu * F_N \\ m*a &= \mu * m*g \\ a &= \mu * 9.81 \end{aligned}$$

Im folgenden wird zur Vereinfachung des Modells angenommen, dass der Reibbeiwert konstant und somit unabhängig von der Beschleunigung und der Geschwindigkeit ist.

5.1.2.3 Reibungskraft

Eine Messung der Reibungskraft ist mit Hilfe einer Federwaage möglich. Hierbei sollte die Quer- und Längsreibung gemessen werden (s. 3.1.7.1). Aus der Reibungskraft kann μ berechnet werden. Es gilt

$$\mu = \frac{F_R}{m * g} = \frac{F_R}{m*9.81}$$

5.1.3 Zentripetalkraft

> „Die Zentripetalkraft ist diejenige Kraft, welche auf einen Körper wirken muss, damit er eine gleichmäßige Kreisbewegung ausführt . Sie muss am kreisenden Körper angreifen. Ihre Richtung zeigt zum Zentrum der Kreisbahn.“[Edu02]

$F_{ZP} = m\frac{v^2}{r} \quad \left[N{=}kg * \frac{\left(\frac{m}{s}\right)^2}{m}\right]$

5.1.4 Beschleunigung

Es wird angenommen, dass die Beschleunigung der beiden Räder zum einen in eine geradlinige Bewegung, zum anderen aber auch in eine Drehbewegung umgesetzt wird.

Beispiele:

- $\mathrm{a_l} = 10\left[\frac{\mathrm{m}}{s^2}\right]$; $\mathrm{a_r} = 10\left[\frac{\mathrm{m}}{s^2}\right]$
  geradlinige Bewegung mit einer Beschleunigung von $10\left[\frac{\mathrm{m}}{s^2}\right]$
- $\mathrm{a_l} = 10\left[\frac{\mathrm{m}}{s^2}\right]$; $\mathrm{a_r} = -10\left[\frac{\mathrm{m}}{s^2}\right]$
  Drehbewegung mit einer Beschleunigung von $20\left[\frac{\mathrm{m}}{s^2}\right]$
- $\mathrm{a_l} = 20\left[\frac{\mathrm{m}}{s^2}\right]$; $\mathrm{a_r} = 10\left[\frac{\mathrm{m}}{s^2}\right]$
  geradlinige Bewegung mit einer Beschleunigung von $10\left[\frac{\mathrm{m}}{s^2}\right]$
  Drehbewegung mit einer Beschleunigung von $10\left[\frac{\mathrm{m}}{s^2}\right]$

5.1.4.1 Geradlinige Bewegung

$\mathrm{F} = \mathrm{m} * \mathrm{a_g} \quad \left[\mathrm{N}{=}kg{*}\frac{m}{s^2}\right]$

mit

$$\mathrm{a_g}{=}\begin{cases} \min\left(\mathrm{a_l}, \mathrm{a_r}\right) & \text{falls } \mathrm{a_l} > 0,\ \mathrm{a}_r > 0 \\ \max\left(\mathrm{a_l}, \mathrm{a}_r\right) & \text{falls } \mathrm{a_r} < 0,\ \mathrm{a}_l < 0 \\ 0 & \text{sonst} \end{cases}$$

5.1.4.2 Drehbewegung

$\mathrm{F} = \mathrm{m} * \mathrm{a_D} \quad \left[\mathrm{N}{=}kg{*}\frac{m}{s^2}\right]$

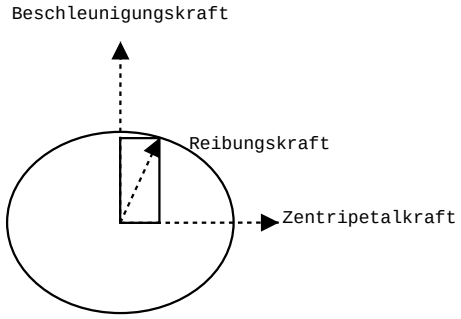
mit

$\mathrm{a_D}{=}\mathrm{a_l} - \mathrm{a_r}$

5.1.4.3 Geradlinige- und Drehbewegung

$\mathrm{F_B} = \mathrm{m} * \left(\mathrm{a_l} - \mathrm{a_r} + \mathrm{a_g}\right) \quad \left[\mathrm{N}{=}kg{*}\frac{m}{s^2}\right]$

Hierbei wird angenommen, dass sowohl der Angriffspunkt als auch die Richtung der beiden Kräfte gleich sind.



5.1.5 Beschleunigungskraft, Reibungskraft und Zentripetalkraft

Zur Vereinfachung des Modells wird angenommen, dass der Angriffspunkt aller drei Kräfte gleich ist und dem Schwerpunkt des Roboters entspricht.

Die folgende Grafik ergibt sich, wenn die Beschleunigungskraft auf der Y-Achse und die Zentripetalkraft auf der X-Achse abgetragen werden. Die X-Komponente der Ellipse beschreibt die Querreibung des Roboters, während die Y-Komponente die Längsreibung beschreibt.

Aus dem Kräfteparallelogramm ergibt sich nach Pythagoras die Formel:

$$F_R^2 = F_{ZP}^2 + F_B^2$$

Aus der Ellipsengleichung:

$$\frac{|F_{ZP}|^2}{|F_{QR}|^2} + \frac{|F_B|^2}{|F_{LR}|^2} = 1$$

.

5.1.6 Kipp- und Standmoment

Ein weiterer Punkt, der untersucht werden muss, ist, ob der Roboter bei Kreisfahrten umfallen kann. Dieses ist der Fall, wenn das Kippmoment größer als das Standmoment wird.

$$\underbrace{Z*h}_{\text{Kippmoment}} = \underbrace{G * \frac{b}{2}}_{\text{Standmoment}}$$

$$\begin{aligned} \mathrm{m} * \frac{v^2}{r} * \mathrm{h} &= \mathrm{m} * \mathrm{g} * \frac{\mathrm{b}}{2} \\ \mathrm{v}^2 &= \frac{\mathrm{g*r*b}}{2\mathrm{h}} \\ \mathrm{v} &= \sqrt{\frac{g*\mathrm{r*b}}{2\mathrm{h}}} \end{aligned}$$

5.1.7 Zwischenergebnisse

5.1.7.1 Beschleunigung

Die maximale Beschleunigung der Roboter ohne Kurvenfahrt $\left(\mathrm{F_{ZP}} = 0 \Leftrightarrow \mathrm{F_B} = \mathrm{F_R}\right)$ errechnet sich durch $\mathrm{a}=\frac{\mathrm{F_B}}{\mathrm{m}} = \frac{\mathrm{F_R}}{\mathrm{m}}$. Für die neue Baureihe ergibt sich somit eine maximale Beschleunigung zwischen $\mathrm{a}=\frac{1.07\mathrm{N}}{0.545\mathrm{kg}} = 1.96\frac{\mathrm{m}}{\mathrm{s}^2}$ und $\mathrm{a}=\frac{1.42\mathrm{N}}{0.545\mathrm{kg}} = 2.60\frac{\mathrm{m}}{\mathrm{s}^2}$.

5.1.7.2 Kurvenfahrt

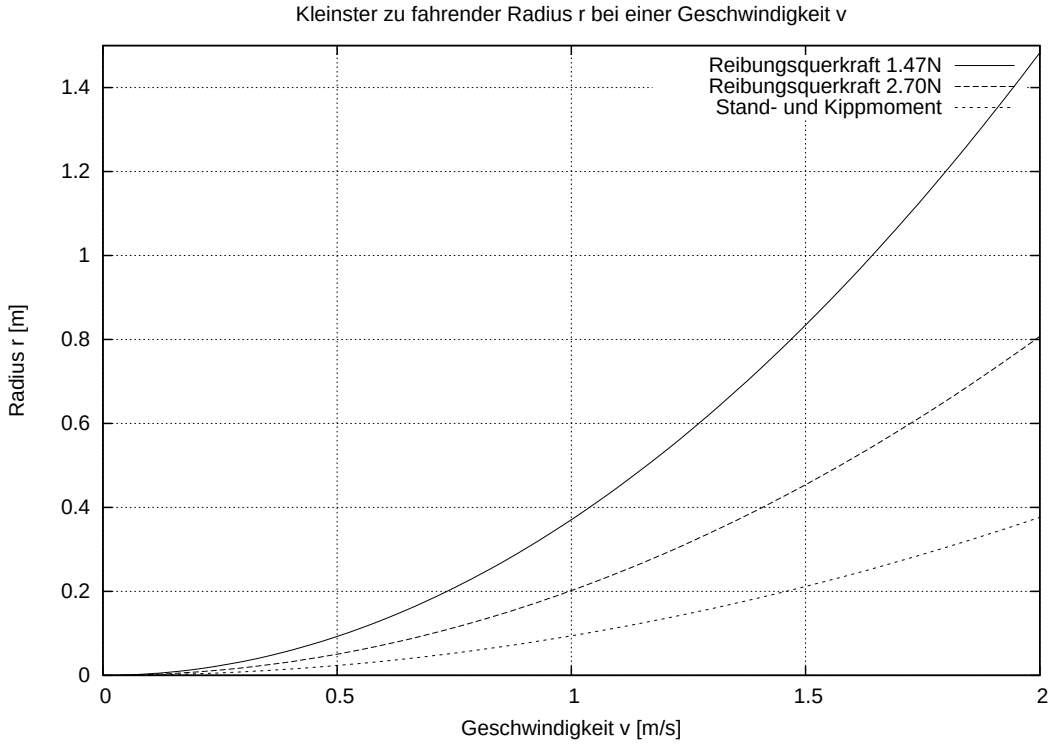
Für den minimalen Kurvenradius für eine nicht beschleunigte Fahrt $\left(\mathrm{F_B} = 0 \Leftrightarrow \mathrm{F_{ZP}} = \mathrm{F_R}\right)$ gilt unter Berücksichtigung der Reibung:

$$\begin{aligned} \mathrm{F_{ZP}} &= \mathrm{m}\frac{\mathrm{v}^2}{\mathrm{r}} \\ \mathrm{r} &= \mathrm{m}\frac{\mathrm{v}^2}{\mathrm{F_{ZP}}} \\ \mathrm{r} &= \mathrm{m}\frac{\mathrm{v}^2}{\mathrm{F_R}} \end{aligned}$$

Unter der Berücksichtigung des Kipp- und Standmoments gilt:

$$\begin{aligned} \mathrm{v}^2 &= \frac{\mathrm{g*r*b}}{2\mathrm{h}} \\ \mathrm{r} &= \frac{\mathrm{v}^2}{\mathrm{g*b}} * 2\mathrm{h} \end{aligned}$$

Die folgende Grafik verdeutlicht, welche Kurvenfahrten bei welcher Geschwindigkeit möglich sind. Es zeigt sich, dass bei den gemessenen Reibbeiwerten das Kipp- und Standmoment vernachlässigt werden kann, da die möglichen Radien deutlich unter denen liegen, die unter Berücksichtigung der Reibung möglich sind.



5.1.7.3 Beschleunigung und Kurvenfahrt

Bei einer Kombination aus Beschleunigung und Kurvenfahrt muss beachtet werden, dass die Ungleichung $\frac{|F_{ZP}|^2}{|F_{QR}|^2} + \frac{|F_B|^2}{|F_{LR}|^2} \leq 1$ erfüllt ist.

5.1.8 Bedeutung für die Implementierung von Rampen

Für die Implementierung von neuen Rampen muss die maximal mögliche Beschleunigung während einer Kurvenfahrt berechenbar sein:

$$\begin{aligned} \frac{|F_{ZP}|^2}{|F_{QR}|^2} + \frac{|F_B|^2}{|F_{LR}|^2} &= 1 \\ \frac{|F_B|^2}{|F_{LR}|^2} &= 1 - \frac{|F_{ZP}|^2}{|F_{QR}|^2} \\ |F_B| &= \sqrt{\left(1 - \frac{|F_{ZP}|^2}{|F_{QR}|^2}\right) * |F_{LR}|^2} \end{aligned}$$

Aus $F_B = m * (a_l - a_r + a_g)$ folgt:

$$|a_l - a_r + a_g| = \frac{1}{m} \sqrt{\left(1 - \frac{|F_{ZP}|^2}{|F_{QR}|^2}\right) * |F_{LR}|^2} = a_{max}$$

5.1.8.1 Funktionsweise

1. Drehung

   Als erstes wird getestet, ob die gewünschte Drehung möglich ist.

   Dies ist der Fall, falls $|a_{Dl} - a_{Dr}| \leq a_{max}$ gilt. Falls diese Ungleichung nicht erfüllt ist, werden a_{Dl} und a_{Dr} so angepasst, bis diese erfüllt ist.

   $a_{Dl} = \begin{cases} \frac{a_{max}}{2} & \text{bei einer Linksdrehung} \\ -\frac{a_{max}}{2} & \text{bei einer Rechtsdrehung} \end{cases}$

   $a_{Dr} = -a_{Dl}$

2. Geradlinige Beschleunigung

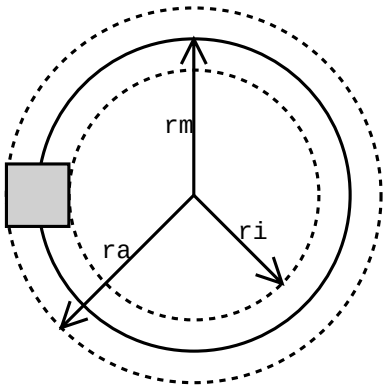
   Eine geradlinige Beschleunigung kann mit maximal $a_g = a_{max} - (a_l - a_r)$ in die gesamte Beschleunigung mit einbezogen werden.

3. Bestimmung der Radbeschleunigungen

   $a_l = a_{Dl} + a_g$

   $a_r = a_{Dr} + a_g$

5.1.8.2 Bemerkungen



Das Setzen von unterschiedlichen Radgeschwindigkeiten führt zu einer Kreisfahrt mit dem Radius r_m.

Das äußere Rad fährt einen Kreis mit dem Radius r_a und das innere Rad einen Kreis mit dem Radius r_i. Nach [Fen02] gilt:

$$\frac{v_i}{v_a} = \frac{r_m - \frac{b}{2}}{r_m + \frac{b}{2}}$$

Die Rampe sollte also versuchen, das Verhältnis $\frac{v_i}{v_a}$ immer entsprechend dem gewünschten Radius beizubehalten.

5.2 Neue Pfadplanung über Kreisanfahrtsalgorithmus

Die Anfahrtsplanung, wie sie bisher vom D.U.R.S.T. eingesetzt wurde (der sog. CMU-Algorithmus), zeigte schon nach wenigen Probeläufen deutliche Schwächen. So hat sich die Projektgruppe entschlossen, einen alternativen Anfahrtsalgorithmus auszuarbeiten. Im folgenden wird dieser Anfahrtsalgorithmus detailliert beschrieben. Ein anderes Team hat übrigens in [Mes98] ebenfalls diesen Algorithmus umgesetzt.

5.2.1 Die Nachteile der CMU-Anfahrtsplanung

Der CMU-Algorithmus hat nach einigen Testspielen ein zu schlechtes Fahrverhalten mit vielen unbegründeten Richtungswechseln gezeigt. Außerdem fuhr der Roboter aufgrund dieses Anfahrtsalgorithmus relativ häufig an dem Ball vorbei. Analysen zeigten dann, daß – neben einigen Implementationsfehlern – es Situationen zu geben scheint, in denen der Roboter aus voller Fahrt eine 90° Kurve in Nullzeit bewerkstelligen können müßte, um dem CMU-Algo. zu folgen. Des weiteren berücksichtigte er kaum einen gegnerischen Roboter. D.h. die Gegnervermeidung läßt sich nur global für jeweils Freund- und Gegnerroboter ein- oder ausschalten (dieses Problem ist jedoch nicht direkt dem CMU-Algo. zuzuordnen); weil aber eine generelle Gegnervermeidung zu einer stark zurückhaltenden Spielweise führt, war die Gegnervermeidung meist während eines Spiels ausgeschaltet.

Es wurde daraufhin vorgeschlagen, einen recht naheliegenden alternativen Pfadplanungsalgorithmus zu implementieren, der zunächst auf einer Kreisbahn seine aktuelle Ausrichtung und Geschwindigkeit in eine günstige Konfiguration bringt (den Roboter in Richtung des Zielpunktes ausrichtet), dann auf einer geraden Strecke beschleunigen kann und schließlich erneut mit einer Kreisbahn auf die Zielausrichtung und -geschwindigkeit einschwenkt.

Der Algorithmus bietet außerdem die Möglichkeit, eine Pfadplanung auf Wegpunktbasis zu realisieren, so daß ein Umfahren von Hindernissen möglich ist, sofern dieses für den *einzelnen* Roboter gewünscht ist.

Die nächsten Abschnitte beschäftigen sich nun näher mit diesem Algorithmus.

5.2.2 Der Algorithmus im Detail

Der Algorithmus benötigt als Eingabe die Position, Ausrichtung und Geschwindigkeit des jeweiligen Start- und Zielpunktes $(x_{akt}, y_{akt}, d_{akt}, v_{akt}, x_{ziel}, y_{ziel}, d_{ziel}, v_{ziel})$ sowie die zu fahrenden Radien der

zwei Kreise (r'_1, r'_2). Die Radien ließen sich im Prinzip auch aus den zu fahrenden Geschwindigkeiten ableiten. Momentan sind dafür aber noch umfangreiche Testfahrten notwendig, um eine Funktion des Radius in Abhängigkeit von Geschwindigkeit und u.U. der aktuellen Boden- und Reifenbeschaffenheit zu ermitteln. Siehe dazu auch Abschnitt 5.2.6 ab Seite 146.

Hinweis: Die y-Werte werden jeweils negiert, um der umgekehrten positiv-Richtung der y-Ordinaten im robotsoccer-Progamm Rechnung zu tragen.

Zunächst einmal wird geprüft, ob eine Tangentenanfahrt in der aktuellen Situation überhaupt schon Sinn macht. So ist es u.U. sinnvoller, einfach die Fahrtrichtung zu wechseln, anstatt erst einen Halbkreis zu fahren. Hierfür sei auf [PG362, S. 18] verwiesen, wo das Verfahren bereits beschrieben wurde.

Als nächstes müssen die Mittelpunkte der zwei zu fahrenden Kreise ermittelt werden. Am Start- und am Zielpunkt gibt es dafür jeweils zwei Kandidaten: einen Kreis rechts der aktuellen Ausrichtung und einen Kreis links der aktuellen Ausrichtung (siehe Abb. 5.1).

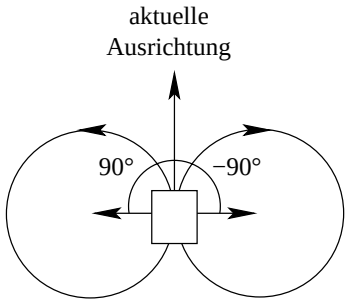


Abbildung 5.1: Kreiskandidaten

Die Mittelpunkte befinden sich jeweils im rechten Winkel zur aktuellen Ausrichtung bzw. der Zielausrichtung und in einem Abstand von r'_1 zum Startpunkt sowie r'_2 zum Zielpunkt.

Als nächstes müssen jetzt die zwei von den vier Kreisen ausgewählt werden, die den optimalen Weg vom Start- zum Zielpunkt beschreiben. Dieses wird derzeit durch schlichtes Ausprobieren aller vier Möglichkeiten ausgeführt, es müßte aber auch einen eleganteren Weg geben.

Es gibt hier bereits verschiedene Ansätze, die jedoch alle in einem mehr oder weniger großen Bereich aller möglichen Fälle zu nicht optimalen Lösungen führen. Weil hier eine nicht optimal berechnete Lösung zu einem längeren Pfad führt, deren Abfahrt in ungünstigen Fällen zu einem Zeitverlust in der Größenordnung von Sekunden führt (in einer Sekunde könnte sich z.B. der Ball schon um bis zu ca. 4 m weiterbewegt haben), fällt die relativ lange Rechenzeit für den Brute-Force Ansatz nicht so stark ins Gewicht, denn diese Berechnung dauert höchstens ein paar hundertstel Sekunden.

Für eine eventuelle Weiterentwicklung der Nicht-Brute-Force Ansätze sind im Abschnitt 5.2.6 auf Seite 146 die bereits untersuchten Kreisauswahlverfahren mit ihren erkannten Problemen aufgeführt.

Eine weitere Problematik gibt es bei der Kreisauswahl aufgrund der Banden zu beachten. Diese wird im Abschnitt 5.2.3.2 auf Seite 142 behandelt.

Sind also erst einmal die Kreise ausgewählt, so lassen sich zwischen den Kreisen (wie in Abb. 5.2 zu sehen) vier unterschiedliche Tangenten ziehen.

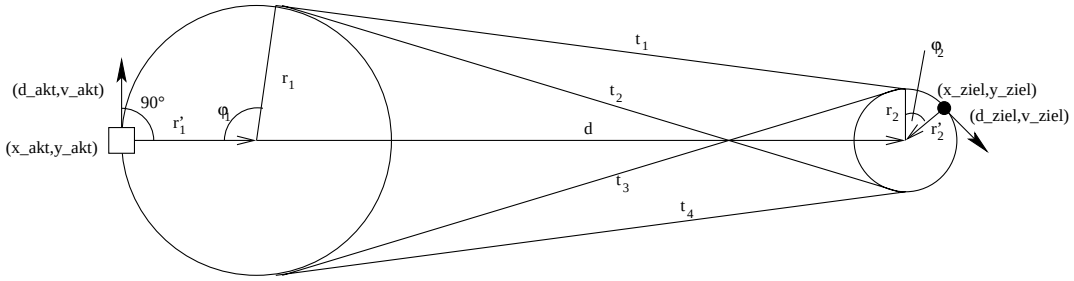


Abbildung 5.2: Tangentenauswahl

Die Auswahl der richtigen Tangente ergibt sich aus den Kreisorientierungen: der Kreis rechts vom Roboter unter Berücksichtigung der Roboterausrichtung ist gegen den Uhrzeigersinn gerichtet, der Kreis links der Ausrichtung des Roboters entsprechend mit dem Uhrzeigersinn.

Sind die Kreisorientierungen von Start- und Zielkreis gleichgerichtet, so kommen die Tangenten t_1 oder t_4 in Betracht, bei ungleicher Orientierung die Tangenten t_2 bzw. t_3.

Wenn man die Tangenten als gerichtete Strecken betrachtet, die am Berührungspunkt mit dem Startkreis beginnt und am Berührungspunkt mit dem Zielkreis endet, so ist diejenige die richtige Tangente, deren Richtung der Richtung der Kreise an den jeweiligen Berührungspunkten entspricht.

Es können hier Sonderfälle auftreten, die im Abschnitt 5.2.2.3 ab Seite 139 behandelt werden.

$$\text{Sei } \vec{d} = \begin{pmatrix} x_{ziel} \\ y_{ziel} \end{pmatrix} - \begin{pmatrix} x_{akt} \\ y_{akt} \end{pmatrix}$$

$\vec{r_1'}$ bzw. $\vec{r_2'}$ ergeben sich aus den übergebenen Beträgen für die Radien, den Ausrichtungen d_{akt} und d_{ziel}, sowie den zu Ausrichtungen der zu fahrenden Kreise. Betrachtet man hier die Vektoren mal wie Komplexe Zahlen in der Gauß-Ebene, so ergeben sich $\vec{r_1'}$ und $\vec{r_2'}$ als:

$$\vec{r_1'} = \begin{cases} r_1' \cdot e^{i\cdot\left(d_{akt}+\frac{\pi}{2}\right)} & \text{falls Startkreis linksdrehend} \\ r_1' \cdot e^{i\cdot\left(d_{akt}-\frac{\pi}{2}\right)} & \text{sonst} \end{cases}$$

$$\vec{r_2'} = \begin{cases} r_2' \cdot e^{i\cdot\left(d_{ziel}+\frac{\pi}{2}\right)} & \text{falls Zielkreis linksdrehend} \\ r_2' \cdot e^{i\cdot\left(d_{ziel}-\frac{\pi}{2}\right)} & \text{sonst} \end{cases}$$

Als nächstes werden die Winkel zwischen dem Vektor, der beide Kreismittelpunkte verbindet ($\vec{d}$), und den Vektoren, die die Kreismittelpunkte mit dem zugehörigen Tangentenschnittpunkt ($\vec{r_1}$ bzw. $\vec{r_2}$ im Abbildung 5.2 und Abbildung 5.3) verbindet, ermittelt. Der Betrag von r_1 und r_2 ist ja der selbe wie von r_1' bzw. r_2'.

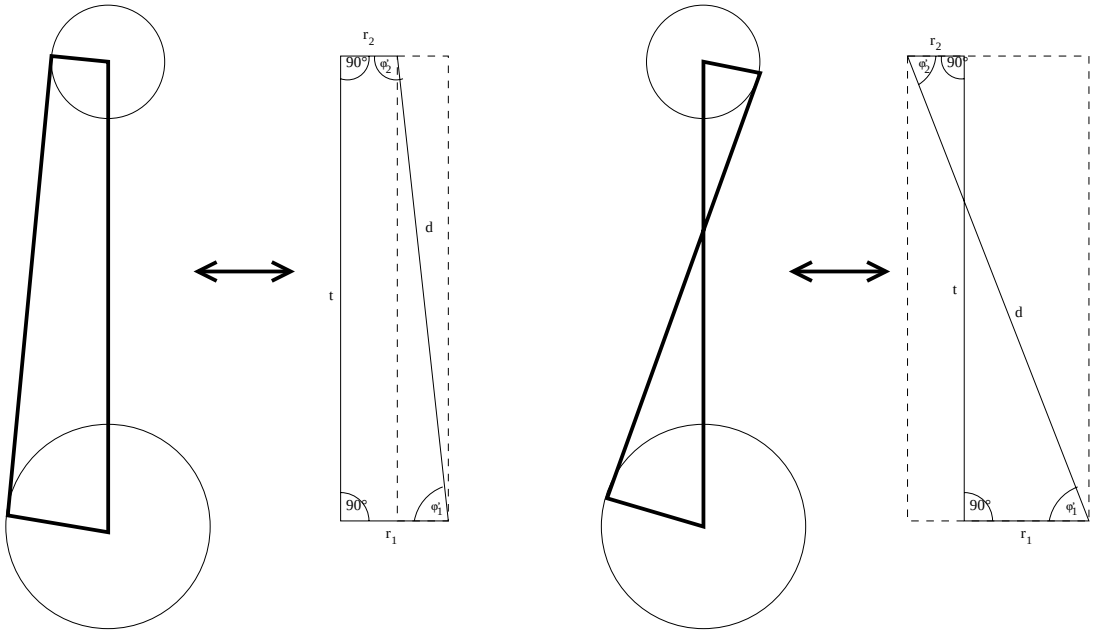


Abbildung 5.3: Winkelberechnung

Um letztenendes die eigentlichen Winkel für die Kreisfahrten zu erhalten, sind noch einige weitere Zwischenschritte notwendig. Einen ersten Eindruck, wie die Winkel untereinander zusammenhängen, läßt sich in Abbildung 5.3 gewinnen. Es gibt zwei Fälle: einen Fall, in dem die Tangente die Verbindungslinie der Kreismittelpunkte kreuzt, und einen Fall, in dem sie sich nicht kreuzen. Man beachte, daß das Vorhandensein einer Kreuzung direkt aus entgegengesetzten Kreisorientierungen abzuleiten ist.

In beiden Fällen kann das entstandene Gebilde wie in Abbildung 5.3 verdeutlicht zu einem Recht-

eck ergänzt werden (gestrichelte Linie). Es ergeben sich somit folgende Formeln zur Berechnung der Winkel:

Falls die *Kreisorientierungen der beiden Kreise gleich* sind ist:

$$\varphi_1' = \begin{cases} \arccos\left(\frac{|\vec{r_1}|-|\vec{r_2}|}{|\vec{d}|}\right) & \text{falls Startkreis rechtsdrehend und } |\vec{r_1}| \geq |\vec{r_2}| \\ -\arccos\left(\frac{|\vec{r_1}|-|\vec{r_2}|}{|\vec{d}|}\right) & \text{falls Startkreis linksdrehend und } |\vec{r_1}| \geq |\vec{r_2}| \\ -\pi + \arccos\left(\frac{||\vec{r_1}|-|\vec{r_2}||}{|\vec{d}|}\right) & \text{falls Startkreis rechtsdrehend und } |\vec{r_1}| < |\vec{r_2}| \\ \pi - \arccos\left(\frac{||\vec{r_1}|-|\vec{r_2}||}{|\vec{d}|}\right) & \text{falls Startkreis linksdrehend und } |\vec{r_1}| < |\vec{r_2}| \end{cases}$$

und für die Seite am Zielkreis gilt:

$$\varphi_2' = \begin{cases} \pi + \varphi_1' & \text{falls } |\vec{r_1}| \geq |\vec{r_2}| \\ -\arccos\left(\frac{|\vec{r_1}|+|\vec{r_2}|}{|\vec{d}|}\right) & \text{falls Zielkreis rechtsdrehend und } |\vec{r_1}| < |\vec{r_2}| \\ \arccos\left(\frac{|\vec{r_1}|+|\vec{r_2}|}{|\vec{d}|}\right) & \text{falls Zielkreis linksdrehend und } |\vec{r_1}| < |\vec{r_2}| \end{cases}$$

Die Länge der Tangentenstrecke ergibt sich mit dem Satz des Pythagoras als:

$$t = \sqrt{\left|\vec{d}\right|^2 - \left(|\vec{r_1}| - |\vec{r_2}|\right)^2}$$

Falls die *Kreisorientierungen ungleich sind*, so gilt:

$$\varphi_1' = \begin{cases} \arccos\left(\frac{|\vec{r_1}|+|\vec{r_2}|}{|\vec{d}|}\right) & \text{falls Startkreis rechtsdrehend} \\ -\arccos\left(\frac{|\vec{r_1}|+|\vec{r_2}|}{|\vec{d}|}\right) & \text{sonst} \end{cases}$$

und am Zielkreis einfach:

$$\varphi_2' = \varphi_1'$$

Die Länge der Tangentenstrecke ergibt sich diesmal als:

$$t = \sqrt{\left|\vec{d}\right|^2 - \left(|\vec{r_1}| + |\vec{r_2}|\right)^2}$$

Nun ist φ_1' bzw. φ_2' noch nicht der eigentlich benötigte Winkel φ_1 bzw. φ_2. Mit Hilfe von Abbildung 5.4 läßt sich jedoch die gesuchte Formel zur Umrechnung von φ_1' in φ_1 leicht ermitteln:

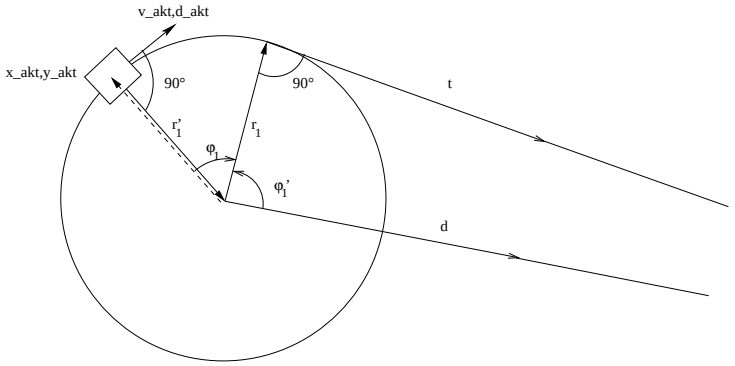


Abbildung 5.4: Winkel am Startkreis

Es wird jetzt über die Umwandlung der relativen Winkel in Absolutwinkel phi_1 bestimmt:

$$\varphi_1 = \varphi_1' + \arg(\vec{d}) - \left(\arg(\vec{r_1'}) + \pi\right)$$

Am Zielkreis wird das ganze nochmal analog durchgeführt, wobei aber darauf zu achten ist, daß $\vec{d}$ jetzt nicht vom Kreismittelpunkt weg zeigt, sondern auf ihn zu, d.h. $\vec{d}$ wird vorher umgedreht.

Abschließend werden φ_1 und φ_2 in einen äquivalenten Winkel im Intervall $]-2\pi, 0]$ für eine Rechtskurve bzw. $[0, 2\pi[$ für eine Linkskurve transformiert.

Als Ergebnis liefert der Algorithmus nun ein Tripel $(\varphi_1, t, \varphi_2)$ welches mit φ_1 den abzufahrenden Winkel am Startkreis, mit t die Länge der zu fahrenden Geraden und schließlich mit φ_2 den abzufahrenden Winkel am Zielkreis beschreibt.

Nun kann dieses Tripel in entsprechende Befehle für die Motoren umgesetzt werden. Für diesen Zweck sind die folgenden zwei Punkte sehr hilfreich:

5.2.2.1 Radgeschwindigkeiten im Kreis

Es soll ein Kreis mit einem Radius von r_m gefahren werden. Zur weiteren Berechnung werden nun die maximale Geschwindigkeit des äußeren Rades v_a gewählt und der mittlere Radabstand a gemessen. Die Innenradgeschwindigkeit v_i ergibt sich dann einfach aus:

$$r_a \quad := \quad r_m + \frac{a}{2}$$

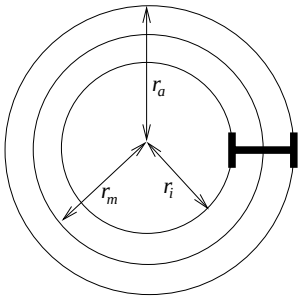


Abbildung 5.5: Berechnung der Radgeschwindigkeiten bei einer Kreisfahrt

$$
\begin{aligned}
r_i &:= r_m - \frac{a}{2} \\
U_a &= 2\pi r_a \\
U_i &= 2\pi r_i \\
\frac{v_i}{v_a} &= \frac{U_i}{U_a} = \frac{r_m - \frac{a}{2}}{r_m + \frac{a}{2}} \\
v_i &= \frac{r_m - \frac{a}{2}}{r_m + \frac{a}{2}} v_a
\end{aligned}
$$

5.2.2.2 Überstrichener Winkel bei einer Kreisbogenfahrt

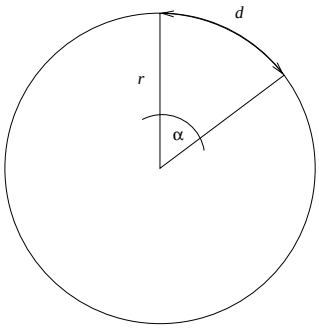


Abbildung 5.6: Berechnung eines Kreisbogens

Wird ein Kreis nach der o.g. Formel gefahren, so läßt sich der überstrichene Winkel aus folgenden zwei Formeln herleiten:

$$U = 2\pi r$$

$$\frac{U}{2\pi} = \frac{d}{\alpha}$$

Wobei r der tatsächlich von dem jeweiligen Rad gefahrene Radius (also für das äußere Rad $r_m + \frac{a}{2}$

und für das innere Rad $r_m - \frac{a}{2}$), und d die von diesem Rad gefahrene Strecke ist.

Hat man das Verhältnis von Ticks pro Radumdrehung t_r, den Radumfang U sowie den gefahrenen Radius r und die gefahrenen Ticks t gegeben, so kann der überstrichene Winkel wie folgt berechnet werden:

$$\begin{aligned} \frac{2\pi r}{2\pi} &= \frac{t \cdot \frac{U}{t_r}}{\alpha} \\ \Rightarrow \alpha &= \frac{t}{r} \cdot \frac{U}{t_r} \end{aligned}$$

Bei den Dortmunder Robotern ist ein Verhältnis von 512 Ticks pro Radumdrehung bei der alten Baureihe (vor 2002) und 20460 Ticks pro Radumdrehung bei der neuen Baureihe gegeben. Der Radumfang ist jeweils 4 cm.

D.h. $\frac{U}{t_r}$ ist für die alte Baureihe gleich

$$\frac{4 \text{ cm}}{512 \text{ Ticks}} = 7,8125 \cdot 10^{-3} \frac{\text{cm}}{\text{Tick}}$$

Für die neue Baureihe gilt dementsprechend:

$$\frac{4 \text{ cm}}{20460 \text{ Ticks}} \approx 1,9550 \cdot 10^{-4} \frac{\text{cm}}{\text{Tick}}$$

Es kann nun anhand der hier berechneten Radgeschwindigkeiten und Winkel der Roboter dazu veranlaßt werden, die vorher berechnete Route abzufahren. Es muß nur immer mal wieder ein Vergleich der bisher abgefahrenen Winkel mit den zu fahrenden Winkeln gezogen werden. Ist z.B. φ_1 nahezu abgefahren (einige Ungenauigkeiten kommen alleine schon durch das idealisierte mathematische Modell dieses Algorithmus zustande), wird einfach in die gerade Tangente abgebogen, indem die Radgeschwindigkeiten der beiden Roboterräder auf den gleichen Wert gesetzt werden. Auch hier wird wieder verglichen, ob die gefahrene Strecke auf der Tangente die Länge von s_t erreicht hat. Nachdem diese Methode auch für φ_2 angewandt wurde, ist der Roboter an seinem Ziel angekommen.

5.2.2.3 Sonderfälle

Der erste der beiden zu betrachtenden Sonderfälle stellt einen Schnitt der beiden Kreise in einer Überkreuzfahrt dar, wie sie in Abbildung 5.7 dargestellt ist. Es existiert hier keine Tangente, die angelegt werden könnte. Dieser Fall tritt z.B. auf, wenn die Startposition relativ nahe am Zielpunkt

liegt oder der Roboter während der Fahrt auf einer vorher berechneten Anfahrt leicht vom Kurs abkommt.

Derzeit besteht die Lösungsstrategie in diesem Fall lediglich darin, in der aktuellen Ausrichtung geradeaus weiter zu fahren, bis Start- und Zielposition wieder weit genug auseinander liegen, so daß eine Tangente angelegt werden kann.

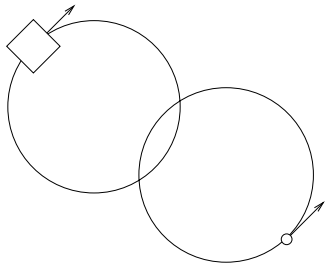


Abbildung 5.7: Sonderfall: Kreise überschneiden sich bei einer Überkreuzfahrt

Der zweite Sonderfall tritt auf, wenn – wie in Abbildung 5.8 – einer der beiden Kreise vollständig im anderen enthalten ist und keine Berührungspunkte existieren. Auch in diesem Fall existiert keine Tangente. Auftreten kann dieser Fall vor allem, wenn sich der Roboter auf dem Zielkreis befindet und hier etwas vom Kurs abkommt und/oder die Zielgeschwindigkeit noch nicht ganz erreicht hat. Der derzeitige Lösungsansatz ignoriert Abweichungen bis zu einem gewissen Grad und fährt so, als sei der Zielkreis erreicht und perfekt getroffen worden. Sollten jedoch in diesem Fall die Geschwindigkeiten unverhältnismäßig auseinanderliegen, ist eher von einer ungünstigen Start-/Zielpositions-Kombination auszugehen, und der Algorithmus muß die Geschwindigkeiten geeignet anpassen.

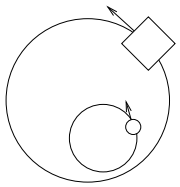


Abbildung 5.8: Sonderfall: Ein Kreis ist vollständig im anderen enthalten

5.2.3 Hinderniserkennung und -meidung

5.2.3.1 Roboter

Auf einem Spielfeld mit zwei Mannschaften von jeweils fünf Robotern ist es recht wahrscheinlich, daß Kollisionen mit anderen Robotern auftreten, wenn nicht gesonderte Maßnahmen ergriffen werden, die eben dieses vermeiden. Es gibt hier zwei Unterprobleme:

- Hinderniserkennung
- Hindernisumgehung

Für die Hinderniserkennung gibt es zwei grundsätzliche Ansätze:

Als erstes: Prüfen eines jeden Objekts, ob es mit dem vorausberechneten Pfad kollidieren kann. Diese Methode hat den entscheidenden Nachteil, daß dies einen hohen Rechenaufwand bedeutet. Mit zunehmendem Abstand der Objekte zur Position des Roboters, der gerade betrachtet wird, wird auch die Zeitspanne immer größer, die der Roboter braucht, um dieses Hindernis zu erreichen. Somit sinkt jedoch auch mit zunehmender Entfernung die Wahrscheinlichkeit, daß ein Objekt, was in einer vorherigen Berechnung noch als Hindernis klassifiziert wurde, zu dem Zeitpunkt, zu dem der Roboter eine Ausweichbewegung einleiten müßte, sich noch an dem selben Ort befindet. Die vormals teuer bezahlte Berechnung ist somit wertlos geworden. Dieser Ansatz ist somit nicht zu empfehlen.

Eine Alternative dazu bietet die Kollisionserkennung, wie sie auch schon im CMU-Algorithmus angewendet wurde (siehe [PG362, S. 16ff]). Hierbei wird einfach nur in einem festgelegten Radius um die aktuelle Position des Roboters herum nach Hindernissen gesucht. Dies ist relativ schnell, und die Kollisionswahrscheinlichkeit ist schon bedeutend höher als bei entfernteren Objekten. Nachteil ist allerdings, daß erst recht spät auf ein Hindernis reagiert werden kann.

Ist ein Hindernis erst einmal erkannt, gibt es wiederum zwei Ansätze, wie es mit Hilfe dieses Algorithmus umfahren werden kann:

Eine Möglichkeit besteht darin, um das Hindernis einen Kreis zu legen, und wie für das Ziel an diesen Kreis eine Tangente zu legen. Also kurzgefaßt einen Zwischen-Wegpunkt auf einem Kreis um das Hindernis herum zu legen, der wie ein Zielpunkt angefahren werden kann. Wenn dieser Zwischen-Wegpunkt erreicht ist, kann der alte Zielpunkt wieder angefahren werden. Abb. 5.9 verdeutlicht das Vorgehen.

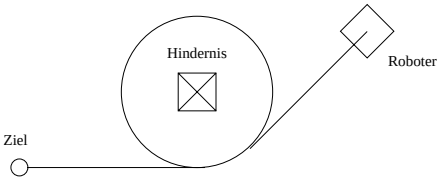


Abbildung 5.9: Hindernis im Mittelpunkt des Umgehungskreises

Die Alternative stammt aus dem schon eingangs erwähnten Abstract: [Mes98]: Es wird nicht der Kreis direkt um das Hindernis gelegt, sondern es wird ein Kreis wie in Abb. 5.10 zu sehen ist *an* das Hindernis gelegt.

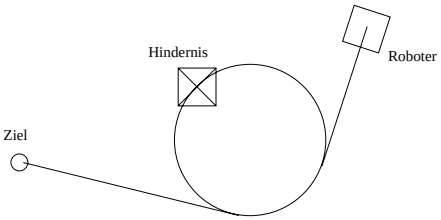


Abbildung 5.10: Hindernis auf dem Umgehungskreis

5.2.3.2 Banden

Ein Hindernis ganz anderer Art stellen die Banden dar. Ihre Position ist entgegen denen der Roboter immer konstant. Wie genau die Banden vermieden werden können, ist bisher noch nicht geklärt. Der generelle Ansatz besteht jedoch darin, eine mögliche Kollision mit einer Bande anhand der Abstände von Start- und Zielposition zu den Banden zu erkennen und auf geeignete Weise, d.h. z.B. durch Modifizieren der Geschwindigkeiten und damit auch der Radien, zu vermeiden.

Sollte hier ein brauchbarer Ansatz entwickelt werden, so könnte dieser eventuell auch zur Strafraumvermeidung Verwendung finden.

5.2.4 Aufteilung der Programmteile auf das DSP- und Hostsystem

Um diesen Algorithmus zu implementieren, mußten einige Überlegungen in Bezug auf die Aufteilung der Programmteile auf die Host- und die DSP-Seite vorgenommen werden. Während es für die Hostseite kein Problem darstellt, die Absolutpositionen aller Roboter festzustellen (wichtig z.B. für

Kollisionsvermeidung), hat der DSP bestenfalls die Möglichkeit, im Rahmen seiner Radencoder die tatsächliche Ausrichtung und relative Position zu schätzen: eine mögliche Seitwärtsbewegung (wie sie z.B. auftritt, wenn der Roboter von einem anderen seitlich weggeschoben wird) kann der DSP nicht feststellen. Demgegenüber hat der DSP jedoch im Rahmen seiner Sensorik die Möglichkeit, einzelne Radgeschwindigkeiten wesentlich präziser regeln zu können, als es dem Hostsystem mit seiner recht ungenauen Bilderkennung und der relativ hohen Latenzzeit je möglich wäre.

Somit sieht eine vernünftige Aufteilung in etwa so aus:

Host: generelle Pfadplanung mit Hindernisauswertung

DSP: Umsetzung der vom Hostsystem berechneten Ergebnisse in eine Regelung der Radgeschwindigkeiten

Nun ist ursprünglich die Vergrößerung des Autonomiegrades der Roboter als Ziel dieser PG gesetzt worden. Um nicht vollends diesen Gedanken abzuschreiben, ist jeder Teil dieses Algorithmus so implementiert worden, daß er ohne große Anpassungen sowohl auf dem Hostsystem, als auch auf dem DSP laufen könnte, sobald die oben angesprochenen zugrundeliegenden Probleme gelöst wurden.

Um nun nicht für diese zwei Systeme zwei separate Dateien für die Implementierung des Algorithmus warten zu müssen, wurde im Quellcode reichlich von Compilerschaltern Gebrauch gemacht. Im wesentlichen handelt es sich dabei um systemspezifische Anweisungen, wie auf einzelne benötigte Werte zugegriffen werden kann. Um diese Technik anwenden zu können, muß für die Kompilation der Sourcen der Kompilerschalter „ROBOTSOCCER“ für das Host-System und „DSP“ für die Roboter gesetzt sein. Außerdem muß die Dateiendung für das Hostsystem „.cpp“ und die für das Roboter-System „.c“ lauten. Die Datei „AnfahrtKreisfahrt.c*“ ist also geeignet umzubenennen.

Es muß sichergestellt sein, daß die Dateien „AnfahrtKreisfahrt.cpp“ und „AnfahrtKreisfahrt.c“ immer den gleichen Inhalt haben, um die oben beschriebene systemübergreifende Funktionalität der Sourcen nicht zu gefährden.

5.2.4.1 vector2d

Zu dem eigentlichen Algorithmus wurde auch noch eine Bibliothek von Vektor-Funktionen implementiert. Eine wesentliche Hilfestellung gab dabei [Gla93, Seite 195ff]. Zwar exisitert schon eine Klasse mit gleicher Funktionalität in „Vektor2D.cpp“, jedoch ist diese nur für C++ konzipiert. Für die DSPs wird jedoch eine Implementation in C benötigt. Kurzerhand wurden die benötigten Funktionen in der Datei „vector2d.c“ bzw. „vector2d.cpp“ implementiert.

5.2.4.2 Die Testumgebung

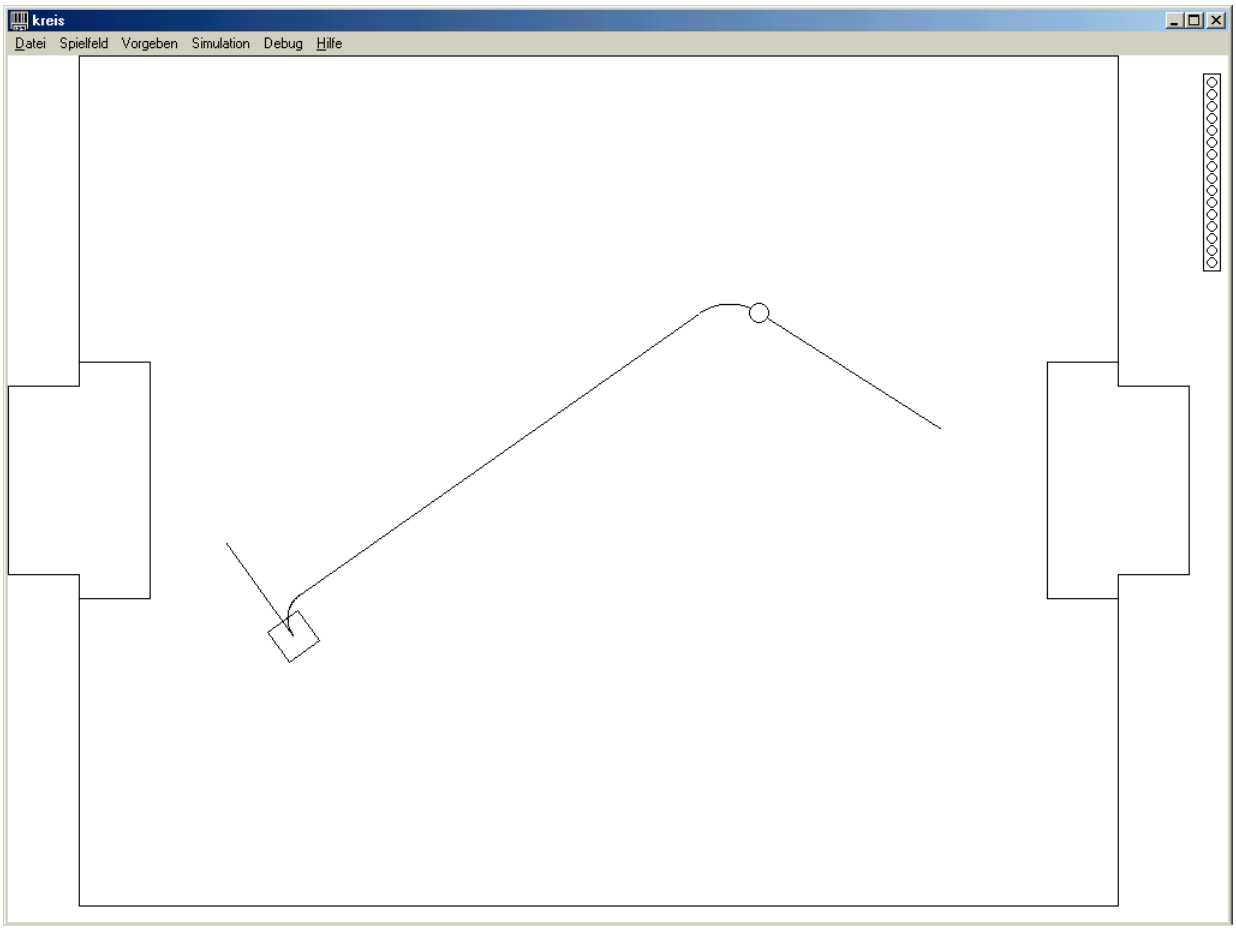


Abbildung 5.11: Testumgebung

Die Testumgebung wurde parallel zur Entwicklung des Algorithmus geschrieben. Sie bietet die Möglichkeit, die Funktionalität des Algorithmus in verschiedenen Situationen zu testen. Der Roboter (dargestellt als Quadrat) wird mittels der linken Maustaste auf dem virtuellen Spielfeld plaziert und die Ausrichtung und Geschwindigkeit mit festgehaltener SHIFT-Taste verändert. Die Zielposition (dargestellt als kleiner Kreis), -geschwindigkeit und -ausrichtung werden jeweils analog mit der rechten Maustaste gesetzt. Die Linie zwischen Start und Endpunkt ist der berechnete Pfad.

In der Menüleiste ist eine Auswahl an Optionen gegeben. So kann zwischen den beiden Spielfeldgrößen gewechselt, eine Richtungsumkehr-Behandlung zugeschaltet, die zu fahrenden Kreise vorher schon festgelegt (d.h. Kreisauswahl wird nicht verwendet), eine einfache Bewegungssimulation gestartet und dort auch mit einer Unschärfe in Ausrichtung und Position belegt werden (zum Testen der Stabilität gegen Ungenauigkeiten in der BV im Robosoccer-System usw.). Auch eine Ausgabe der im

Programmcode gegebenen Debug-Meldungen kann initiiert werden. Die Ausgaben werden in die Datei „PfadOut.txt“ gescrieben.

Zu erwähnen ist noch die Statusleiste am rechten Bildschirmrand. Hier wird der Wert der Status-Variablen im Quellcode binär ausgegeben, LSB ist oben, MSB entsprechend unten. Hier kann das Auftreten von fehlerhaften Berechnungen und Sonderfällen überwacht werden. Die Proportionen von Spielfeld und Roboter sind eingehalten. Die Zielposition hat die Größe des Balls, ist jedoch nicht mit ihm gleichzusetzen (Eine Zielposition kann auch unabhängig vom Ball gesetzt werden).

Zwei Anmerkungen zur Testumgebung sind noch zu machen: Die Testumgebung ist nicht besonders gut geschrieben. D.h. in der Projektgruppe verstand sich niemand im Umgang mit Visual C++. Die Konsequenz ist, daß viele Elemente der Testumgebung recht umständlich und evtl. einige sogar fehlerhaft sind. Außerdem wird auf Dauer eine Testumgebung unter Linux gegenüber einer unter Windows vorzuziehen sein, weil das robotsoccer-Projekt ebenfalls unter Linux läuft.

5.2.5 Stand der Umsetzung

Bis zum Zeitpunkt dieser Dokumentation sind nur einige Elemente des Anfahrtsalgorithmus fertig implementiert. Wesentliche Elemente stehen noch aus, und einige weitere benötigen noch eine brauchbare Konzeption.

Prinzipiell funktioniert bisher nur die reine Anfahrtsplanung – mit folgenden Einschränkungen:

- Die Bestimmung der fahrbaren Kreisradien bei einer gegebenen Geschwindigkeit ist aufgrund fehlender Tests noch nicht abgeschlossen.
- Eine Banden- und Hindernisvermeidung ist bisher nicht verwirklicht.
- An verschiedenen Stellen muß noch eine Behandlung von „unscharfen Kriterien“ (z.B. fast erreichte Zielposition, etwas zu weit in den Kreis hineingefahren usw.) umgesetzt werden.
- Viele der Programmteile sind noch nicht ausreichend getestet, insbesondere deshalb, weil einige von ihnen erst in der „realen Welt“ richtig zu testen sind.
- Viele der Steuerungs- und Regelungsabläufe beinhalten Schwellwerte, die bisher nur auf weitestgehend willkürlich festgelegten Werten beruhen.
- Der Algorithmus funktioniert z.Z. im Simulator ohne größere Schwächen zu zeigen, jedoch vollführen die realen Roboter sehr seltsame Bewegungen. So gibt es Situationen wo der Roboter

nur noch Halb-Drehungen auf der Stelle vollführt, dann kurze Zeit innehält und dann in die entgegengesetzte Richtung die Halbdrehung zurückdreht. Wenn der Roboter sich überhaupt in eine ersichtliche Richtung fortbewegt, so nur in einer Art „Tänzelbewegung“ in die immer mal wieder eine Halbdrehung auf der Stelle eingebaut wird.

Es scheinen also einige Fehler (oder nicht bedachte Probleme) in den Sourcen noch zu korrigieren sein.

5.2.6 Erweiterungs- und Verbesserungsmöglichkeiten

Aus den eben genannten Problemen ergeben sich zwangsläufig die weiteren Schritte, die noch zum fertigen Anfahrtsalgorithmus fehlen. Neben diesen Problemen, gibt es aber auch noch weitere Dinge, mit denen der Anfahrtsalgorithmus noch verbessert werden kann:

Zunächst wäre da der schon weiter oben angesprochene Radius. Er wird durch die Formel:

$$radius = \frac{Speed}{\sqrt{\frac{F_{ZentMax}}{m}}}$$

berechnet, wobei m die Masse eines Roboters (durchschnittlich 450g) und $F_{ZentMax}$ die aufgrund der Boden- und Reifenbeschaffenheit maximal zulässige Zentrifugalkraft bezeichnet. Diese unterscheidet sich bei jedem Spielfeld. Es ist nun notwendig ein Testverfahren zu entwickeln, welches z.B. mit einer kurzen Testfahrt auf einem unbekannten Spielfeld selbstständig die zulässige maximale Zentrifugalkraft ermittelt. Alternativ kann auch mit einer Federwaage dieser Wert bestimmt werden. Es könnte allerdings recht komisch wirken, wenn jemand mit einer Federwaage kurz vor einem Turnier erst das Spielfeld ausmißt. Außerdem verändert sich die Oberfläche eines Spielfeldes und vor allem die Beschaffenheit der Reifen während eines Spiels z.B. durch Staub. Es währe hier von Vorteil, während einer Spielunterbrechung nochmals einen kurzen Testlauf starten zu können.

Ein weiterer Punkt ist die Kreisauswahl. In den folgenden Diagrammen kann beispielhaft eine Problem-Situation anhand des Beinahe-Kreises am Startpunkt erkannt werden. In einigen Fällen kann es auch zu ähnlichen Gebilden am Zielpunkt kommen.

Bisher wurden die folgenden Heuristiken auprobiert:

1. Halbebenen

   Die Ebene wird entlang der Ausrichtung des Roboters bzw. der Zielausrichtung in zwei Halbebenen unterteilt. Für den Startpunkt wird derjenige Kreis gewählt, in dessen Halbebene auch der Zielpunkt liegt. Für den Zielkreis gilt dies jeweils analog.

Probleme gibt es hier in den Fällen, wo die Kreisradien unterschiedlich groß sind und der Start- oder Zielpunkt relativ nah an der Teilungslinie liegt, was aber recht häufig vorkommt (Siehe Abbildung 5.12).

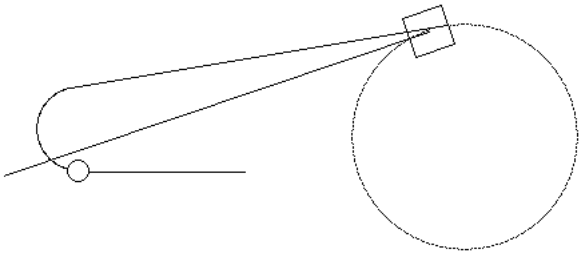


Abbildung 5.12: Problematik Halbebenen

2. Distanz der Mittelpunkte zwischen den Start- und Zielkreisen

   Es werden jeweils die Distanzen zwischen dem Mittelpunkt eines Startkreiskandidaten und dem eines Zielkreiskanditaten errechnet. Die kürzeste Distanz gewinnt.

   Probleme gibt es hier wenn die Startgeschwindigkeit von der Zielgeschwindigkeit abweicht und z.B. die Positionen fast auf einer Linie mit annähernd gleichen Ausrichtungen liegen (Siehe Abb. 5.13).

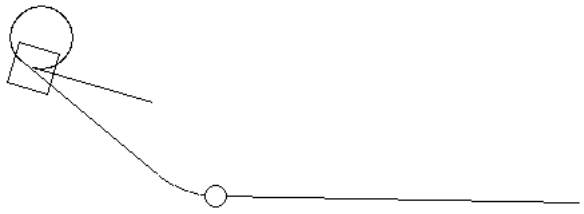


Abbildung 5.13: Problematik Mittelpunktdistanzen

3. Distanz zwischen einem Kreismittelpunkt und dem Ziel- bzw. Startpunkt

   Von den Startkreiskandidaten werden jeweils die Distanzen zwischen ihren Mittelpunkten und dem Zielpunkt verglichen. Die kürzere Distanz gewinnt. Für die Zielkreiskandidaten wird das gleiche nochmal mit dem Startpunkt durchgeführt.

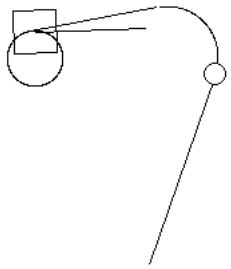


Abbildung 5.14: Problematik Distanz Kreismittelpunkte zu Start-/Zielpunkt

Probleme bereitet dieser Ansatz z.B. bei einer Fahrt wie in Abb. 5.14 beschrieben

4. Brute-Force

   Hier werden einfach alle Kombinationen von Start- und Zielkreisen durchprobiert. Diejenige mit der kürzesten Pfadgesamtlänge wird ausgewählt.

   Das Problem ist hier natürlich die etwa viermal so lange Berechnungszeit als bei den anderen Ansätzen, dafür liefert sie immer einen optimalen Pfad.

5.3 Roboter Mechanik

Neben den Softwarekomponenten wurden auch die Roboter selbst auf Verbesserungen hin überprüft. Bevor mit derartigen Erweiterungen allerdings begonnen werden konnte, wurden zunächst die bereits vorhandene Hardware untersucht. Hierbei wurden, ähnlich wie bei den Tests der Strategiekomponente, Fehler festgestellt, die ebenfalls dem ungehinderten Fahren der Roboter hinderlich sind. So fiel besonders bei der neuesten Version der Roboter auf, dass das Zahnradgetriebe, das lediglich nur in der Verbindung vom Motorritzel zum Zahnkranz des Rades besteht, kein gleichmäßiges Fahrverhalten zuläßt, vielmehr zeichnet sich diese Verbindung durch ein starkes Haken aus. Der verantwortliche Konstrukteur antwortete hierauf, dass sich diese Verbindung, die aus unterschiedlichen Materialien besteht, erst einfahren müsse. Ein Uhrmacher wird für eine Uhr, die aufgrund gewisser Ungenauigkeiten in den ersten Wochen nicht richtig funktioniert wohl kaum einen Käufer finden.

Mit den Rädern verbindet sich auch ein weiteres Problem, dass noch zu beheben ist: Vor der Eigenkonstruktion dieser Felge wurde offensichtlich nicht an eine geeignete Bereifung gedacht. Derzeit werden die Reifen aus einer zwei Millimeter starken Beschichtung für Tischtennisschläger geschnitten und zu einem Ring zusammengeklebt. Das Problem hierbei ergibt sich aus der unvermeidbar dickeren Klebestelle, die in die Fahreigenschaften der Roboter eine zusätzliche Ungenauigkeit einbringen. Erste Nachforschungen bei Gummiherstellern und Vertreibern für Keilriemen o.ä. ergaben bislang kein zufriedenstellendes Ergebnis, ebenso die Suche nach passenden O-Ringen oder Dichtungen in diversen Baumärkten. Auch im Modellbaubereich wurden keine passenden Reifen gefunden, hier ergab sich aber die Information, dass der Reifen für optimalen Halt aus einem weichen Gummi und etwas *klebend* sein sollte.

Aufgrund der Größe der Motoren ist es nicht möglich ein Rad direkt anzutreiben hierdurch scheiden auch komplette Modellbauräder erst einmal aus. Die Motoren müssen nebeneinander angebracht werden, ein einfaches Getriebe ist somit bei der derzeitigen Konstruktion unumgänglich. Aufwendige Umstrukturierungen der Roboterhardware könnten die Motoren vielleicht hochkant anordnen und die Räder direkt über eine Gelenkwelle o.ä. antreiben, zunächst sollten jedoch Lösungen gefunden werden, die das Fahrverhalten schon kurzfristig verbessern. Auf längere Sicht sollte über eine andere Lösung nachgedacht werden, da diese auch die Probleme des Getriebes beheben könnte. Eine exakte Herstellung der Roboter könnte in Verbindung mit wenigen Verbesserungen der bestehenden Konstruktionszeichnungen kurzfristigt schon hilfreich sein. Bei einer Umstrukturierung der Hardware sollte zusätzlich überprüft werden, ob eine neuere Verison der “Schaufeln”, die weiter ins Roboterinnere hineinreichen und den Ball somit besser führen könnten, Platz finden würden und getestet werden,

ob sie sich als sinnvoll erweisen. Es sollte weiterhin darauf geachtet werden, dass der Schalter zur Stromversorgung in die dafür vorgesehene Aussparung passt; besser wäre noch gewesen, der Roboter wäre so konstruiert, dass ein bestimmter Schalter problemlos eingesetzt werden kann. In der älteren Version musste der Schalter schief angeschraubt werden, die neuere Version sieht einen geklebten Schalter vor, dieser kann sich im Spielverlauf jedoch lösen und fällt schließlich vorerst unerreichbar ins Roboterinnere zurück; ein ausgeschalteter Roboter wird somit für gewisse Zeit zu einem Totalausfall. Es sollte eine Schaltervariante eingesetzt werden, die möglichst stabil gegenüber Wackelkontakten ist, auch hierdurch ergaben sich in der Vergangenheit gewisse Probleme.

Weiterhin wurden Nachforschungen über mögliche neue Akkus angestellt. Die alte Roboterausführung benutzt einen Akku bestehend aus sechs AA-Zellen, die in Ausdauer und Kapazität ausreichend sind. Die neue Ausführung besitzt einen praktischen Klappmechanismus, der das Auswechseln der Akkus vereinfacht. Leider wurde diese Version auch mit einer Art Akkuschale erweitert, die für die alten Akkus etwas zu klein, für die neuen, gezwungenermaßen bestehend aus sechs kleineren AAA-Zellen, jedoch zu groß ist. Somit sind die neuen Roboter derzeit mit einem Akku augestattet, der einerseits durch seinen fehlenden Halt in der Akkuschale das Fahrverhalten negativ beeinflussen kann und andererseits wegen des geringeren Platzangebotes auch eine geringere Leistungsfähigkeit und Haltbarkeit aufweist. Recherchen im Modellbaubereich ergaben keine andere Lösung; Handy-, Notebook- oder Gel-Akkus erwiesen sich vorerst als zu kostspielig. Die Nachforschungen in diesem Bereich sollten jedoch ebenfalls noch weitergeführt werden.

Der letzte noch nicht abgeschlossene Punkt in diesem Bereich befasst sich mit dem Problem des “Hin- und Herkippens”. Da der Roboter nur zwei Räder und vorne wie auch hinten keine Führung besitzt, die ihn in einer ständig waagerechten Position hält, kann der Roboter wenige Millimeter nach vorne und hinten kippen. Ein Problem hierbei ergibt sich vorallem beim Anfahren und Bremsen, der Roboter kratzt mit einer Kante über das Spielfeld und ist dadurch in seiner gleichmäßigen Fahrt gestört. Dies bedingt auch, dass einige Roboter beispielsweise beim Anfahren zuerst eine kleine Kurve fahren. Die Lösung bestand anfangs darin, unter den Boden jeweils am vorderen und hinteren Ende etwas unterzukleben, das mit Isolierband abgeklebt als eine Art Gleitfuß wirken sollte. Die Praxis zeigte, dass das Isolierband schnell durchscheuerte, deshalb wurden der Einsatz von Möbelgleitern getestet, doch auch diese wetzten zu schnell ab und sammelten zusätzlich noch viel Staub auf. Aufgrund dieser ersten Erkenntnisse wird derzeit ein untergeklebter Gleitfuß eingesetzt, der aus einem Teil eines Klettverschlusses besteht; dadurch wird zwar immer noch Staub aufgesammelt, das Material jedoch scheuert nicht ab.

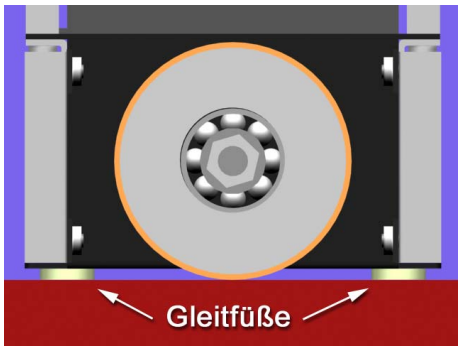


Abbildung 5.15: Roboter mit Gleitfüßen

Auf der Suche nach Alternativen wurden auch Gleitfüße aus Metall ins Auge gefasst. Ein Prototyp verwendete vier aussen an den Schaufeln angebracht Winkel, in die jeweils eine Schlossschraube oder eine Schraube mit Hutmutter eingeschraubt war. Hierdurch wurde dem Materialverschleiss vorgebeugt, die Füße sammelten keinen Staub auf und waren über das Gewinde und eine entsprechende Kontermutter auch in ihrer Höhe einstellbar. Das Kippen des Roboters konnte hiermit vermieden werden, leider wurde das Fahrverhalten durch die vier Gleitpunkte leicht verschlechtert. Das geringe Platzangebot aussen am Roboter ließ den Einsatz dieser Gleitfüße letztendlich nicht zu; die Aussenmaße des Roboters entsprechen den maximal zulässigen, so daß kein weiteres Teil nach aussen abstehen darf. Als einzige Stelle, an der eine solche Konstruktion in Miniaturausführung angebracht werden kann, bietet sich nur der Platz zwischen Rad und Schaufel an. Hutmuttern in dieser Größe sind schon wieder so klein, dass sie weniger gleiten sondern vielmehr über die Spielfeldoberfläche kratzen würden; entsprechend kleine Schlossschrauben sind nur schwer zu besorgen.

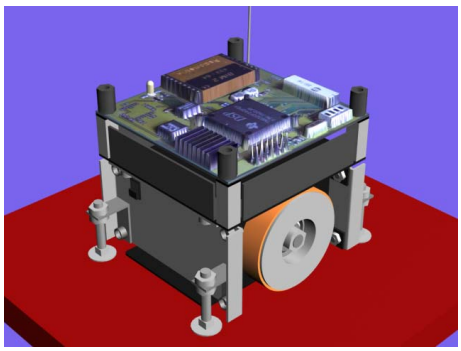


Abbildung 5.16: Prototyp Schloßschraube

Somit wurde das Interesse wieder auf eine Variante gelenkt, die nur zwei Gleitpunkte besitzt. Die

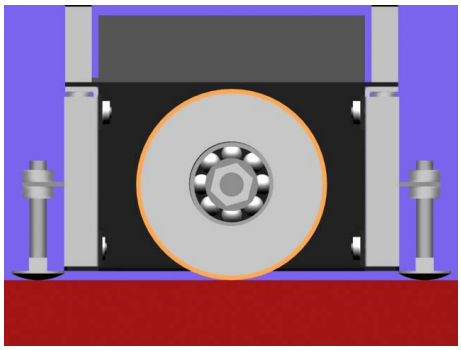


Abbildung 5.17: Platzangebot zwischen Rad und Schaufel

Anbringung muss somit zentriert und möglichst weit vorne bzw. hinten erfolgen; die einzigen möglichen Stellen sind somit einerseits unter dem Boden und andererseits im Fuß, der sich unter der Schaufel befindet. Da untergeklebte Gleitfüße schon vielfach ausprobiert wurden, ging die Suche weiter in Richtung einer in den Roboterfuß integrierten Gleitvorrichtung. Hierfür existieren allerdings nur Ideen und Überlegungen, Prototypen und Tests wurden noch nicht angestellt.

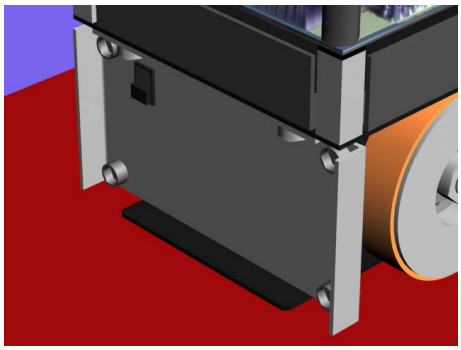


Abbildung 5.18: Roboterfuß

Der Fuß unter der Schaufel ist lediglich sechs Millimeter breit. Eine höhenverstellbare eingeschraubte Version wurde zuerst überdacht. Aufgrund des geringen Platzangebotes kommt in diesem Fall nur eine Madenschraube in Frage, der relativ spitze, wenn auch abgerundete Kopf eignet sich wohl wiederum weniger zum Gleiten und würde wohl eher Kratzer hinterlassen. Eine Madenschraube mit Kugel im Kopf und einer dahinterliegenden Feder, federndes Druckstück, genannt scheint hierfür besser geeignet zu sein. Diese Möglichkeit könnte ausprobiert werden, wenn auch federnde Druckstücke eher nicht zum Gleiten sondern zum Einrasten bei Verschiebungen eingesetzt werden; in geringen Stückzahlen sind diese Normteile auch schwer zu bekommen, ein Prototyp in einer vorgeschraubten Verlängerung

des Fußes sollte einen Test auch ohne eine Beschädigung des eigentlichen Fußes möglich machen; selbstsichernde Druckstücke machen selbst den Einsatz von Kontermuttern überflüssig.

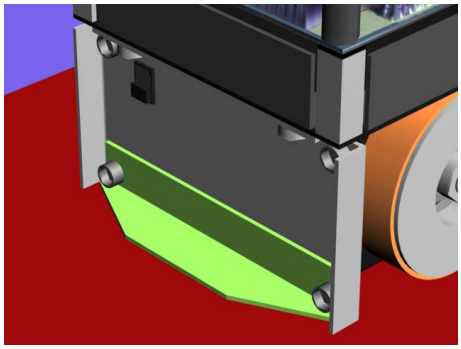


Abbildung 5.19: Roboterfuß-Testwinkel

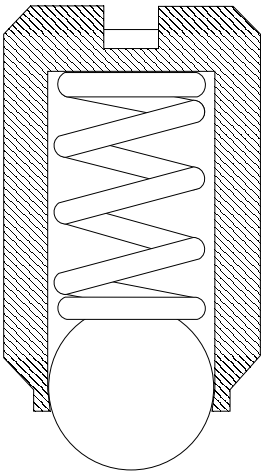


Abbildung 5.20: Federndes Druckstück

Normteile, die wirklich rollen und das auch noch in jede beliebige Richtung werden im Industriebereich Kugelrollen genannt. Ein Normteil, daß in seiner Eigenschaft wie für den Einsatz als Gleitfuß, vielmehr als Rollfuß, für dem Bereich Roboterfußball geschaffen zu sein scheint. Bisherige Nachforschungen, sogar der Einblick in spezielle Kataloge für Kugelrollen hinterließen die Erkenntnis, daß es keine Ausführungen zu geben scheint, die für die klein genung ist, um im Bereich der Fußballroboter Einsatz zu finden. Die Nachforschungen in diesem Bereich sollten jedoch weitergeführt werden, besonders bei einer möglichen Umstrukturierung der Roboterhardware sollte der Einsatz von Kugelrollen unbedingt berücksichtigt werden.

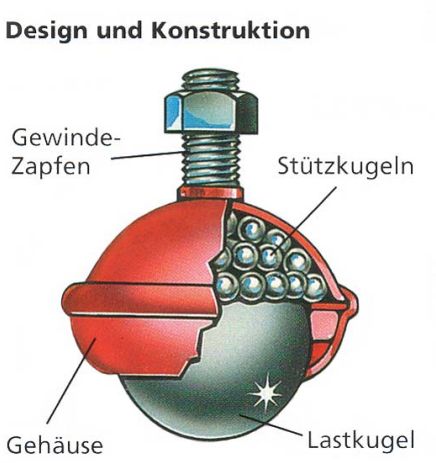


Abbildung 5.21: Kugelrolle

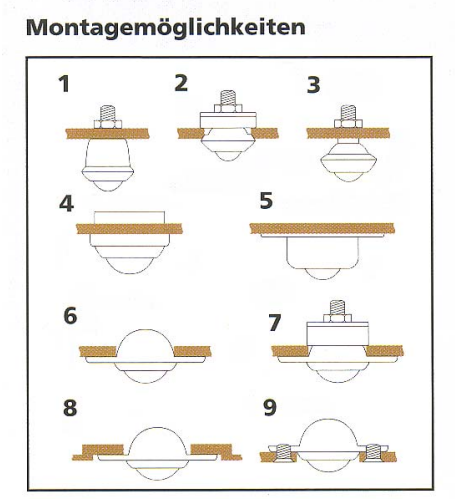


Abbildung 5.22: Montagemöglichkeiten

Als weitere Gleitvorrichtungen, vielmehr Rollvorrichtungen, die in den Fuß des Roboters zu integrieren wären, sind noch weitere Ideen zu testen. Nach Vorlage der älteren koreanischen Roboter, die im Besitz des Lehrstuhles sind, könnte eine kleine Achse mit Rolle eingesetzt werden, auch im koreanischen Team “KingGo” wird eine derartige Variante verwendet. Testfahrten der koreanischen Roboter hinterließen bei Drehungen allerdings Kratzer, da die Rollen bei dieser Bewegung seitlich über den Boden schleifen. Besser wäre an dieser Stelle vielleicht der Einsatz einer breiteren Rolle mit weniger scharfen Kanten oder sogar eine Kugel, die sich in Vor- und Rückwärtsbewegung um die angebrachte Achse dreht und bedingt durch ihre Form bei eher seitlichen Bewegungen als ein guter Gleitfuß funktionieren würde.

Die letzten vorgestellten Ideen sollten für kurzfristige Lösungen getestet werden, aber auch langfristigt, sollte eine neuere, besser gleitende Roboterversion entstanden sein, müsste für die älterern Roboter, sofern diese noch eingesetzt werden, eine derartige Gleitvorrichtung gefunden werden.

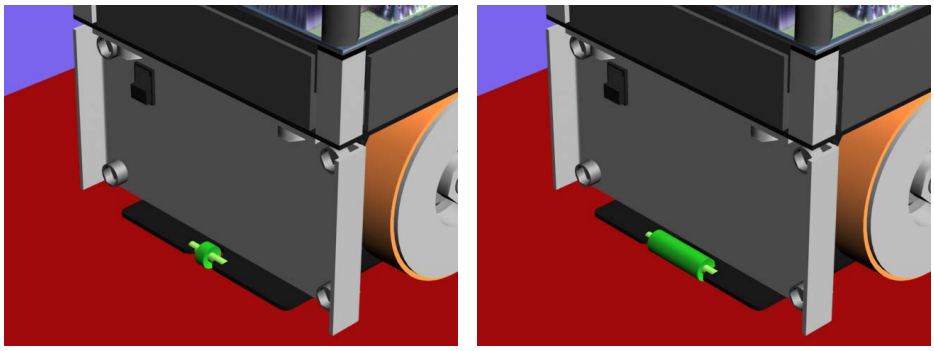


Abbildung 5.23: Gleitrolle (links) und Gleitrolle (rechts)

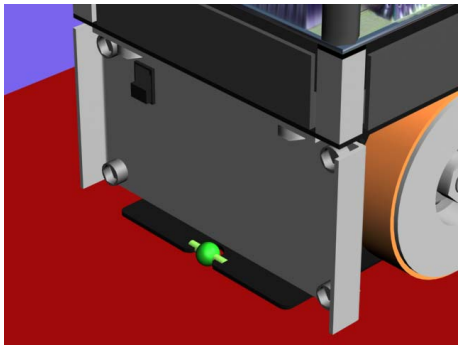


Abbildung 5.24: Gleitkugel

5.4 Systematische Farbeinstellung mittels Graphen

Für die Bildverarbeitung muss man schon vor dem Spiel alle erforderlichen Parameter möglichst schnell und optimal einstellen können, weil diese wahrend des Spiels nicht mehr verändert werden dürfen. Zudem kann die Vorbereitungsphase vor Spielbeginn recht kurz ausfallen und es bleibt kaum Zeit für umfangreiche Tests. Bereits bei der Installation des Bildverarbeitungssystems sollte besonderer Wert auf die optimale Einstellung der Kamerahöhe und der Schärfe des Kamerabildes gelegt werden, da diese Grundlage für eine stabil arbeitende Bildverarbeitung sind.

Die wichtigste und aufwendigste Einstellung innerhalb der Bildverarbeitung ist die Farbeinstellung. Erschwert wird dieser Vorgang dadurch, dass Patchfarben und Ballfarbe, die durch die Kamera erkannt werden, nicht immer mit den vom menschlichen Augen erkannten Farben übereinstimmen. Deswegen versucht man, basierend auf eigene Erfahrungen und Vermutungen, und den daraus entstehenden Farbwerten (z.B. H-S-L Werten), die gewünschten Farben (Ballfarbe, Patchfarbe) einzustellen. Diese HSL Farbwerte werden als Basisfarbwerte fur Ballerkennung oder Patcherkennung festgelegt und zudem werden entsprechende Toleranzen für diese Werte gesetzt. Dieser Vorgang wird so lange wiederholt, bis sich eine nahezu optimale Farberkennung eingestellt hat. Die Methode ist insgesamt allerdings sehr unsystematisch und stark abhangig von den eigenen Erfahrungen des Benutzers.

Um diesen Arbeitsschritt zu vereinfachen besteht die Möglichkeit, fur die Farbeinstellung einen Graphen zu verwenden. So wird z.B. ein Ball mit einer bestimmten Anzahl von Pixeln und mit bestimmten Farbwerten dargestellt. Jeder einzelne Pixel besitzt einen eigenen HSL-Wert und diese kann man als Graphen auf dem Monitor darstellen lassen. Man kann einfach erkennen, welche HSL-Werte am häufigsten auftreten und wo sich sogenannte Farb-Cluster bilden. Durch die Bestimmung der Grenzen der Farb-Cluster ist es möglich, schnell und systematisch die optimalen Farbwerte zu finden.

Kapitel 6

Services

In diesem Kapitel werden die während der Projektgruppe 416, zur Arbeitskontrolle und -steuerung, zur Informationserfassung und -verarbeitung sowie zur Präsentation des Projektes, eingeführten Dienste (im weiteren Verlauf auch als "Services" bezeichnet) beschrieben. Auf die Beweggründe für die Einführung bestimmter Services und deren Selektion aus verfügbaren Produkten wird der Schwerpunkt gelegt. Eine kurze Einführung in die Benutzung der ausgewählten Produkte wird ebenfalls durchgeführt. Eine detaillierte Benutzungsanleitung der Produkte würde allerdings den Rahmen dieser Arbeit sprengen und ist daher den jeweils verfügbaren Betriebsanleitungen der verwendeten Produkte vorbehalten.

Zur besseren Übersicht wurde eine Aufteilung in externe und interne "Services" durchgeführt. Externe Services dienen der Präsentation des Projektes nach aussen, während die internen Services ausschließlich der Optimierung der Arbeitsprozesse und der Informationserfassung und -verarbeitung dienen.

Folgende Services wurden im Laufe der Projektgruppe eingeführt:

- Webpräsenz (extern)
- Dokumenten- und Informationsverwaltung (intern)
- automatische Quellcode-Dokumentation (intern)
- Wiki (intern)
- Fehler- und Featureverwaltung (intern)

Im weiteren Verlauf werden nun die Beweggründe für die Einführung der einzelnen Services erläutert, sowie eine Auswahl eines Produktes vorgenommen, mit welchem sich die gewünschten Ziele optimal umsetzen lassen.

6.1 Webpräsenz

Zu Beginn der Projektgruppe existierte keine vom Lehrstuhl betreute Webpräsenz für den Roboterfussball, obwohl dieser Forschungsbereich doch einen recht hohen Stellenwert besitzt. Forschungsgruppen anderer internationaler Institute bieten dahingegen sehr umfangreiche Informationen zu ihrer Forschungsarbeit an.

Eine bereits vorhandene Webpräsenz, welche von einigen ehemaligen Projektgruppenmitgliedern gestaltet und betreut wurde, bot nicht die nötige Seriosität, um diese für die Präsentation des Projekts zu benutzen. Ausserdem bot die privat betreute Webpräsenz nur sehr wenige und teilweise veraltete Informationen an. Hinzu kam die Tatsache, dass die Webpräsenz privat gehostet wurde und somit nicht auf einem Universitätsserver verfügbar war. Dies hätte bedeutende Nachteile für die Kontrolle über die Webpräsenz zur Folge und hätte einen weniger offiziellen Charakter als eine von der Universität gehostete Webpräsenz.

Gründe für die Gestaltung einer neuen Webpräsenz gab es viele. An erster Stelle stand die Bereitstellung offizieller Informationen zum Forschungsprojekt “Roboterfussball” mit der nötigen Seriosität. Eine von der Universität gehostete Webpräsenz würde zudem einige Vorteile bieten:

- bessere Kontrolle der Webpräsenz und der Inhalte
- gesteigerte Verfügbarkeit
- Nutzung der zentralen Datensicherung des Fachbereichs

Zahlreiche anstehende öffentliche Veranstaltungen (siehe Seite 113) könnten die Popularität des Forschungsprojekts zudem enorm steigern, sofern eine ansprechende Webpräsenz zur vertiefenden Informierung verfügbar wäre. Nicht desto trotz könnte ein offizielles Auftreten im Internet auch beim ausfindig machen neuer industrieller Partner helfen.

Zur Neugestaltung der Webpräsenz wurde eigens eine kleine Arbeitsgruppe innerhalb der Projektgruppe gebildet, welche sich intensiv mit der schnellen Umsetzung der gemachten Forderungen beschäftigte. Zusätzlich wurden von dieser Gruppe eine Reihe von Massnahmen zur Steigerung der Popularität des

Forschungsprojekts geplant und durchgeführt. Die Ergebnisse dieser Arbeit sind sowohl in die Webpräsenz als auch in gesondertes Informationsmaterial geflossen, welches im Anhang dieser Arbeit zu finden ist. Alle erstellten Informationen wurden zudem einheitlich nach den Richtlinien einer erarbeiteten Corporate Identity gestaltet.

6.1.1 Inhalte

Der erste Schritt zur Erstellung der neuen Webpräsenz ist eine genaue Festlegung der Art und es Umfangs der bereitzustellenden Informationen. Hier muss ein besonderer Augenmerk auf die angestrebten Zielgruppen gelegt werden, um das Informationsangebot zielgruppengerecht anpassen zu können.

Als vorrangige Zielgruppe soll die interessierte öffentliche Masse angesehen werden. Hier soll die Webpräsenz das Forschungsprojekt anschaulich und verständlich präsentieren. Als besonderer Schwerpunkt soll die Vermittlung der Notwenigkeit derartiger Forschung angesehen werden, da gerade dies in der Vergangenheit eine sehr häufig gestellte Frage von Aussenstehenden war. Als zweite wichtige Zielgruppe sollen andere themenverwandte Forschungsgruppen angesehen werden. Diese bedürfen tiefergehende Informationen wie z.B. Diplomarbeiten, ausgearbeitete Vorträge und Konzepte. Zusätzlich ist es in Bezug auf diese Zielgruppe wichtig, dass alle angebotenen Informationen ebenfalls in Englisch verfügbar sind, da der Großteil des potentiellen Fachpublikums nicht aus dem deutschsprachigen Raum stammen wird.

Folgendes Informationsangebot wurde für die angestrebten Zielgruppen ausgearbeitet:

- Geschichte des Roboterfussballs
- Regelwerk der FIRA
- Beschreibung der verwendeten Hard- und Software
- Links zu themenverwandten Webseiten
- am Projekt beteiligte Personen (Projektgruppen, Betreuer, Mitarbeiter, Diplomanden)
- Veranstaltungsübersicht
- Spielberichte
- Diplomarbeiten
- Projektgruppenberichte

- erzielte Ergebnisse
- Bereitstellung von themenbezogenen Medien aller Art (Bilder, Videos, Dokumente)
- Diskussionsforum

Zusätzlich sollen noch einige obligatorische Informationen wie Kontaktadresse bzw -person, Impressum und ein Disclaimer bereitgestellt werden. Ein internen Webseitenbereich soll ebenfalls zur Verfügung gestellt werden, in welchem beispielsweise noch nicht freigegebene Inhalte zur Kontrolle zugänglich gemacht werden könnten. Des weiteren sind Dokumente für den ausschließlichen internen Gebrauch in diesem Bereich denkbar.

6.1.2 Struktur

Die Definition der Struktur der neuen Webpräsenz umfasst zum einen die Festlegung der zu unterstützenden Informationsmedien (WWW, WAP, usw.) und zum anderen die genaue inhaltliche Strukturierung der Webpräsenz. Beides sollte vor der Umsetzung genau definiert werden, um eine geeignete Informationsbasis für die Erstellung eines Seitendesigns zu haben.

Als Informationsmedium soll in erster Linie das normale WWW genutzt werden. Eine Präsentation in Form von WAP-Seiten soll vorerst nicht vorgenommen werden, da der Großteil der Zugriffe wahrscheinlich über normale WWW-Seiten erfolgen wird. Die Bereitstellung einer WAP-Version der Webpräsenz wäre zumindest für den Anfang unnötig verschenkte Zeit.

Bei der Strukturierung der Webpräsenz muss auf keinerlei bestehende Struktur Rücksicht genommen werden. Eine Übernahme der Struktuierung der bisherigen privaten Webpräsenz wäre zwar möglich, ist allerdings nicht ratsam, da diese für die Fülle der Informationen, welche bereitgestellt werden sollen, nicht ausreicht. Eine Benutzung dieser Strukturierung würde zwangsläufig zu unübersichtlich werden. Um die Übersichtlichkeit weiter zu steigern, sollen selten abgerufene bzw. relativ unwichtige Informationen aus der Hauptstruktur heraus genommen werden.

Auf Basis der bereitzustellenden Informationen und der Anforderungen an die Struktur wurde die folgende Seitenstruktur erarbeitet:

- Roboterfussball
  - Geschichte - Geschichte des Roboterfussballs (allgemein)

  - Organisationen - Übersicht über verschiedene Roboterfussballorganisationen
  - Regelwerk - grober Überblick über die aktuelle Fassung des Regelwerks

- Mitglieder
  - Projektgruppen - Liste der durchgeführten Projektgruppen
  - Betreuer - Betreuer der einzelnen Projektgruppen
  - Diplomanden - Liste der Diplomanden/Diplomarbeiten
  - HiWis - aktuelle Liste der studentischen Mitarbeiter

- System
  - Hardware - Beschreibung der Hardware
  - Software - Beschreibung der Software
  - Konzepte - Vorstellung erarbeiteter Konzepte

- Informationen
  - Pressemitteilungen - Mitteilungen für die Presse
  - Pressespiegel - Sammlung von Artikeln in der (lokalen) Presse
  - Veranstaltungen - Liste der besuchten Veranstaltungen
  - Spielberichte - Spielberichte von gespielten Turnieren
  - Titel / Ergebnisse - gewonnene Titel

- Galerie
  - Trophäen - Bilder der bisher gewonnen Preise
  - Jahresgalerie - Bilder nach Aufnahmejahr und Kategorien sortiert

- Download
  - Dokumente - alle freigegebenen Dokumente zum Download
  - Videos - angefertigte Videos zum Download

- Links
  - Organisationen - Verweise auf Roboter(fussball)-Organisationen
  - Hersteller - Verweise auf Hardwarehersteller

  - Mannschaften - Verweise auf andere Mannschaften/Forschungsgruppen

- Sitemap
- Login

Folgende weniger genutzte Informationen werden in einem gesonderten Menü auf der Webseite bereitgestellt:

- Kontakt - Kontaktperson, Anschrift, Mailadresse und Telefonnummer
- Impressum - die Verantwortlichen für die Webseite
- Disclaimer - Standardinformationen zum Haftungsausschluss
- Sponsoren - industrielle Sponsoren der Roboterfussball Mannschaft

Diese Struktur bietet ausreichende Übersicht über die bereitzustellenden Informationen. Zur weiteren Verbesserung der Übersicht sollen alle auf der Seite eingebundenen Dokumente sowohl im Download-Bereich als auch an der thematischen Stelle verlinkt werden.

6.1.3 Design

Das Design der Webpräsenz stellt einen der Schlüsselfaktoren für die erfolgreiche Präsentation des Projektes dar. Der erste optische Eindruck der Besucher ist besonders wichtig, da hier bereits die erste kritische Beurteilung von den Besucher vorgenommen wird. Ein unseriös wirkendes Design lässt bei der Präsentation derartiger Inhalte ein falsches Bild der Webseite bei den Besuchern entstehen. Obwohl diese noch keine Informationen durchgesehen haben, empfinden sie die angeboten Informationen als minderwertig oder gar fehlerhaft.

Das vorhandene Design der privaten Webpräsenz kann daher auf keinen Fall wiederverwendet werden, da die notwendige Seriosität der Webpräsenz nicht gewährleistet wäre. Hier muss also nach einer ganz anderen Gestaltung gesucht werden, was gegebenenfalls zu einem komplett neuen Design führt. Wünschenswert wäre natürlich die Adaption des Designs der Lehrstuhl- oder der Universitätswebseiten, um die Zugehörigkeit zur Universität klar zu kennzeichnen.

Im weiteren Verlauf sollen alle Überlegungen und Entscheidungen im Rahmen der Neugestaltung des Designs erläutert werden. Es kann allerdings keine professionelle Designstudie erwartet werden, da

dies zum einen sehr umfangreiche Kenntnisse in dieser Thematik voraussetzt und zum anderen den Umfang dieser Arbeit zu sehr ausweiten würde. Vielmehr soll auf Basis der verfügbaren Lehrstuhl- und Universitätsdesigns ein angemessenes Design für die neuen Webseiten erstellt werden.

Das Design der Lehrstuhlwebseiten[1] wirkt sehr einfach und schlicht. Es strahlt die nötige Seriosität aus, die man von einer derartigen Webpräsenz erwarten kann. Allerdings wirken die Webseiten aber auch sehr starr und eher altmodisch, was den doch sehr jungen Forschungsthema nicht gerecht wird. Die verwendeten blassen Farben lassen die Webseiten sehr trist aussehen. Dies kommt für die neue Webpräsenz aber nicht in Frage, da viele mit dem Roboterfussball neben der Forschung auch Spass assoziieren, was durch die Festlegung des Forschungsthemas auch beabsichtigt wurde.

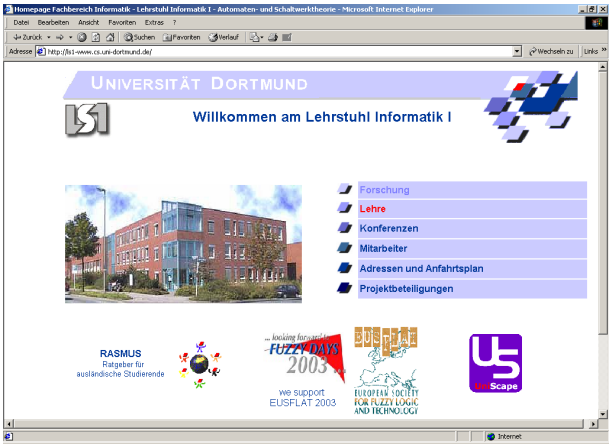


Abbildung 6.1: Webpräsenz des LS 1

Das Design der Universitätsseiten[2] wurde erst vor kurzer Zeit von Grund auf modernisiert. Daher wirken die Seiten sehr zeitgemäss. Die verwendeten Farben sind kräftig aber nicht aufdringlich. Die gesamte Farbgestaltung strahlt eine gewisse Freundlichkeit aus.

Alles spricht für die Anlehnung an das Design der Universitätsseiten, zumal das derzeitige Design der Lehrstuhlseiten an das alte Design der Universitätsseiten angelehnt wurde. Hier kann es bald zu einer erneuten Anpassung des Designs kommen, wodurch die Wahl des Universitätsdesigns auch die einzige Alternative für die Projektwebseiten darstellt. Eine derartige Umsetzung könnte gar eine enorme Arbeitserleichterung darstellen, sofern man bei der Anpassung der Lehrstuhlseiten auf die jetzt durchzuführenden Arbeiten an den Projektwebseiten zurück greift.

Um die zuvor festgelegte Struktur umsetzen zu können, bedarf es jedoch einiger Änderungen am Uni-

[1] `http://www.ls1-www.cs.uni-dortmund.de`
[2] `http://www.uni-dortmund.de`

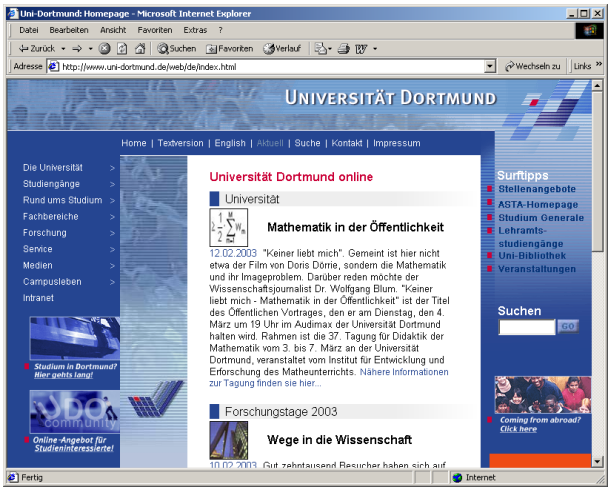


Abbildung 6.2: Webpräsenz der Universität Dortmund

versitätsdesign. Die Navigation auf den Universitätswebseiten ist technisch recht anspruchsvoll und funktioniert nicht mit allen Webbrowsern einwandfrei. Ausserdem wirkt die verwendete Navigation unnötig kompliziert, was einige Besucher von einer intensiveren Benutzung der Projektwebseiten abhalten könnte. Der für die Inhalte vorgesehene Bereich ist sehr klein. Da die Universitätswebseiten für eine feste Auflösung von 800 mal 600 Bildschirmpunkten ausgelegt ist, wird bei höheren Auflösungen unnötig viel Platz verschenkt. Hier wäre eine dynamische Anpassung der Webseite an die vorhandene Auflösung vorteilhafter. Als untere Grenze sollte aber weiterhin eine Auflösung von 800 mal 600 Bildschirmpunkten unterstützt werden.

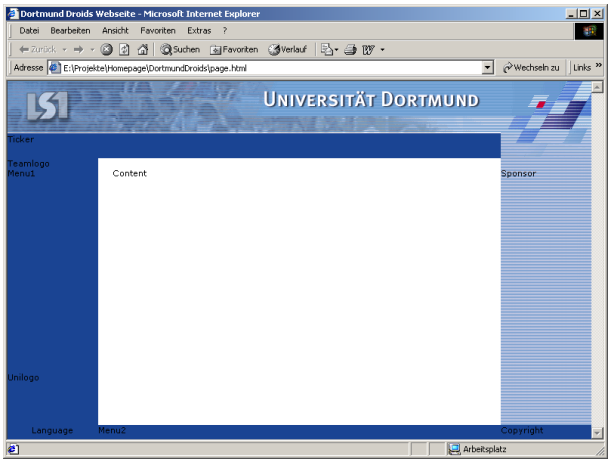


Abbildung 6.3: Designvorlage für die neue Webpräsenz

6.1.4 Redaktionssystem

Durch die beschlossene Schaffung einer offiziellen Webpräsenz für den Roboterfussball am Lehrstuhl, muss nicht auf bestehende Webseiten-Strukturen Rücksicht genommen werden. Zusätzlich kann die technische Form der Webpräsenz nun diskutiert und vorteilhaft ausgewählt werden. Eine Weiterführung der bisher für die private Roboterfussball-Webseite verwendeten Technik kann ausgeschlossen werden. Diese technische Umsetzung bestand in der manuellen Generierung der Webseiten, was einige Probleme nach sich zieht. Zum einen sind zwingend HTML-Kenntnisse erforderlich und zum anderen wird kein richtiger Mehrbenutzer-Betrieb ermöglicht. Ausserdem ergeben sich Probleme bei der einheitlichen Gestaltung der einzelnen Webseiten, da ein einheitliches Design nur mit sehr viel Mühe umgesetzt werden kann.

Die Lösung für alle Probleme ist die Benutzung eines professionellen Redaktionssystems. Diese sind in der Regel mehrbenutzerfähig und erfordern meistens keinerlei HTML-Kenntnisse. Des weiteren lässt sich somit leicht ein einheitliches Seitendesign umsetzen, da gute Redaktionssysteme zwischen Inhalt, Struktur und Design unterscheiden.

In der Praxis könnte man Inhalte eingeben, ohne die genaue Ausgabestruktur zu kennen. Die gleichen Inhalte könnten so für verschiedene Ausgabemedien genutzt werden (HTML, WAP, XML, usw.), was einer eventuell später vorgesehenen WAP-Version der Webpräsenz entgegen kommt. Zusätzlich muss sich ein Redakteur keinerlei Gedanken über das eigentliche Seitendesign machen, da das System die bereitgestellten Inhalte automatisch in die vorgegebene Designvorlage einpasst.

6.1.4.1 Anforderungen

Um ein geeignetes Redaktionssystem zu finden, werden zuerst eine Reihe von Anforderungen an das System gestellt. Diese werden in harte und weiche Anforderungen unterteilt: harte Anforderungen müssen zwingend erfüllt werden, während weiche Anforderungen lediglich als wünschenswert angesehen werden:

- strikte Trennung zwischen Inhalt und Design (hart)
- Verfügbarkeit fertiger Teilkomponenten (hart)
- Mehrsprachfähigkeit (hart)
- Mehrbenutzerfähigkeit (hart)

- einfache Bedienung (hart)
- einfache Installation und Wartung (weich)

Der Trennung zwischen Inhalt, Struktur und Design wird große Bedeutung beigemessen, da ein einheitliches Design für eine seriöse Präsentation des Forschungsprojekts unerlässlich ist. Bereits verfügbare Teilkomponenten, wie definierte Inhaltstypen, erlauben eine viel schnellere Umsetzung der Webpräsenz. Um die Bereitstellung aller Informationen in mehreren Sprachen zu ermöglichen, muss das System mehrsprachfähig sein. Die Mehrbenutzerfähigkeit ist ebenfalls von enormer Wichtigkeit, da gerade in der Aufbauphase der Webpräsenz besonders viele Informationen zeitgleich eingegeben werden müssen. Eine einfache Bedienung des Systems soll ebenfalls gewährleistet sein, um so die Einarbeitungs- und Umsetzungszeit weiter zu minimieren. Zusätzlich ist eine einfache Installation und Wartung des Systems zwar wünschenswert, aufgrund der relativen Seltenheit dieser Arbeitsschritte aber nicht zwingend erforderlich.

Neben den oben aufgeführten technischen Anforderungen an das Redaktionssystem ist die freie Verfügbarkeit im Internet für die in Frage kommenden Redaktionssysteme elementar, da die recht hohen Lizenzkosten für kommerzielle Redaktionssysteme nicht im Rahmen der Projektgruppe aufgebracht werden können. Aus diesem Grund werden im weiteren Verlauf nur kostenlose Redaktionssysteme miteinander verglichen.

6.1.4.2 Redaktionssystem *Typo3*

Die Wahl fiel aufgrund verschiedener Aspekte auf das CM-System Typo3. Einige der Projektgruppenmitglieder hatten bereits ausserhalb der Projektgruppenarbeit mit diesem System gearbeitet. Dies, und die gewünschte möglichst schnelle Bereitstellung der neuen Webpräsenz, waren ausschlaggebend dafür, keine weiteren Redaktionssysteme auf ihre konkrete Eignung zu überprüfen.

Das CM-System Typo3 erfüllt fast alle aufgestellten Anforderungen. Wie bei fast allen Redaktionssystemen üblich, werden Inhalt und Design vollständig voneinander getrennt. Aufgrund dessen kann ohne Probleme die Eingabe der Inhalte noch während der Umsetzung der Designvorlage vorgenommen werden, was eine enorme Zeitersparnis im Verlauf der Arbeiten zur Folge hat. Durch die Unterstützung mehrerer Sprachen ist eine mehrsprachige Umsetzung der Inhalte auf der Webseite leicht zu realisieren. Die Verfügbarkeit einer Vielzahl zusätzlicher Komponenten erhöht die Zeitersparnis zusätzlich. Durch die Mehrbenutzerfähigkeit sowie durch das sehr einfache Bedienungskonzept wird die Arbeit der einzelnen Redakteure wesentlich erleichtert und beschleunigt. Ausserdem werden die oft

sehr langen Einarbeitungsphasen auf ein Minimum reduziert. Einzig die Installation von Typo3 gestaltet sich etwas schwieriger. Durch einige Eigentümlichkeiten bleibt die Installation nur eher erfahrenen Linuxbenutzern vorbehalten. Aufgrund eines durchdachten Update-Konzepts ist die Aktualisierung einer Typo3-Installation aber wiederum sehr einfach. Ein in Typo3 integrierter Erweiterungsmanager erlaubt zudem die einfache Installation neuer Module und die Aktualisierung älterer Module per Mausklick und kann somit auch von unerfahrenen Benutzern leicht durchgeführt werden.

Zusätzliche Informationen zu Typo3 können unter `http://www.typo3.org` sowie unter `http://www.typo3.info` gefunden werden. Hier werden neben weiterführenden Informationen auch Anleitungen zur Installation und Benutzung in verschiedenen Sprachen angeboten.

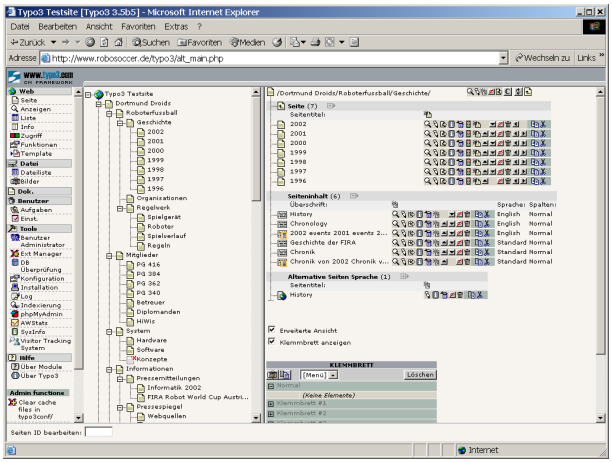


Abbildung 6.4: Content Management System *Typo3*

6.1.5 Ergebnis

Die Umsetzung der Webpräsenz mit Hilfe von Typo3, auf Basis der erstellten Designvorlage, ging sehr schnell von statten. Dank des CM-Systems konnte bereits zu einem sehr frühen Zeitpunkt der Umsetzung mit der Eingabe von Inhalten begonnen werden, was einer extrem kurzen Realisierungszeit zu Gute kam. Das Bedienungskonzept erwies sich, wie bereits erwartet, als sehr intuitiv, und ermöglichte eine schnelle Einarbeitung der Redakteure sowie das zügige Einpflegen der Inhalte. Das Informationsangebot auf den neuen Webseiten wurde wie geplant umgesetzt. Zusätzlich wurde für die Webpräsenz ein Diskussionsforum eingerichtet, um eine öffentliche themenbezogene Diskussion zu fördern.

Die erste Version der Webseite ging aufgrund zeitlicher Probleme anfänglich nur in deutscher Sprache online. Eine englische Version der Inhalte war auch erst zur GI-Tagung von Interesse. Mittlerweile ist

die komplette Webpräsenz übersetzt und steht neben der deutschen auch in einer englischen Version für internationale Besucher zur Verfügung.

Die neue Webpräsenz ist seit dem 7. August erreichbar unter `http://www.robosoccer.de`.

Abbildung 6.5: die neue Projekt-Webpräsenz

Durch die sehr modulare Umsetzung der neuen Webpräsenz können die entstandenen Vorlagen und Typo3-Strukturen leicht für andere Projekte genutzt werden. Es ist sogar möglich weitere Projekte in derselben Typo3 Installation anzulegen. In diesem Falle müssen nur noch die spezifischen Bestandteile des neuen Webprojekts, wie beispielsweise das Logo, angepasst werden.

6.2 Arbeitsabläufe

Um die im weiteren Verlauf geschilderten Beweggründe für die Notwendigkeit der Einführung bestimmter Produkte besser nachvollziehen zu können, werden an dieser Stelle die organisatorischen Abläufe innerhalb einer Projektgruppe beschrieben. Besonders problematische Situationen in den Arbeitsabläufen werden aufgezeigt, um auf dieser Basis die gemachten Verbesserungen zu verdeutlichen.

Mit einer Dauer von gut einem Jahr und einer bis zu 12 Personen großen Mitarbeiterzahl, ergeben sich eine Reihe von Problemen, die es zu meistern gilt, sofern man mit die gestellten Aufgaben mit Erfolg bewältigen möchte. Hier interessieren aber in erster Linie die organisatorisch bedingten Probleme innerhalb der Mitarbeitergruppe - aufgabenbedingte Probleme werden nicht betrachtet.

Damit die Aufgaben der einzelnen Mitarbeiter verteilt, kontrolliert und koordiniert werden können,

finden anfänglich zwei Projektgruppensitzungen pro Woche statt. Zu einem späteren Zeitpunkt wird die Anzahl der regelmäßigen Treffen auf eine Sitzung pro Woche reduziert, sofern nichts gegen diese Massnahme spricht. Im Normalfall sollten diese Treffen für die Koordinierung der Arbeiten ausreichen. Doch im Unterschied zu einer Software-Entwicklungsfirma ist oft ausserhalb dieser Sitzungen keine direkte Kommunikation mit den Mitarbeitern möglich. Wichtige Entscheidung können von den beteiligten Personen oft nur asynchron getroffen werden. In der Woche gesammelte Erkenntnisse und Fragen werden zudem nicht immer vollständig auf den wöchentlichen Sitzungen präsentiert. Teilweise gehen die auf der Sitzung angesprochenen Informationen auch verloren. Zwar werden alle Informationen protokolliert, jedoch ist es dem Protokollführer gelegentlich nicht möglich die geäußerten Informationen in ihrem vollen Umfang zu erfassen. Nicht selten kommen auch Ungenauigkeiten in der Präsentation der Informationen vor, was zu großen Missverständnissen führen kann. Zu guter letzt werden die Informationen durch ein simples Protokoll oft nicht in der gewünschten Form bereit gestellt. Zusammenfassend können folgende Probleme ausgemacht werden:

- Asynchrone Kommunikation
- ungenaue Informationserfassung bzw. -präsentation
- dezentrale Speicherung von Informationen
- nicht zweckgebundene Speicherung von Informationen

Die Probleme durch die meist asynchrone Kommunikation und die ungenaue Informationspräsentation lassen sich nur direkt durch die beteiligten Mitarbeiter lösen. Alle anderen Probleme kann man jedoch mit dem Einsatz von spezieller Software abmildern oder sogar eliminieren.

6.3 Dokumenten- und Informationsverwaltung

Mit einer zentralen Dokumenten- und Informationsverwaltung lassen sich alle nicht personenbezogenen Probleme der Informationserfassung, -speicherung und -weitergabe beseitigen. Schon zu Beginn der Projektgruppe wurde aus diesem Grund ein professionelles Dokumentenverwaltungssystem vom Veranstalter bereit gestellt.

Das unter dem Namen *Intrexx*[3] bekannte Produkt ermöglicht durch eine Vielzahl verfügbarer Module eine kontextabhängige Speicherung von Informationen und Dokumenten. Alle Informationen werden

[3] `http://www.intrexx.de`

zentral auf dem Intrexx-Server abgelegt und können so ohne Probleme mit Hilfe einer zentralen Datensicherung gesichert werden. Der Zugriff auf den Server erfolgt mit einem gewöhnlichen Webbrowser, was die Einsicht der und die Bearbeitung der Daten jederzeit von jedem Ort ermöglichen sollte. Mitarbeiter können gezielt Informationen eingeben und diese auf Wunsch anderen bereitstellen. Auf diese Weise kann es nur sehr schwer zu ungenauen Informationen kommen, da diese nicht erst durch Dritte erfasst werden. Natürlich ist aber auch hier ein gewisses Maß an Genauigkeit bei der Eingabe der Daten gefragt. Durch die kontextabhängige Speicherung stehen die Daten sofort im richtigen Zusammenhang zueinander und sind entsprechend kategorisiert. Jeder Mitarbeiter kann sich so gezielt die Informationen von dem System anzeigen lassen, ohne sich mit für ihn irrelevanten Daten aufhalten zu müssen.

Leider haben sich während der Benutzung des Intrexx-Servers eine Reihe von Problemen ergeben, welche die Arbeit etwas verkompliziert haben. Aufgrund instabiler Hardware kam es mehrmals zu längeren Ausfällen des Servers, wodurch er nicht für die Arbeit bereit stand. Ausserdem gab es für einige Informationen keine fertigen Module, um die Informationen kontextabhängig abzulegen. Einige Module wirkten auch etwas unausgereift. Leider konnte auch die Arbeitsgeschwindigkeit über eine ISDN-Verbindung nicht sonderlich überzeugen.

Hauptsächlich wurde der Intrexx-Server zur Verwaltung aller anfallenden Dokumente genutzt. Sowohl Protokolle als auch Diplomarbeiten und Projektgruppenberichte wurden hier zentral zugänglich gemacht. Anfänglich wurden auch Aufgaben und Zuständigkeiten mit dem Server verwaltet, was allerdings bedingt durch die beschriebenen Probleme relativ schnell wieder aufgegeben wurde.

Insgesamt ist die Benutzung des Intrexx-Servers als positiv anzusehen, da er doch zu Verbesserung der Informationsverwaltung beigetragen hat. Durch die intuitive Bedienung des Intrexx-Servers fällt die Einarbeitungszeit zudem sehr gering aus. Dies ermöglicht eine sehr schnelle Integration in die Arbeitsweise einer Gruppe. Für eine genauere Einführung in die Benutzung des Intrexx-Servers sei an dieser Stelle auf den Seminarband der Projektgruppe 416 verwiesen.

6.4 automatische Quelltext Dokumentation

Werkzeuge zur automatischen Quelltext Dokumentation ermöglichen es, mit Hilfe von formatierten Kommentaren innerhalb eines Quelltextes, eine umfangreiche Dokumentation zu diesem zu erstellen. Die entsprechenden Strukturen für die Kommentare sind sehr einfach gehalten und sorgen bei richtiger Verwendung sogar für einen besser lesbaren Quelltext.

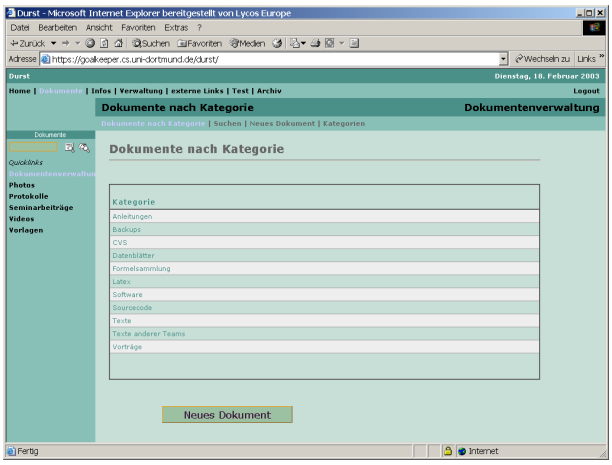


Abbildung 6.6: Dokumentenverwaltung auf Basis des Intrexx-Servers

Mit *doxygen*[4] gibt es ein solches Dokumentationssystem für C/C++, welches ausserdem kostenlos verfügbar ist. Basierend auf JAVADOC verarbeitet doxygen die für dieses System übliche Kommentarstruktur. Zusätzlich wird jedoch auch die Kommentarstruktur des mit QT ausgelieferten Dokumentationstools unterstützt. Dank verschiedener Ausgabemedien (HTML, LaTeX, PDF) ist doxygen sehr vielseitig. Doxygen ermöglicht es sogar aus nicht richtig strukturierten Kommentaren bzw. aus völlig unkommentierten Quelltexten eine recht übersichtliche Dokumentation erzeugen. Eine derartige Dokumentation kann gerade in einer Einarbeitungsphase nützliche Dienste leisten.

Leider fand ein derartiges Werkzeug bisher keine Verwendung in Projekt. Doch dank der Fähigkeit auch aus teilweise unkommentierten Quelltexten eine recht übersichtliche Dokumentation zu erzeugen, konnte die Einarbeitungszeit in die bestehenden Quelltexte stark verkürzt werden. Die mit doxygen erzielten Ergebnisse ermutigten den weiteren Einsatz: neu geschriebene Quelltexte wurden mit entsprechenden Kommentarstrukturen versehen. Einige überarbeitete Passagen wurden ebenfalls entsprechend neu kommentiert. Durch den stetigen Einsatz wurde beschlossen, die Dokumentation jede Nacht automatisch erzeugen und zentral im geschützten Bereich des Webservers bereitstellen zu lassen. Die Dokumentation wird nun über Nacht für die jeweils aktuelle Version der Quelltexte angefertigt.

[4] `http://www.doxygen.org`

6.5 Wiki

Mit Hilfe eines Wikis lassen sich in kürzester Zeit und mit minimalen Verwaltungsaufwand Informationen bereitstellen und stark untereinander vernetzten. Aufgrund des Fehlens einer formlosen Informationseingabe als Basis für eine Ideen- und Konzeptsammlung für den Intrexx-Server, wurde daher die probeweise Installation und Benutzung eines Wikis vorgeschlagen.

Die verwendete Wiki-Version[5] basiert auf PHP und besteht nur aus sehr wenigen Skriptdateien. Eine Installation ist sehr schnell und einfach möglich, da neben dem bereits vorhandenen PHP keinerlei weitere Systemvoraussetzungen gestellt werden. Ausserdem lässt sich das Wiki leicht modifizieren und so perfekt an die Anforderungen anpassen.

Die Benutzung des Wikis ist sehr einfach. Es gibt weder eine Benutzerverwaltung noch eine Zugriffsbeschränkung. Jeder Beteiligte kann beliebige Seiten ansehen und diese durch das Auswählen eines Links beliebig verändern. Mit Hilfe einiger Schlüsselwörter lassen sich leicht die eingegebenen Informationen formatieren. Zusätzlich können Verknüpfungen mit anderen Seiten herstellen. Hierzu wird einfach der Titel der Zielseite mit einem bestimmten Schlüsselwort-Paar umschlossen. Sofern die angegebene Seite existiert, wird auf der Seite ein Link zu dieser angelegt. Sollte die Seite jedoch noch nicht existieren, wird auf der Seite eine Option zur Eingabe von Informationen zu der neuen Seite angeboten.

Zu Beginn wurden exemplarisch alle bestehenden Services auf einer geschützten internen Wiki-Webseite zusammen gefasst. Von hier haben alle Beteildigten direkten Zugang zu allen Informationswerkzeugen. Obwohl sich das Wiki perfekt zur Strukturierung neuer Ideen und Konzepte einsetzen lässt, wurde es bisher nur sehr spärlich benutzt. Dies kann eventuell auf die recht späte Einführung dieses Werkzeugs zurückgeführt werden.

6.6 Fehler- und Featureverwaltung

Mit Produkten zur Fehlerverwaltung lassen sich auf einfache Weise aufgetretene Fehler zentral erfassen. Oft lassen sich durch entsprechende Konfiguration mehrere Projekte mit unterschiedlichen Fehlerkategorien in einem System verwalten. Gute Systeme stellen hier auch direkt umfangreiche Funktionen zur Bearbeitung der Fehler und zur Diskussion möglicher Ursachen zur Verfügung.

Ein derartiges System fand zuvor im Projekt keine Verwendung. Fehler wurden einfach schriftlich erfasst und verwaltet. Eine zentrale Verwaltung der gesammelten Informationen war so nur sehr schwer

[5] `http://ontosys.com/phiki/`

möglich. Die Einführung eines solchen Systems war also mehr als überfällig, da gerade durch papierbasierte Informationserfassung wichtige Informationen verloren gingen. Leider wies der zur Verfügung stehende Intrexx-Server keine entsprechende Funktionalität auf. Letztendlich musste ein zusätzliches System beschafft und installiert werden.

Damit ein System zur Fehlerverwaltung sinnvoll im Rahmen der Projektgruppe eingesetzt werden kann, werden die folgenden Anforderungen für ein solches gestellt. Erneut wird zwischen notwendigen (harten) und wünschenswerten (weichen) Anforderungen unterschieden:

- Benutzerverwaltung mit Rechtemanagment (hart)
- Zuweisung von Fehlern / Arbeitsworkflow (hart)
- frei definierbare Fehlerklassen und Kategorisierung (hart)
- Prioritätenvergabe für Fehler (hart)
- einfache Bedienung (hart)
- einfache Installation (weich)

Um das Arbeiten in der Gruppe zu vereinfachen, ist eine Benutzerverwaltung mit Rechtemanagment unerlässlich. Funktionen zur Zuweisung bestimmter Rollen (Entwickler, Tester, usw.) wären ebenfalls wünschenwert, um die Arbeit stärker organisieren zu können. In diese Richtung zielt auch der Wunsch nach einer Zuweisungsmöglichkeit für Fehler. Protokollierte Fehler können so gezielt einzelnen Benutzern zur Bearbeitung zugewiesen werden. Für eventuelle Rückfragen können die Benutzer angesprochen werden, welche den jeweiligen Fehler gemeldet hatten. Frei definierbare Fehlerklassen ermöglichen eine Kategorisierung der Fehler schon bei der Eingabe und sorgen so für gesteigerte Übersicht. Eine Prioritätenvergabe sorgt letztendlich für eine zusätzliche Kategorisierung auf Basis der Dringlichkeit. Natürlich sollte das verwendete Produkt ebenfalls leicht bedienbar sein und keine lange Einarbeitungszeit voraussetzen. Eine einfache Installation des Produktes wäre natürlich wünschenswert, ist allerdings nicht notwendig, da dieser Arbeitsschritt nur einmalig zu erfolgen hat.

Fehlerverwaltungssysteme sind im OpenSource-Bereich bereits sehr zahlreich. Viele Systeme unterstützen aber nicht alle Anforderungen. Gerade ein umfangreiches Rechtemanagment ist in vielen Systemen nicht vorhanden. Bei der Suche nach geeigneten Systemen haben sich die Produkte *Bugzilla*[6]

[6] `http://bugzilla.mozilla.org`

und *Mantis*[7] von der Masse der verfügbaren Produkte abgehoben. Beide unterstützen laut Herstellerspezifikation die gemachten Anforderungen und sind somit für den Einsatz geeignet. Aufgrund der eher komplexeren Installation von Bugzilla wurde das Produkt Mantis für einen ersten Test vorgezogen. Da Mantis lediglich eine MySQL-Datenbank und die Skriptsprache PHP voraussetzt, gestaltete sich die Installation denkbar einfach, da die entsprechende Systemumgebung bereits für das Redaktionssystem Typo3 hergestellt werden musste. Da Mantis während der Testphase erstaunlich schnell akzeptiert wurde und keinerlei Schwächen zeigte, hat man auf weitere Tests mit Bugzilla verzichtet.

Die Benutzung der Fehlerverwaltung hat sich sehr schnell in die Arbeitsweise der Entwickler eingegliedert. Durch die zentrale Erfassung der Fehler und Featurewünsche haben alle Entwickler jederzeit Zugriff auf die Informationen. Durch die Möglichkeit zur Kommentierung von Fehlern konnte eine effektive themenbezogene Diskussion gemachter Lösungsvorschläge geschaffen werden. Die Auswirkungen der Einführung von Mantis sind durchweg als positiv zu bezeichnen.

6.7 Ausblick

Durch die stetige Anpassung und Erweiterung der Services konnte eine sehr effiziente Arbeitsweise etabliert werden. Viele durch die Arbeitsform bedingten Nachteile konnten durch die Einführung der einzelnen Services gemildert oder ganz behoben werden. Leider wurde aber aufgrund der derzeitigen Schwächen von Intrexx eine Verteilung der Informationen auf mehrere spezialisierte Werkzeuge nötig. Diese benötigen größtenteils eigene Zugangsdaten, wodurch es zu erhöhtem administrativen Aufwand kommt.

Sobald entsprechende Intrexx-Module verfügbar sind, sollte überprüft werden, ob diese nicht alternativ eingesetzt werden können. Ziel wäre es die komplette Informationsverwaltung auf ein System zu zentralisieren. Durch eine Beschränkung auf ein einzelnes System zu Informationsverwaltung ergeben sich eine Reihe von Vorteilen:

- zentrale Verfügbarkeit aller Informationen
  - vereinfachter Zugriff auf Informationen
  - Datensicherung erheblich einfacher zu realisieren
- weniger administrativer Aufwand

[7] `http://mantisbt.sourceforge.net`

  - eine einzelne Benutzerverwaltung
  - weniger zeitaufwendige Systempflege

- einfachere Handhabung durch einheitliche Bedienung

Vor einer Umstellung sollte allerdings genau überprüft werden, ob diese wirklich sinnvoll ist. Bereits etablierte Arbeitsabläufe und Werkzeuge sollten wirklich nur verändert werden, sofern die sich ergebenden Vorteile beim Einsatz wesentlich größer sind als die Nachteile bedingt durch erneute Einarbeitung und Datenmigration.

Anhang A

Arbeitsumgebung und Werkzeuge

A.1 Rechnerlandschaft

Während der Projektgruppenarbeit ist die Rechnerlandschaft mehrfach umgebaut worden. Einer der wesentlichen Gründe war das unerwartet gute Abschneiden der Vorgänger-Projektgruppe 384 bei den Weltmeisterschaften in Korea. Aufgrund der Plazierung als Vizeweltmeister ergab sich eine gute Gelegenheit sich erneut um Sponsorengelder zu bemühen. Dafür mußte die Projektgruppe jedoch einige Arbeit in die Präsentation des Roboterfußballs investieren, und das Ergebnis ist ein neu aufgesetzter Webserver mit der wohl europaweit besten Roboterfußballseite im FIRA-Bereich.

A.1.1 IP-Addressbereich der Dortmund Droids am LS1

Zu Anfang der PG ist ein größerer IP-Adresspool für Notebooks beantragt worden. Die Tabelle in Abb. A.1 gibt Aufschluß über die bewilligten Adressen und deren derzeitige Verwendung:

A.1.2 soccer

Soccer ist unter allen vom Dortmund Droids Team eingesetzten Rechnern der am stärksten belastete Server. Er hat eine eingebaute Framegrabberkarte und ist leicht zu transportieren, so daß er für Turnierspiele mitgenommen werden kann. Auf ihm laufen – abgesehen von Intrexx und Web-Serverdiensten – sämtliche Serverdienste, die die Projektgruppe für das Arbeiten benötigt.

Im wesentlichen sind das ein NFS-Server, ein NIS-Server sowie ein CVS-Server. Der NFS-Server sorgt für die Verteilung der Homeverzeichnisse auf die anderen Rechner, während der NIS-Server

IP-Adresse	Rechnername	Bemerkungen
129.217.57.64	fira	(siehe A.1.3)
129.217.57.65	soccer	(siehe A.1.2)
129.217.57.66	kicker	(siehe A.1.4)
129.217.57.67	goaly	Arbeitsrechner
129.217.57.68	robot	Arbeitsrechner
129.217.57.69	mirosot	Arbeitsrechner
129.217.57.70	durst	Arbeitsrechner
129.217.57.71	patchi	(reserviert für Notebooks, derzeit belegt von Carsten Fenneker)
129.217.57.72	piwo	(reserviert für Notebooks, derzeit belegt von Marc Sölter)
129.217.57.73	crispy	(reserviert für Notebooks, derzeit belegt von Norman Weiß)
129.217.57.74	epoxy	(reserviert für Notebooks, derzeit belegt von Hendrik Valentin)
129.217.57.75	referee	(reserviert für Notebooks, derzeit belegt von Sung-Kwon Jung)
129.217.57.76	laola	(reserviert für Notebooks)
129.217.57.77	foul	(reserviert für Notebooks, derzeit belegt von Sven Bursch)
129.217.57.78	penalty	(reserviert für Notebooks)
129.217.57.79	coach	(reserviert für Notebooks)
129.217.57.80	kickoff	(reserviert für Notebooks)
129.217.57.81	khepera	(reserviert für Notebooks)
129.217.57.82	trainer	(reserviert für Notebooks)
129.217.57.83	goalkeeper	(siehe A.1.5)

Abbildung A.1: IP-Adressen des Fira-Roboterpool am LS1

die Usernamen/Passwort-Anfragen beantwortet. Der CVS-Server bietet ein Versionsmanagement für die Programmentwicklung.

A.1.3 fira

Fira ist der zweite Rechner mit einer Framegrabberkarte, er kann also ebenfalls auf Turniere mitgenommen werden. Seit kurzem ist in fira auch eine Firewire-Karte eingebaut, so daß zukünftig auch mit Hilfe einer Firewire-Kamera gespielt werden kann, sobald das robotsoccer-Programm geeignet angepasst wurde.

Ansonsten ist fira jedoch ein ganz gewöhnlicher Abeitsrechner ohne besondere zusätzliche Services.

A.1.4 kicker

Kicker ist ein Rechner mit einem Windows-Betriebssystem – der Code-Composer benötigt Windows (es existiert keine Linux-Version). Somit ist Kicker einer der wenigen Rechner, auf dem die Programme für die DSPs weiter entwickelt und getestet werden können. Hierfür ist an Kicker ein Emulator-Board

angeschlossen, welches direkt mit der Bordelektronik der Roboter verbunden werden kann.

A.1.5 goalkeeper

Goalkeeper ist „das Tor zur Welt“. Auf ihm läuft zunächst einmal ein Webserver für `http://www.robosoccer.de/`. Dieser Webserver wird auf goalkeeper mit einem Paket-Filter von möglichen Angriffen aus dem Internet weitestgehend abgeschirmt. Ein weiterer Dienst auf goalkeeper ist ein SSL-Tunnel, welcher es ermöglicht, auf interne Bereiche des Webservers verschlüsselt und Passwort-geschützt zugreifen zu können.

A.1.6 devon

Devon ist kein eigener Rechner des Dortmund Droids Teams. Er steht auch nicht im DD-Rechnerpool, sondern in Raum 118. Auf diesem Rechner lief bis vor kurzem noch ein Intrexx-Server, der von der PG als Dokumentenserver genutzt wurde. Leider ist durch einen unglücklich verlaufenen Updateversuch dieses Rechners der Intrexx-Server sehr instabil geworden. Zukünftig sollen Dokumente im Intrexx-Server des Lehrstuhls eingebunden werden und somit devon für das Roboterfußballteam bedeutungslos werden.

A.2 Eigenes robotsoccer-System installieren

Während des Verlaufs der PG ist es zweimal passiert, daß angehende Diplomanden und Teilnehmer der PG eine eigene Installation des Robotsoccer-Projektes auf ihren Privatrechnern benötigten bzw. zuhause an der Weiterentwicklung arbeiten wollten. Leider gab es bis zu diesem Zeitpunkt keine Anleitung, die Schritt für Schritt die Installation für eines solchen Systems erklärte.

Somit lag die Idee nahe, eine Kurzanleitung in den Endbericht einzubauen, zumal zukünftig möglicherweise Kooperationen mit anderen Universitäten anstehen, und die dortigen Teams zumindest eine grobe Übersicht bekommen sollten, wie das System in Betrieb genommen werden kann.

Die folgenden Abschnitte beschäftigen sich nun mit genau diesem Thema. Es kann allerdings nur ein grober Überblick gegeben werden, wie ein System aufgesetzt werden kann, da viele der Schritte wesentlich davon abhängen, welche Hardware eingesetzt werden soll. Deshalb wird die Anleitung hier auf die Installation eines Systems mit einer nVidia-kompatiblen Grafikkarte und ohne Kamera beschränkt.

A.2.1 Linux installieren

Das robotsoccer-Programm läuft unter dem Betriebssystem Linux. Da es hier die unterschiedlichsten Distributionen gibt, wird hier nur exemplarisch S.u.S.E. Linux 7.3 installiert. Einmal ist es das in Deutschland am weitesten verbreitete Linux und zum zweiten die im Roboterpool derzeit eingesetzte Distribution.

A.2.1.1 S.u.S.E. Linux 7.3

Als erstes ist mit Yast2 die „KDE Desktop-Umgebung“ und „Entwicklung“ in der „Erweiterten-Paketauswahl“ zu installieren. Wie das im einzelnen gemacht wird ist dem mitgelieferten Linux-Installationshandbuch oder der Datei „book-suselinux-reference_de.pdf“ auf dem ersten Installationsmedium im Ordner „docu“ zu entnehmen.

Ausgehend von diesem System werden hier alle zusätzlich noch benötigten Pakete angegeben; diese sind somit noch zusätzlich zu installieren.

Zu den noch benötigten Paketen zählen zum einen ein C++ Compiler mit allen dazu gehörenden Unterpaketen sowie alle noch nicht installierten Bibliotheken und include-Dateien, die vom Robotsoccer-Programm verwendet werden. Einen Hinweis, welche Bibliotheken das Robotsoccer-Projekt benötigt, erhält man aus den Makefiles des Projektes – konkret aus der Datei „options“ im Ordner „makefiles“ unter den Punkten „MF2_LINK_QT“ und „MF2_LINK_MISC“. Hier stehen jeweils hinter den „-l“ (Strich-klein-el) die benötigten Bibliotheken.

Unglücklicherweise sind nicht alle diese Bibliotheken auch unter diesem Namen als Paket auf den Installationsmedien gespeichert, so daß noch eine weitere Datei zu Rate gezogen werden muß: „ARCHIVES.gz“ auf dem ersten Installationsmedium enthält eine komplette Liste aller Dateien, die auf allen Datenträgern der gleichen Distribution enthalten sind, mit den zugehörigen Paketnamen. In Abb. A.2 werden die z.Z. benötigten Libraries und die zugehörigen Pakete aufgelistet. Im Zweifelsfall sollte mit *rpm -q <Paketname>* auf soccer nachgeprüft werden, welche der möglichen alternativen Pakete installiert sind.

Sind alle noch ausstehenden Pakete ausgewählt und alle Paketabhängigkeiten erfüllt (zur Not automatisch auflösen lassen), so können die Pakete installiert werden.

Bibliothek	enthalten in Paket
libGL	siehe A.2.3
libGLU	xf86glu-devel.rpm
libXmu	xdevel.rpm
libXext	xdevel.rpm
libm	glibc-devel.rpm
libqt	qt-devel.rpm und qt.rpm (Version 2.3.1)
libX11	xdevel.rpm
libjpeg	libjpeg.rpm
libg++	libgpp.rpm
libpthread	glibc-devel.rpm
libncurses	ncurses-devel.rpm

Abbildung A.2: Für robotsoccer benötigte Pakete

A.2.2 Grafikkartentreiber

Des weiteren ist jetzt ein OpenGL 3D-Graphikkartentreiber notwendig. Dieser ist wesentlich von der eingesetzten Graphikkarte abhängig. Hier wird Beispielhaft die Installation eines nVidia-Treibers erklärt.

A.2.3 nVidia

Zunächst muß der aktuelle Treiber von den Internetseiten von nVidia heruntergeladen werden: `http://www.nvidia.com`

Die derzeitigen Treiber bestehen aus zwei Dateien: zum einen ein Kernelmodul und zum zweiten ein GLX-Treiber. Beide werden jeweils für die auf diesem Rechner installierte Linux Distribution und -Version heruntergeladen und installiert, z.B. mittels

```
rpm~-U~NVIDIA\_GLX-1.0-4191.suse73.i386.rpm
```

und

```
rpm~-U~NVIDIA\_kernel-1.0-4191.suse73.i386.rpm
```

(die Dateinamen sind selbstverständlich durch die Namen der heruntergeladenen Dateien zu ersetzen!). Eventuell noch offene Abhängigkeiten sind dabei aufzulösen.

Hinweis: Die nVidia Dateien mit der Endung *.suse73.i386.rpm ist ausschließlich für den vorkompilierten Kernel 2.4.10-4GB gedacht. Wurde ein anderer (Sourcecode-) Kernel installiert, so muß einer der Sourcecode-Treiber (am besten der *.rpm) heruntergeladen, installiert und kompiliert werden.

Anschließend ist mit *init 3* der X-Server zu beenden; als *root* muß dann der Befehl *sax2 -m 0=nvidia* ausgeführt werden. Dieser stellt den Xserver vom nv-Treiber, der von SuSE mitgeliefert wurde, auf den nVidia-Treiber um. Die anschließend angezeigte Dialogbox bestätigt man mit *OK*, und ein *init 5* startet den neuen X-Server. In der xconsole bzw. auf dem Bildschirm, der mit STRG-ALT-F10 angezeigt wird, sollte nun eine Nachricht wie „kernel: 0: nvidia: loading NVIDIA Linux x86 nvidia.o Kernel Module 1.0-4191 Mon Dec 9 11:49:01 PST 2002“ ausgegeben werden.

Damit ist der nVidia Treiber installiert und betriebsbereit.

A.2.4 robotsoccer installieren

Als erstes muß eine aktuelle Version des Projektes im PG-Pool ausgecheckt sein. Der Befehl dazu lautet: *cvs co robotsoccer*.

Als nächstes wird im ausgecheckten Verzeichnis *robotsoccer* der Befehl *make tgz* gestartet. Die anschließende Frage beantwortet man mit ENTER. Dieser Befehl erzeugt einen Abzug der ausgecheckten robotsoccer-Version in einem tgz-Archiv. Der Dateiname lautet *src-<Datum>snap.tgz* (mit <Datum> ist das Erstellungsdatum des Archivs). Dieses Archiv ist nun in ein Verzeichnis *robotsoccer* auf das eben installierte eigene Linux-System zu kopieren (der Ordner muß gegebenenfalls vorher angelegt werden). Anschließend wird das Archiv mittels des Befehls *tar -xvzf src-<Datum>snap.tgz* entpackt. Ein simpeles *make* erzeugt nun das robotsoccer-Programm. Tritt kein Fehler auf, wird noch in der Datei *prefs.cfg* kontrolliert (und gegebenenfalls modifiziert), ob „kamera = 0“ gesetzt ist.

robotsoccer startet nun das Programm – fertig ist die Installation.

A.3 Doxygen

A.3.1 Warum doxygen ?

Doxygen ist ein Dokumentationssystem. Es kann einerseits bestehende Kommentare aus Quellcode übersichtlich darstellen (in diversen Formaten, z.B. html, pdf, usw.) und andererseits aus Quellcode direkt die Programmstruktur extrahieren und z.B. in Form von Abhängigkeitsgraphen darstellen.

Speziell die letztgenannte Möglichkeit ist nützlich, falls man ein umfangreiches wenig dokumentiertes System vorliegen hat und sich einen Überblick verschaffen möchte. In dieser Situation wurde das Tool von der PG416 eingesetzt.

Doxygen steht unter der GPL und wurde schon mit Erfolg in vielen umfangreichen Projekten eingesetzt.

A.3.2 Quick Guide

Die Beschreibung einiger nützlicher Hinweise zum Umgang mit doxygen an dieser Stelle erhebt bei weitem kein Anspruch auf Vollständigkeit. Dazu sei auf die Websites verwiesen.

Die Erstellung von Übersichten über die Codestruktur gelingt doxygen ohne weiteres Zutun.

Die Extraktion der Dokumentation erzwingt jedoch sich an einige wenige Konventionen zu halten und erlaubt dann vielerlei Klassifikationen der Kommentare (z.B. TODO- Kommentare, ausführliche Kommentare, Beschreibung von Parametern, usw.). Es ist lohnenswert dies zu nutzen, da doxygen dann übersichtlicher darstellen kann.

1. Kurzkommentare
2. Ausführliche Kommentare
3. Kommentar in der gleichen Zeile
4. Gruppen
5. Methodenparameter

A.3.2.1 Kurzkommentare

Kurzkommentare können z.B. bei der Klassendeklaration eingesetzt werden um über jede Methode kurz drüberzuschreiben, was sie tut. Das sieht dann z.B. so aus:

```
//! Ein Konstruktor.

Graph();
```

Doxygen stellt es dann so dar, dass man einen Link zum folgenden ausführlichen Kommentar erhält, sofern vorhanden.

A.3.2.2 Ausführliche Kommentare

Folgendermassen kennzeichnet man einen ausführlichen Kommentar:

```
/*!
  Eine ausführliche Klassenbeschreibung.
*/

class KlasseX
{
}
```

A.3.2.3 Kommentar in der gleichen Zeile

Will man z.B. die Datenmember einer Klasse innerhalb der gleichen Zeile kommentieren, bekommt der Kommentar einen zusatz, nämlich "<" .

```
class KlasseX
{

    int wert1,     /*!< Wert 1 detailliert */

    int wert2      //!< Wert 2 kurz
```

```
    };
```

A.3.2.4 Gruppen

Es besteht eine allgemeine Möglichkeit (semantisch Verwandtes) zu gruppieren. Dazu muss man eine Gruppe definieren:

```
/*!

   \ defgroup WichtigeKlassen Wichtige Klassen

 */
```

Und im folgenden Mitglieder zu dieser Gruppe angeben:

```
/*!
   \ ingroup WichtigeKlassen
 */

class WichtigeKlasse1 {};

/*!
   \ ingroup WichtigeKlassen
 */

class WichtigeKlasse2 {};
```

Doxygen erstellt dann eine Übersicht über all diese Gruppen (Module genannt) und zeigt einem dann auf Klick auf die Gruppe alle Mitglieder derselben an.

A.3.2.5 Methodenparameter

Eine der häufigsten Fragen ist die nach der Bedeutung von Methodenparametern. Das geht unter doxygen so:

```
/*!
  \ param a     ein Integer
  \ param s     ein konstanter Character Zeiger
  \ return      das Ergebnis

 */

int KlasseX::methode (int a, const char *s)
{
  ...
}
```

Für jeden Parameter also \ param davorschreiben und für Methodenergebnisse \ return.

A.4 Laufzeit Tools

1. Einleitung
2. Mpatrol
3. ccmalloc
4. LeakTracer
5. valgrind

A.4.1 Einleitung

Bei den hier zu beschreibenden Tools handelt es sich um genau diejenigen, die schon unter "Compiler Warnings, Debugging" auf Seite 74 genannt wurden. Es handelt sich ausnahmslos um Tools, die Robotsoccer zur Laufzeit quasi begleiten können und dabei viele Fehlerarten bemerken und im Quellcode dann lokalisieren können.

Der Sinn dieser Beschreibung ist die Hürde, die durch die auf den ersten Blick vielleicht kompliziert erscheinende Benutzung entsteht, zu nehmen.

Es finden sich deshalb kurz zusammengefasst für jedes Tool Erläuterungen zur Installation, zur Anwendung auf Robotsoccer, zu Optionen und zur (Deutung der) Ausgabe des Tools.

Will man mehr zu den Tools wissen, gebe man den Namen bei google ein und gelangt stets mit dem ersten Link auf die Hauptseite des Werkzeuges.

In allen Fällen wird nun davon ausgegangen, dass das Tool (von seiner Website) heruntergeladen und entpackt wurde.

A.4.2 Mpatrol

Ein mächtiges Tool mit vielen Optionen.

1. Installation

   Man wechsle in den build-Ordner. Dort gebe man make ein. Alle dann entstehenden libraries (alles was mit lib beginnt) kopiere man in das lokale lib-Verzeichnis (vermutlich usr/local/lib). Die mitkopierten Links müssen im Zielverzeichnis neu auf die zugehörigen libs gesetzt werden (analog

zum Quellverzeichnis). Man kopiere mpatrol, mprof, mptrace und mleak nach /usr/local/bin. Dann mkdir /usr/local/mpatrol und cp ../../src/*.h usr/local/mpatrol/ um alle header in das lokale Headerverzeichnis zu kopieren.

Mpatrol muss den Pfad zu seiner library kennen. Dazu muss mittels export LD_LIBRARY_PATH=/usr/local/lib die zugehörige Umgebungsvariable gesetzt werden. Man kann das der Bequemlichkeit wegen in die .profile schreiben.

2. Anwendung auf Robotsoccer

   Die einfachste Möglichkeit besteht darin die Dynamikoption von mpatrol zu nutzen:

   ```
   mpatrol - -dynamic ./robotsoccer
   ```

   Hinter dynamic kann man dann noch weitere der vielen Optionen angeben.

   Falls –dynamic nicht funktioniert (wird nicht immer unterstützt), dann muss man robotsoccer geeignet kompilieren. Dazu müssen in dem file makefiles/options folgende Zusätze gemacht werden:

   Unter MF2_COMPILE_OPTIONS füge man folgende 2 Zeilen hinzu:

   ```
   -include /usr/local/include/mpatrol.h
   -g
   ```

   Dies sorgt für das Einbinden des mpatrol-Headers und für die Generierung von Debuginformation.

   Weiterhin müssen zwei libraries hinzugelinkt werden. Dazu unter MF2_LINK_MISC -lmpatrol -lbfd hinzufügen.

   Hat man so neu kompiliert (ein vorheriges make clean empfiehlt sich!), dann starte man robotsoccer z.B. so

   ```
   mpatrol -C robotsoccer
   ```

   wobei -C recht viel Information generiert.

3. Ausgaben

Nachdem das Programm beendet ist (Mpatrol ist dabei recht unempfindlich gegenüber brutaler Beendigung bzw Abstürzen), finden sich die Ausgaben von Mpatrol in einer Datei, die dem Muster mpatrol.*.log entspricht.

Dabei ist * durch eine aufsteigende Nummerierung zu ersetzen, für den Fall dass man Mpatrol mehrfach benutzt.

Die konkreten Ausgaben sollten einigermassen selbsterklärend sein und geben meist die genaue Fehlerstelle im Code an.

A.4.3 ccmalloc

1. Installation

   Hier entspricht die Installation dem Standardverfahren unter Linux, d.h.: ./configure, make, make install (als root) in dieser Reihenfolge eingeben.

2. Anwendung auf Robotsoccer

   Die Datei ccmalloc.cfg enthält alle Optionen für ccmalloc. Man kopiere sie in das Verzeichnis wo das robotsoccer-Binary liegt und benenne es in .ccmalloc um. Darin ist ungewöhnlicherweise auch der wichtigste Teil der Dokumentation enthalten.

   Wichtig ist jetzt darin einen Logfile zu wählen, denn sonst wird man keinerlei Ausgaben erhalten. Für ccmalloc sind folgende Zusätze in makefiles/options unter MF2_LINK_MISC hinzuzufügen:

   ```
   /usr/local/lib/ccmalloc-gcc.o -L/usr/local/lib -lccmalloc -ldl.
   ```

   Nach make clean und make kann man dann robotsoccer wie gewohnt ausführen (also einfach mit robotsoccer starten).

3. Ausgaben

   Die Ausgaben von ccmalloc erscheinen im gewählten Logfile.

A.4.4 LeakTracer

1. Installation

   Das Programm wird möglicherweise schon mit beiliegenden Binaries ausgeliefert. Man sollte diese dennoch löschen und mittels make neu kompilieren!

2. Anwendung auf Robotsoccer

   Robotsoccer selbst muss mit -g compiliert worden sein, man prüfe dies in makefiles/options.

   Nun starte man Robotsoccer mittels LeakCheck robotsoccer.

3. Ausgaben

   Nach Beendigung des Testlaufs erhält man eine Logdatei dem Muster leak*.out entsprechend. Diese Übergebe man an leak-analyze (also z.B. leak-analyze leak.out) und erhält dann eine Übersicht über gefundene Fehler.

A.4.5 valgrind

Ein Tool, dass man unbedingt im Auge behalten sollte. Es wird sich wahrscheinlich zum Standarddebugtool entwickeln, da es mehrere andere (keines von den genannten) in sich vereint.

1. Installation

   Hier folgt wieder der Standardinstallationsweg: ./configure, make, make install (als root).

2. Anwendung auf Robotsoccer

   Robotsoccer muss nicht mit -g compiliert worden sein. Wenn man also nur kürz prüfen will, ob ein Fehler existiert, kann man valgrind direkt auf beliebige Programme anwenden. Die Angabe von -g führt aber im Falle einer Fehlerausgabe zu nützlichen Zeilenangaben.

   In jedem Fall muss man dann valgrind robotsoccer eingeben. Weitere (zur Situation passende) Optionen schlägt valgrind selbständig vor.

3. Ausgaben

   Die Ausgaben erfolgen direkt auf stdout und sind selbsterklärend.

A.5 Corporate Identity

Schon zu Beginn dieser Projektgruppe wurde von den Teilnehmern und Betreuern entschieden, das Roboterfußball-Projekt in einem offizielleren und organisatorisch besser geführten Rahmen zu präsentieren. Hierzu war es unvermeidlich das Projekt an eine Art Corporate Identity, wie es für jedes Projekt oder Unternehmen existiert, zu schaffen. Hierbei sollte es sich um ein Auftreten handeln, das schnell mit der Universität und dem Lehrstuhl I in Verbindung gebracht werden kann.

Bereits der Name "D.U.R.S.T" des Roboter-Teams schien dahingehend unpassend und nicht international genug. Somit startete die Schaffung der Corporate Identity mit der Findung eines geeigneten Namens. Nach zahlreichen Vorschlägen hierzu blieben schließlich drei Namen, über deren endgültige Auswahl erst nach Erstellung einiger Logos entschieden werden sollte. Als Namen blieben "R.I.S.C" als Kürzel für "Robotic Intelligence System of Cubes", weiterhin "I-Cubes" und schließlich "Dortmund Droids".

Von den wiederum zahlreichen Logo-Vorschlägen werden im folgenden nur jeweils zwei abgebildet.

Abbildung A.3: Vorschlag RISC-Logo

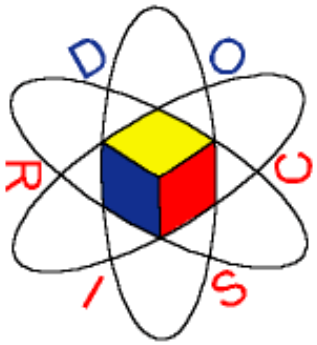


Abbildung A.4: Vorschlag RISC-Logo

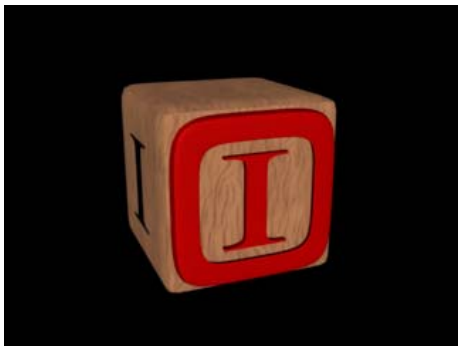


Abbildung A.5: Vorschlag I-Cubes-Logo

Abbildung A.6: Vorschlag I-Cubes-Logo

DORTMUND
ROIDS

Abbildung A.7: Vorschlag Dortmund Droids-Logo

Abbildung A.8: Vorschlag Dortmund Droids-Logo

Als Gewinner wurde schließlich folgendes Logo ausgewählt

Abbildung A.9: Gewinner-Logo

Somit wurde beschlossen, dass das Dortmunder Roboterfussball-Team ab diesem Zeitpunkt den Namen “Dortmund Droids” trägt. Um dieses Grund-Logo nun in einen Zusammenhang mit der Universität Dortmund und dem Lehrstuhl I zu bringen musste nun ein wiedererkennbares Gesamt-Logo gefunden werden in dem sich die einzelnen Logos widerspiegeln.

Von den hierzu angefertigten Vorlagen sollen im folgenden wiederum nur einige wenige abgebildet werden.

Das schließlich ausgewählte Gesamt-Logo diente zusätzlich als Vorlage für Aufkleber und T-Shirts.

Da nun ein Name und ein entsprechendes Logo gefunden wurden, musste noch zusätzlich dafür gesorgt werden, dass auch von den Mitgliedern der Projektgruppe erstellte Dokumente sich in diese Corporate Identity eingliedern. Hierfür wurden diverse Dokument-Vorlagen für Seminarband, Sitzungsprotokolle, andere Dokumente und auch für diesen Endbericht angefertigt. Beispielhaft hierfür zeigt die folgende

Abbildung A.10: Logo Universität Dortmund

Abbildung A.11: Logo Lehrstuhl I

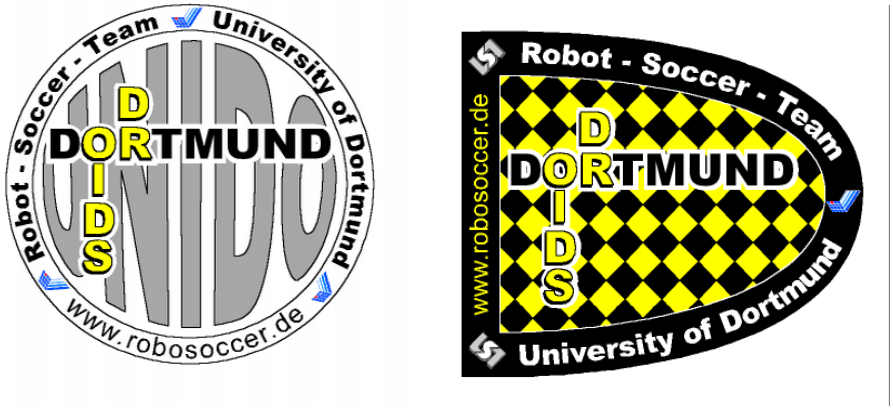


Abbildung A.12: Vorschlag Gesamt-Logo

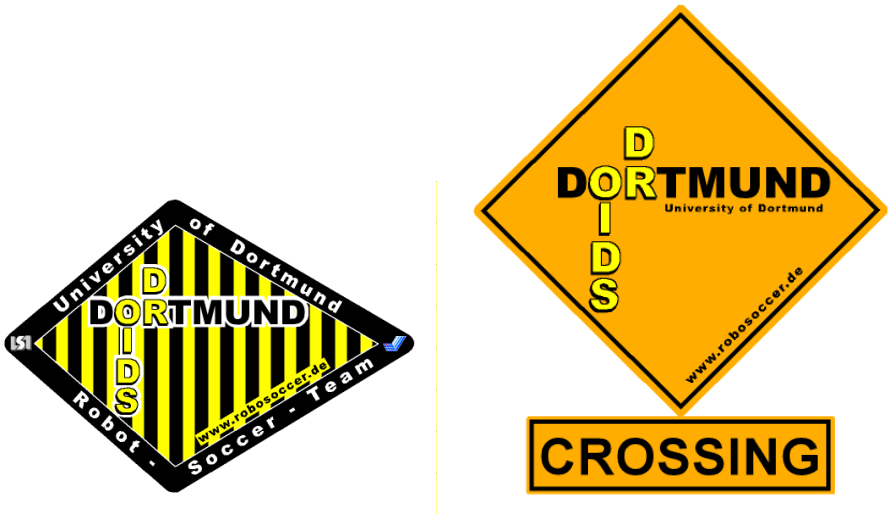


Abbildung A.13: Vorschlag Gesamt-Logo

Abbildung A.14: Gewinner-Logo

Abbildung A.15: T-Shirt-Layout

Abbildung die Vorlage für Powerpoint-Präsentationen, in die noch zusätzlich der obere Balken der Corporate Identity der Universität Dortmund übernommen wurde. Alle weiteren Dokument-Vorlagen sind in einer ähnlichen Art und Weise aufgebaut, auch die Webpräsenz der “Dortmund Droids” bedient sich dieser Elemente. Mit diesen Mitteln wurde schließlich ein einheitliches, wiedererkennbares Auftreten sämtlicher Komponenten des Projektes “Roboterfussball” des Lehrstuhls I der Universität Dortmund erreicht.

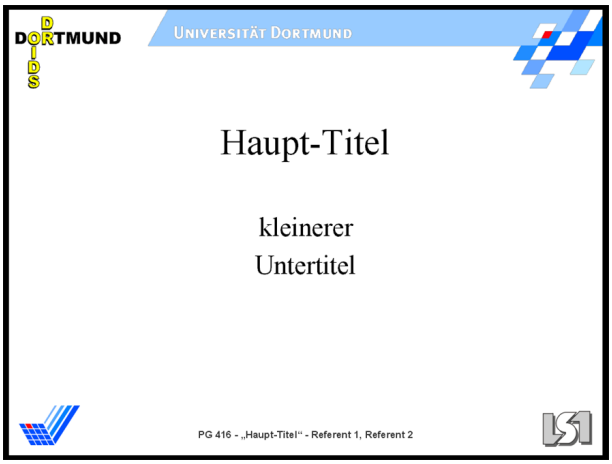


Abbildung A.16: Powerpoint-Vorlage 1

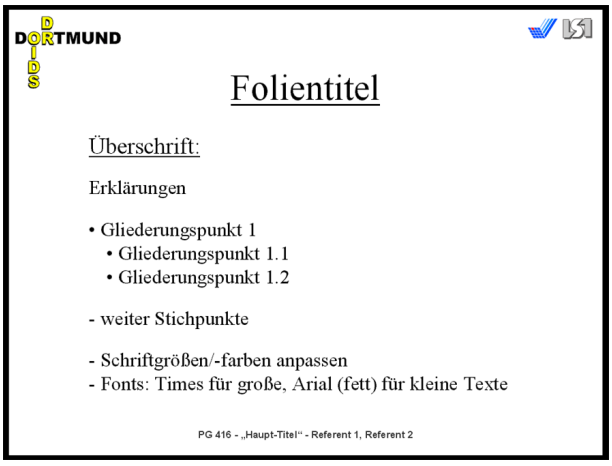


Abbildung A.17: Powerpoint-Vorlage 2

Anhang B

Checklisten

Dem Stichwortverzeichnis folgt eine komplette Anleitung zum Betrieb und zur Wartung des Robotsoccer Systems.

Literaturverzeichnis

[Lir02] Bartelmus: *LIRC Linux Infrared Remote Controll*
`http://www.lirc.org/`

[Dig] Digitus (Hrsg.): *Digitus Connect - USB/IEEE*
`http://www.digitus.info/htdoc/produkte/connect/usb_ieee.htm`

[Edu02] EducETH: *Kreisbewegungen*, 2002
`http://www.educeth.ch/physik/leitprog/kreis/anhang.html`

[Fau00] Dr. Fritz Faulhaber GmbH & Co. KG: *Magnetische Impulsgeber*, 2000
`http://www.faulhaber.de/zip_pdf/produkte/kat_2000/8_impulsgeber/dE16.pdf`

[Fen02] Fenneker: *Radgeschwindigkeiten im Kreis*, 2002
`https://goalkeeper.cs.uni-dortmund.de/durst/files/UpTables/dokumentenverwaltu/radberechnungen.pdf`

[FMS] FIRA: *FIRA Middle League MiroSot Game Rules*
`http://www.fira.net/soccer/mirosot/Middle_League_MiroSot.pdf`

[Gla93] Glaeser: *Von Pascal zu C/C++ - Gemeinsamkeiten in beiden Sprachen*, München: Markt-und-Technik, 1993

[Gor02] Gortmaker: *Real Time Clock Driver for Linux*, 2002
Zu finden in den aktuellen Linux Kernel Sourcen unter Documentation/rtc.txt

[Heu99] Heursch: *Latenzzeitmessungen des Betriebssystems Linux (Real-Time Linux/KURT Linux)*, 1999
`http://inf33-www.informatik.unibw-muenchen.de/research/dipl_heursch/dipl.html`

[Klu01] Klute: *Konzeption und Aufbau eines teilautonomen Fußballroboters*, 2001
http://robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Diplomarbeiten/diplomarbeit_klute.pdf

[Mes98] Messom: *Robot Soccer: Sensing, Planning, Strategy and Control, a distributed real time intelligent system approach*, 1998
http://www.massay.ac.nz/~chmessom/Arob98.PDF

[Moh01] Mohr: *Realisierung der Steuerung eines autonomen Kleinroboters*, 2001
http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Diplomarbeiten/diplomarbeit_mohr.ps

[PG340] PG 340: *Endbericht der PG 340*, 2000
http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Endberichte/endbericht_pg340.ps

[PG362] PG 362: *Endbericht der PG 362*, 2001
http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Endberichte/endbericht_pg362.ps

[PG384] PG 384: *Endbericht der PG 384*, 2002
http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Endberichte/endbericht_pg384.pdf

[SB416] PG 416: *Seminarband der Projektgruppe 416*, 2002
http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Seminarbaende/seminarband_pg416.pdf

[Rad02a] Radiometrix: *Low Power UHF Data Transceiver Module*, 2002
http://www.radiometrix.co.uk/dsheets/bim.pdf

[Rad02b] Radiometrix: *433MHz high speed FM radio transceiver module*, 2002
http://www.radiometrix.co.uk/dsheets/bim2.pdf

[Rad02c] Radiometrix: *869/914MHz high speed FM radio transceiver module*, 2002
http://www.radiometrix.co.uk/dsheets/bim3.pdf

[Son99] Sony (Hrsg.): *Digital Color Camera Module DFW-V500/DFW-VL500*, 1999
http://www.sony.net/Products/isp/products/1394/DFWV500.html

[Tex01] Texas Instruments: *TMS320LF/LC240x - A DSP Controllers Reference Guide System and Peripherals*, 2001
`http://www-s.ti.com/sc/psheets/spru357b/spru357b.pdf`

[Tex02] Texas Instruments: *TMS320F240 DSP Controllers*, 2002
`http://www-s.ti.com/sc/psheets/sprs042e/sprs042e.pdf`

[Tex03] Texas Instruments: *TMS320F2810, TMS320F 2812 DSP Controllers*, 2003
`http://www-s.ti.com/sc/ds/tms320f2812.pdf`

[Wei03] Weiss: *Konzeption und Implemtierung einer flexiblen und robsusten Positionserkennung für den Roboterfußball*, 2003 `http://www.robosoccer.de/fileadmin/dortmunddroids/documents/PG-Berichte/Diplomarbeiten/diplomarbeit_weiss.pdf`

Index

Anfahrtsalgorithmus, 132

Batteriewarnung, 89
Baureihe, 94
Bibliotheken
    Robotsoccer, 181
Bildverarbeitung
    Blobs, 3
    FPS, 52
    Latenzzeit, 54
    Patch, 3

Checklisten, 199

Datenpaket, 58
DIP-Schalter, 90

Farbeinstellung
    Graphen, 156
fira, 178
Funkmodul, 58

goalkeeper, 179

Hostsystem
    Compiler Warnungen, 74
    Debugging, 74

Interrupt, 90
Interrupt, Realtime, 90

kicker, 178

Kommunikation, bidirektional, 58
Korea Roboter
    Datenpaket, 102
    DeltaSpeed, 103
    Funkmodul, 102
    Rampen, 103
Kreisbogen, 138

Led, 89
Linux
    SuSE, 180
Logger, 11

nVidia, 181

Oberfläche
    Testdrive, 69

Pfadplanung, 132
    Überkreuzfahrt, 139
    Banden, 142
    Dateikonzept, 143
    Hinderniserkennung, 141
    Hindernisvermeidung, 141
    keine Tangente, 139
    Kreis im Kreis, 140
    Kreisauswahl, 133, 146
    Kreisbogen, 138
    Kreiskandidaten, 133
    Kreisorientierung, 134

Programmaufteilung, 142
Radgeschwindigkeit, 137
Radiusbestimmung, 146
Sonderfälle, 139
Strafraumvermeidung, 142
Tangente, 134
Testumgebung, 144
Umkehr, 133
vector2d, 143
Winkelberechnung, 135

Radencoder, 93
Radgeschwindigkeit, 137
Rampe, 87
Rechner
fira, 178
goalkeeper, 179
kicker, 178
soccer, 177
Robotsoccer
Bibliotheken, 181
robotsoccer, 3

Scheduling, Round Robin , 60
soccer, 177
Strategie, 9
Handlung
AnstossTick, 12
Apos, 12
AufdenBall, 13
BallBlocken, 13
BallFuehren, 14
BallvonBande, 14
Enthaken, 15
FreistossTorwart, 15
FreistossWir, 16
GegnerBlocken, 16
Geradeaus, 17
Klaeren, 17
Mittelstuermer, 17
Npos, 18
Pass, 18
PassiverGegnerBehindern, 19
Torschuss, 19
TorschussBlocken, 20
Torwartabwehr, 20
Vorstopper, 21
ZweiterTorwart, 21
Handlungen, 5, 10
Handlungsebene, 10
Rolle
AktiveAbwehr, 22
AngriffMitBall, 23
AngriffsPositionierung, 24
AufdenBall, 24
Enthaken, 24
FreistossTorwart, 24
FreistossWir, 24
Geradeaus, 25
Klaeren, 25
Mittelstuermer, 25
NeutralePositionierung, 25
PassiveAbwehr, 25
Rammbock, 26
TestRolle, 26
Torwart, 26
Vorstopper, 26

ZweiterTorwart, 26
Rollen, 5, 9
Rollenebene, 10
Spielzüge, 5, 9
Spielzug
Angriff, 27
AnstossDie, 28
AnstossWir, 29
Freiball, 30
FreistossDie, 31
FreistossWir, 32
NeutralEigeneHaelfte, 32
NeutralGegnerHaelfte, 34
Verhakt, 35
Verteidigung, 35
Spielzugebene, 9
Spielzugmodell, 9
Testumgebung, 10
Zielvektor, 5

Test
Radencoder, 93
Timeout, Funk, 89