Applikation Intelligenter Systeme
Wintersemester 2005 / 2006

Semesterarbeit „Autonomer Fußballspieler“

Bearbeitet von:

Benjamin Hoepner

Maurice Hüllein

30.01.2006

1. Einleitung..... 3

2. Analyse der Aufgabenstellung.....4

3. Das Vorgängermodell.....5

3.1 Konstruktion.....5

3.1.1 Antrieb..... 5

3.1.2 Sensorturm..... 6

3.1.3 Ballkäfig.....6

3.1.4 Sensorik.....7

3.2 Programmierung.....9

3.2.1 Sensorauswertung..... .9

3.2.2 Hindernis ausweichen..... 9

3.2.3 Ball suchen.....10

3.2.4 Torsuchen.....10

3.3 Einschätzung der Lösung.....11

3.3.1 Konstruktion.....12

3.3.2 Sensorik und Programmierung.....13

4. Das Nachfolgemodell.....16

4.1 Konstruktion.....16

4.1.1 Neue Sensoren.....16

4.1.2 Stellmotoren und Ballführung.....17

4.1.3 Tragevorrichtung und Halterung für den Nachlaufball.....19

4.1.4 Zusätzliche Wartungsarbeiten.....20

4.2 Softwarearchitektur.....21

4.2.1 Modularisierung.....21

4.2.2 Einbezug der neuen Sensoren und Aktoren.....22

4.2.3 Der neue Zustand – Schuss des Balls.....23

5. Anwendung von Methoden der Künstlichen Intelligenz..... 26

6. Ausblick.....28

7. Anhang.....29

7.1 Quellenverzeichnis..... 29

7.2 RoboCup-Regeln..... 29

7.3 Sourcecode..... 32

1. Einleitung

Im Rahmen des Kurses „Applikation Intelligenter Systeme“ des Wintersemesters 2005/2006 sollte ein autonom agierender Fußballroboter gebaut und programmiert werden. Die Konstruktionsgrundlage bildete hierbei ein umfangreicher Satz Lego-Technik-Bauteile, für die Interaktion mit der Umgebung stand ein AKSEN-Board mit Sensoren und Aktoren zur Verfügung. Der Roboter sollte in der Lage sein, möglichst erfolgreich und im Einklang aller Regeln an einem gemeinschaftlichen Turnier mit den Hochschulen aus Brandenburg, Hamburg und Dortmund teilnehmen zu können.

Die Umstände des Wettbewerbes waren offiziell festgelegt und im Voraus bekannt. Sie basierten weitgehend auf den Regeln des RoboCup Junior [RoboCup]:

Je zwei gegnerische Roboter spielen auf einem durch Wände begrenzten Spielfeld. Es gilt, einen aktiv-strahlenden Infrarotball in das gegnerische Tor zu befördern. Die Tore werden dabei durch je einen Infrarot-Beacon mit unterschiedlicher Frequenz markiert.

Jeder Roboter muss im Regelfall zwei Halbzeiten mit jeweils 90 Sekunden völlig autonom und ohne Fremdeinwirkung oder Modifikation spielen. Reparaturen oder Initialisierungen sind nur in der einminütigen Pause zwischen den Halbzeiten erlaubt. Verstöße gegen diese Vereinbarungen führen, ebenso wie die beabsichtigte Beschädigung anderer Roboter oder die Manipulation von Sensorsignalen, zur sofortigen Disqualifikation.

Alle Aktionen und Konstruktionen, die nicht durch die Statuten reglementiert werden, sind erlaubt. Eine genaue Definition aller Regeln, sowie weitere Informationen über das Turnier sind unter [RoboCupFH2] zu finden.

Im Folgenden soll die Konzeption, der Aufbau und die Programmierung eines Fußballroboters dargestellt werden. Dabei handelt es sich um eine Anschlussarbeit des Projektes „Mobile autonome Systeme“ des Sommersemesters 2005, bei dem bereits ein turnierfähiges Modell entstand. Auf dessen Grundlage und den praktischen Erfahrungen des ersten Wettbewerbes war es möglich, einen erweiterten Roboter zu bauen.

2. Analyse der Aufgabenstellung

Obwohl die Umwelt des Roboters bekannt ist und permanent nur aus drei Elementen besteht, dem Spielfeld, einem Ball, sowie einem Gegner, werden viele komplexe Probleme aufgeworfen. Die grundlegende Herausforderung dieser Aufgabe besteht insbesondere in dem Transfer von der Messung unsicherer Sensorwerte zur Ansteuerung ungenauer Aktorik.

Jeder Sensor basiert auf der Messung eines natürlichen Phänomens. Dies bedeutet zwangsläufig, dass in einer realen Umgebung mit Störquellen zu rechnen ist, welche die Messwerte beeinflussen oder aktiv ausgesendete Sensorsignale überlagern können. Dies geschieht bei einem Infrarotempfänger, welcher außer dem künstlichen Signal noch Teile des Sonnenlichtes empfängt, ebenso wie bei einem Kompass, welcher durch das Magnetfeld eines Motors abgelenkt wird.

Allerdings besteht die Schwierigkeit nicht nur in der Wahrnehmung der Umwelt. Vielmehr muss der Roboter sich über die Ansteuerung von Motoren fortbewegen und auf seine Umwelt einwirken. Arbeitet er dabei ohne eine Rückkopplung, die seine Aktionen kontrolliert, muss er blind auf den Erfolg der ausgeführten Aktionen vertrauen. Hierbei stehen Befehle zur Verfügung, wie die Regulierung der Antriebsmotoren über eine Pulsweitenmodulation. Im idealisierten Falle beträgt die zurückgelegte Strecke die Umdrehungen eines Rades, welches eine bestimmte Zeit von einem Motor angetrieben wird. In der Realität haben unzählige weitere Faktoren einen Einfluss auf die jeweilige Aktion: die Beschaffenheit des Untergrundes, Schlupf der Räder, Spannungsschwankungen, ungleichmäßiger Lauf zweier Motoren etc.

Da es prinzipiell nicht möglich ist, alle Parameter der Umwelt zu kalkulieren oder unvorhersehbare Ereignisse auszuschließen, muss ein autonomes System mit unsicheren und unzuverlässigen Komponenten arbeiten können. Erschwerend kommt hinzu, dass die geforderte Aufgabe ein Echtzeitsystem voraussetzt. Der Roboter kann nur dann auf einen beweglichen Ball reagieren oder einen Wand ausweichen, wenn er seinen Blick auf die Umwelt mehrfach pro Sekunde auswertet. Alle Berechnungen müssen daher einem strengen Zeitlimit unterliegen und sich an den zur Verfügung stehenden Ressourcen orientieren.

3. Das Vorgängermodell

Wie bereits erwähnt existierte vor Bearbeitung der Aufgabe bereits ein Roboter, welcher die geforderten Ziele weitgehend erreicht hatte und sich innerhalb eines Freundschaftsturniers beweisen konnte.

Dies ermöglichte, die bis dahin umgesetzten Konzepte anhand ihres praktischen Nutzens zu beurteilen. Viele Aspekte wirkten das erste Mal auf den Roboter, etwa die konsequente Einhaltung aller Regeln, die Laufzeit über mehrere Spiele und insbesondere die Begegnung mit völlig unbekannten Gegnern.

Der folgende Abschnitt soll einen Überblick geben, wie das Vorgängermodell auf Hard- und Softwareebene aufgebaut war und eine Einschätzung aus gewonnen Erfahrungen bieten. Da der alte Roboter den Kern des neuen bildet und nur um einige Techniken erweitert wurde, kann insbesondere verdeutlicht werden, wie einige Kompromisse innerhalb der Realisierung zustande kamen.

3.1 Konstruktion

Eine der obersten Prämissen war es, den Roboter möglichst kräftig und robust zu bauen. Wie die Erfahrung schnell zeigte, kommt es während der Spielphasen häufig zu Zusammenstößen und Verkeilungen der gegnerischen Roboter. Dies bedeutet, dass das Chassis möglichst unempfindlich gegen äußere Einwirkungen gebaut werden muss. Insbesondere empfindliche Partien, wie die Radaufhängung oder Sensoren brauchen Schutz. Sollte beispielsweise bei einem Aufprall ein IR-Sensor abknicken könnte die gesamte Ballerkennung ineffektiv werden und das Spiel verloren sein. Ein Schaden am Getriebe könnte darüber hinaus die Mobilität des Roboters einschränken. Aus diesen Gründen war das Grundgerüst schlicht als quaderförmiger Käfig aufgebaut, welcher das Getriebe und die Aktorik weit in das Innere verlegt und eine höhergelegte und geschützte Plattform für das AKSEN-Board lieferte.

3.1.1 Antrieb

Bei der Art des Antriebes fiel die Entscheidung auf einen Differentialantrieb mit zwei Rädern. Auch wenn zwei separat gesteuerte Achsen eine perfekte Geradeausfahrt erschweren, so ist die Möglichkeit, auf der Stelle zu drehen, unverzichtbar: Nicht nur, weil das verhältnismäßig

kleine Spielfeld keine großen Wendekreise erlaubt. Vielmehr ist es wichtig, dass der Roboter sich möglicht über kurze Wege zum Ball oder zum Tor ausrichten kann.

Da der Roboter mit zwei Rädern nicht im Gleichgewicht bleiben konnte wurde eine besonders einfache Variante eines Nachlaufrades gewählt: Ein Pingpong-Ball wurde frei unter einem kleinen Käfig installiert, welcher ähnlich einem Schlitten vom Roboter hinterher gezogen wurde. Auf diesem bot sich so auch genügen Platz, um die Akku-Packs anzubringen und für genügend Gegengewicht zu sorgen. Alternativ hätte auch ein richtiges Nachlaufrad, wie sie etwa von Lego verfügbar sind, gewählt werden können, um die Reibung zu minimieren. Im Test zeigte sich jedoch, dass sich einfache Nachlaufräder bei Belastung nur schlecht ausrichten und eher zum Blockieren neigen, wie es auch bei Einkaufswagen häufig geschieht.

Um die geforderte Antriebskraft zu erzeugen wurden pro Rad zwei Modellbaumotoren benutzt und die Übersetzung des Getriebes auf 125:1 ausgelegt. Hierfür war jedoch viel Platz vonnöten, so dass die Motoren oberhalb der Räder montiert werden und je ein Zahnrad als reine Überbrückung dienen mussten. Zudem ergab sich die Schwierigkeit, die Motoren so zu montieren, dass sie genau in die Lego-Zahnräder greifen würden und auch bei hoher Belastung die Position halten. Als praktikable Lösung erwiesen sich Kabelbinder, die möglichst fest um einen Legoträger gezogen wurden, in Verbindung mit doppelseitigem Klebeband.

3.1.2 Sensorturm

Als nächstes wurde am hinteren Teil des Roboters ein Sensorturm angebracht, welcher zunächst nur als Träger der Torsensoren dienen sollte. An dem Turm wurde auf der Höhe der IR-Beacons ein Pappering befestigt, um die zwei IR-Empfänger etwa im Abstand von 100° einbetten zu können. Mit gewöhnlichen Legobausteinen wäre dies nur schwierig möglich gewesen.

3.1.3 Ballkäfig

Zuletzt fehlte eine Möglichkeit, um den Ball auch während des Manövrierens führen zu können. Dazu wurden zwei Führungswände an die Front des Roboters gebaut, die nach Außen

hin abgeschrägt sind, um ein Festhaken zu verhindern. Dieser Führungskäfig deckte etwa dreiviertel des Balles ab, war somit regelkonform und erlaubte eine gute Kontrolle des Balles selbst bei maximaler Dreh- und Fahrgeschwindigkeit.

Allerdings machte sich bemerkbar, dass der Ball von der Rückwand des Käfigs abprallte, wenn der Roboter zu schnell auf ihn zugefahren war. Aus diesem Grund wurde ein gepolstertes Stück Schaumstoff an die Wand geklebt, wodurch der Ball zu Genüge abgebremst werden konnte.

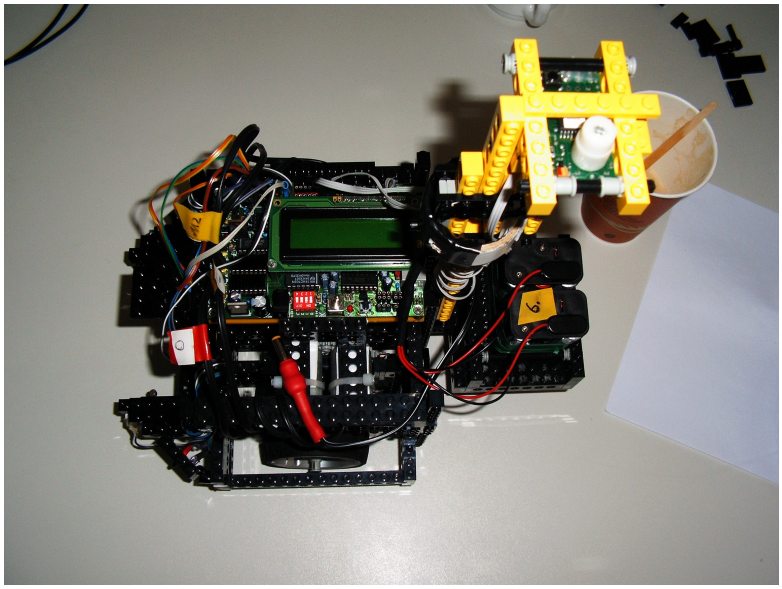

Abbildung 1: Der alte Roboter

3.1.4 Sensorik

Im Bereich der Sensorik sind zumindest drei Bereiche abzudecken: Zum einen muss der Roboter in der Lage sein, die Infrarotstrahlung des Balles und das modulierte IR-Signal der Tore zu erkennen. Zum anderen ist es jedoch auch notwendig, nicht aktive strahlende Objekte zu erkennen, um Wänden und eventuell auch dem gegnerischen Roboter ausweichen zu können.

Ballerkennung

Für die Ballerkennung wurden insgesamt fünf Infrarotempfänger an die Front des Roboters ungefähr auf der Höhe des Ballradius installiert. Drei davon wurden in den Löchern der Rückwand des Ballkäfigs untergebracht. Durch die enge Anordnung sollten sie es

ermöglichen, auch feinere Richtungsänderungen des Balles erkennen zu können, um ihn sehr genau in den Ballkäfig zu führen. Die anderen Beiden Empfänger fanden ihren Platz an den Außenrundungen der Käfigwände. Durch ihre um etwa 45° verdrehte Stellung waren sie für die seitliche Ortung des Balles verantwortlich. Auf diese Weise deckten die Empfänger schätzungsweise 120° Sichtfeld ab.

Der Standardball der Lego-Liga besteht nur aus einer begrenzten Anzahl an IR-Sendern. So kann es vorkommen, dass er kurzzeitig nicht wahrgenommen wird, sobald er rotiert oder Teile von ihm verdeckt werden. Um dennoch zuverlässig bestimmen zu können, ob der Roboter den Ball tatsächlich führt oder wieder verloren hat wurde eine Lichtschranke (IR-Sensor/Empfänger Paar) im Ballkäfig angebracht.

Die Arbeit mit unmodulierten IR-Signalen bringt stets das Problem mit sich, Störstrahlung zu erkennen und herauszurechnen. Die Überwachung der Sensorwerte zeigte überdeutlich, dass selbst geringe Veränderungen der Lichtverhältnisse, sei es die aufgehende Mittagssonne oder eine vorbeiziehende Wolke, massive Einflüsse auf die IR-Empfänger hatten. Daher wurde auf der Höhe des AKSEN-Boards ein zusätzlicher IR-Empfänger montiert, welcher nur für die Messung des Umgebungslichtes zuständig war. Damit direkte Strahlen vom Ball oder einer Abstandserkennung aus möglichst nicht die Werte beeinflussen konnten, musste er durch umgebende Bauteile etwas mehr abgeschottet werden.

Torerkennung

Beim Aspekt der Torfindung stand im Vordergrund, den Roboter so schnell wie möglich zum gegnerischen Tor ausrichten zu können. Aus diesem Grund wurde ein hybrider zweiphasiger Ansatz verfolgt: Zuerst wurde die Orientierung des Roboters von einen Acht-Wege-Kompass übernommen.

Dieser wurde aufgrund seiner Störempfindlichkeit auf die Spitze des Sensorturmes gesetzt, erst mit dieser Distanz zum Rest des Roboters arbeitete er einwandfrei. Selbst nach intensiven Versuchen konnte nicht herausgefunden werden, welches Bauteil oder welcher Effekt den Kompass regelmäßig dazu brachte, Rotationsänderungen zu registrieren.

Da die Anzahl der Direktionen nicht ausreichte, um aus jeder Position des Spielfeldes das Tor zielgenau anzusteuern, wurden zusätzlich zwei digitale IR-Empfänger benutzt. Diese sollten in der zweiten Phase eine genauere Ansteuerung auf kurze Distanzen ermöglichen.

3.2 Programmierung

Das Programm zur autonomen Steuerung des ersten Roboters trug eher provisorischen Charakter. In einer monolithischen Datei wurde eine Zustandsmaschine umgesetzt, welche anhand der Sensorwerte zwischen den wichtigsten Zuständen, Ball suchen, Tor finden und Hindernis ausweichen wechselte. Im Folgenden soll nur ein grober Überblick über den Ablauf des Programms gegeben werden, Abbildung 2 soll dies zusätzlich illustrieren. Eine detailliertere Beschreibung inklusive einer Aufstellung der Zustandshierarchie findet sich im Kapitel über das Nachfolgemodell.

3.2.1 Sensorauswertung

Zunächst galt es, über programmiertechnische Methoden die Sensorwerte möglichst präzise und zuverlässig zu interpretieren. Wie bereits angedeutet, reagierten die unmodulierten IR-Empfänger äußerst empfindlich gegenüber Störstrahlungen, wie sie bei Sonnenschein auftreten. Um diesen Effekt auszugleichen, wurden die Sensorwerte nicht ungefiltert übernommen, sondern erst mit dem Ergebnis des Umgebungslichtempfängers per Differenzverfahren verrechnet. Zusammen mit sorgfältig ausgewählten Schwellwerten zum Feuern der Signale konnten zumindest in abgedunkelten Räumen Ball und Wände gut erkannt werden.

Ein weiteres Problem bildeten schnell wechselnde Sensorwerte, die als getrennte Zustände interpretiert wurden. So konnte es beispielsweise passieren, dass der Roboter mittels seiner IR-Empfänger den Ball verfolgte, für eine kurze Zeit das Signal verlor und sofort in den desinformierten Suchmodus wechselte, der ihn vom nahe liegenden Ball wegdrehen ließ. Als Lösung wurde eine Trägheit in die Auswertung der Sensorik implementiert: Wurde der Ball oder das Tor erkannt, braucht es erst eine kurze Periode ohne Kontaktsignal, bis der Zustand gewechselt wird. Der Roboter fuhr somit erst ein kurzes Stück weiter auf die letzte wahrgenommene Position des aktiven Zieles zu, wodurch kurze Signalaussetzer häufig überbrückt wurden.

3.2.2 Hindernis ausweichen

Dieser Zustand beschreibt die Priorität des Roboters, wenn er nicht im Besitz des Balles ist. Es werden periodisch IR-Signale gesendet und empfangen, um Hindernisse zu erkennen. Für die Registrierung der Hindernisse existieren drei Positionsangaben, *LEFT* und *RIGHT*, die

Kombination beider deutet auf ein frontales Objekt hin. Darüber hinaus wird die Entfernung des Hindernisses über die Stärke des reflektierten Signals in *FAR* oder *CLOSE* eingestuft, um das jeweilige Ausweichmanöver anzupassen: Wird ein weit entferntes Objekt wahrgenommen, wendet sich der Roboter in einer länger gezogenen Kurve von diesem weg. Ist es jedoch nahe, wird mit einer scharfen Kurve reagiert.

Sobald das Hindernis nicht mehr geortet werden kann, wird in den Torsuche- bzw. Ballsuche-Zustand gewechselt, je nachdem, ob der Roboter in Ballbesitz ist oder nicht.

3.2.3 Ball suchen

Solange kein Hindernis geortet wird versucht der Roboter, den Ball über seine IR-Empfänger wahrzunehmen. Dabei ist es wichtig, dass im Entscheidungsbaum zunächst die drei inneren Sensoren ausgewertet werden und danach die beiden äußeren. Da sich die Abtastareale der Empfänger überschneiden käme es im umgekehrten Falle durch den exklusiven Entscheidungsbaum zu einem permanenten, oszillierenden Richtungswechsel: Die äußeren Sensoren würden den Ball als erste wahrnehmen und den Roboter in ihre Richtung drehen lassen. Die inneren Sensoren würden jedoch nie oder nur selten abgefragt werden, der Roboter würde somit nicht vorwärts fahren.

Wenn nun der mittlere der drei Frontsensoren den Ball registriert, wird direkt auf ihn zugesteuert. Die anderen beiden dienen in erster Linie dazu, Abweichungen vom Kurs wahrzunehmen und über die Motorsteuerung auszugleichen. Dies passiert regelmäßig, entweder, wenn sich der Ball bewegt oder weil das Differentialgetriebe zu ungleichmäßig fährt.

3.2.4 Torsuchen

Sobald der Roboter sich im Besitz des Balls wähnt, dreht er sich in die Richtung des Tores und steuert darauf zu. Da die Findung des Tores zweiphasig realisiert ist, muss erst entscheiden werden, ob das Tor in unmittelbarer Sensorreichweite ist (Reichweite < ~70cm). Dies würde bedeutet, dass zumindest einer der beiden IR-Empfänger ein klares Signal der Tor-Beacons erhält. Der Roboter wird sich so lange auf der Stelle ausrichten, bis beide Empfänger Kontakt signalisieren. Das Tor befindet sich dann vor dem Roboter und kann direkt angefahren werden.

Sollten die IR-Sensoren jedoch das Tor nicht wahrnehmen können, wird der Kompass eingesetzt. Dieser muss vor jedem Spiel über einen Druckknopf manuell kalibriert werden, die Blickrichtung des Roboters gilt dabei stets als Norden. Da die Regeln besagen, dass die Roboter zu Beginn des Spieles zum gegnerischen Tor ausgerichtet werden muss, kann dieses über die Richtung des internen Nordens schnell angesteuert werden. Sobald jedoch wieder IR-Kontakt besteht, wird auf diese feinere Art der Navigation umgeschaltet.

Ein weiterer wichtiger Aspekt ist, dass während seiner Torsuche jegliche Hindernisse ignoriert. Der Grund hierfür war die Notwendigkeit des Roboters, den Ball mangels einer Schießvorrichtung in das Tor zu schieben. Weder die Rückwand des Tores, noch ein verteidigender Roboter sollten die Anfahrt verhindern.

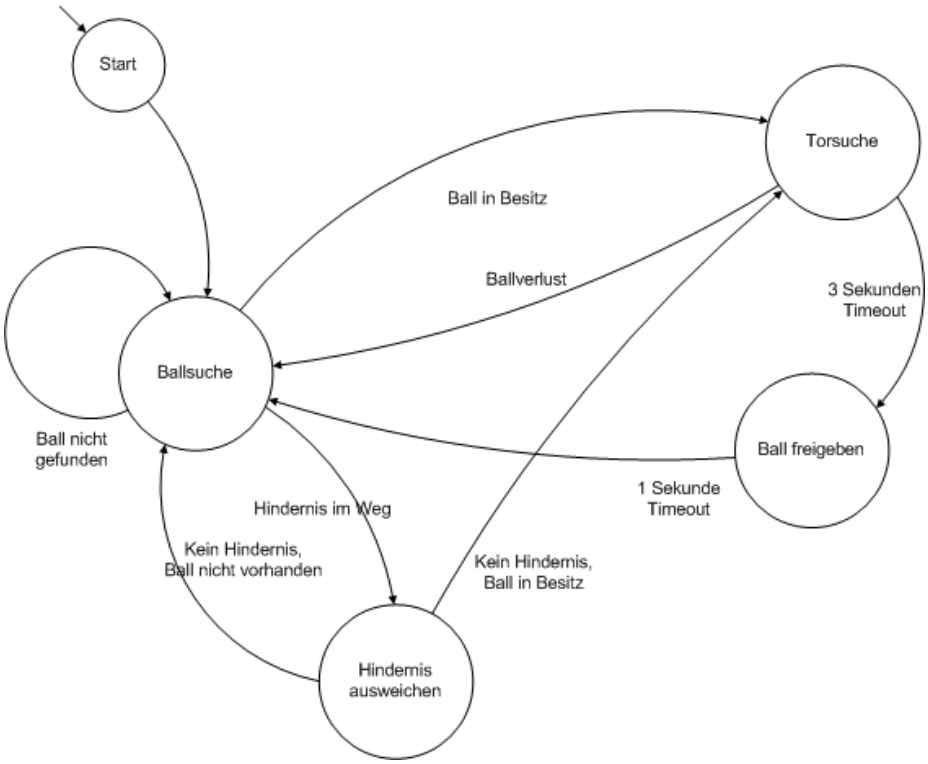

Abbildung 2: Der Zustandsgraph

3.3 Einschätzung der Lösung

Durch das Turnier 2005 konnten die bis dahin umgesetzten Konzepte auf ihre Tauglichkeit untersucht werden. Besonders interessant war, dass viele der vermeintlichen Schwächen während der Spielphasen kaum ins Gewicht fielen.

3.3.1 Konstruktion

Ein sehr gutes Ergebnis wurde mit der Konstruktion des Chassis erzielt. Der Roboter zeigte sich besonders robust gegen die regelmäßigen Zusammenstöße. Selbst, wenn die gegnerischen Roboter mit voller Kraft ineinander fuhren und sich verkanteten, hielten alle Bauteile zusammen. Zudem war zu beobachten, wie die Kraft der vier kombinierten Modellbaumotoren anderen Robotern weit überlegen war: Im Zweifelsfall wurden sie einfach beiseite geschoben oder aus dem Tor gedrängt.

Dies ist ein Gesichtspunkt, der nicht unterschätzt werden darf. Da kein einziger Roboter komplexe Spielstrategien implementiert hatte, verlief der Beginn einer jeden Runde gleich. Beide Teilnehmer fuhren in höchster Geschwindigkeit vorwärts und versuchten den in der Mitte liegenden Ball zu ergattern. Erfolgreich war hierbei entweder der schnellere oder der kollisionsfestere.

Größere Bedenken herrschten bei der Geschwindigkeitsleistung des Roboters. Aufgrund seiner vier großen Motoren, den zwei Akkupacks und verhältnismäßig viel Legokonstruktion gehörte er zu den schwersten Teilnehmern. Dementsprechend existierten viele Konkurrenten, die schneller beschleunigen, fahren und drehen konnten. Zusätzlich hatte der Roboter keine Schussvorrichtung, wodurch auch die Distanz bis zum Tor abgefahren werden musste.

Es stellte sich jedoch heraus, dass dieser Malus nur gering wirkte. Bis auf die erwähnte Startphase gab es nur wenige Spielmomente, in denen die Manövriergeschwindigkeit eine außerordentliche Rolle gespielt hatte. War ein Roboter in Ballbesitz, ergab sich weniger das Problem, diesen einzuholen. Vielmehr verlor der Gegner Sensorkontakt zum Ball und machte sich erst wieder desinformiert auf die Suche.

Dennoch hätte eine Kombination aus Schussvorrichtung und Führungskäfig einen deutlichen Vorteil gebracht. Nicht nur, um bereits aus der Distanz zum Tor zu schießen, auch um den Ball aus schwierigen Situation herauszugreifen. Mit einem solchen Hebearm wäre es möglich gewesen, einen Ball, der in einer Ecke oder zu dicht an einer Wand liegt, einfach zu umschließen und via Rückwärtsfahrt zu befreien.

Völlig unerwartet gab es große Probleme durch die Abstinenz eines Hebegriffes für den Roboter. Während der Spiele musste häufig der Schiedsrichter eingreifen und die Kontrahenten neu positionieren. Da der Unparteiische nicht wusste, wo der Roboter am

besten zu tragen war, kam es einige Male zur Beschädigung: Legoteile knickten ab, Sensoren verrutschten oder die Motoren wurden verdreht und griffen nicht mehr sauber in das Getriebe.

Zudem war es notwendig, stets ein Auge auf den Nachlaufball zu haben, da dieser nicht befestigt war und beim Anheben des Roboters liegen blieb.

3.3.2 Sensorik und Programmierung

Bis auf wenige Ausnahmen zeigte das Turnier, dass entscheidende Mängel eher auf Seiten der Software und der Sensorauswertung zu suchen waren. Einige der gegnerischen Roboter rührten sich nicht oder nahmen den Ball bzw. das Tor nicht wahr.

Die Bemühungen um die Filterung der Sensorwerte durch das Differenzverfahren und die Anordnung der Sensorik machten sich hingegen bezahlt. Bis zum Ende des Turniers funktionierten die Erkennungsmechanismen wie erwartet: Der Ball wurde grundsätzlich erkannt, solange der Roboter ungefähr auf diesen ausgerichtet und angenähert war. Besonders effektiv funktionierte die hybride Torerkennung. Der Kompass unterlag keinen Störungen und ermöglichte eine sofortige Ausrichtung, durch die IR-Sensoren konnte das Tor dann nahezu zentriert angefahren werden.

Der Vorteil dieses Ansatzes wurde besonders im direkten Vergleich mit anderen Robotern deutlich, die sich auf eine Art der Erkennung verließen. Ohne Kompass verging häufig viel Zeit, bis der Kontakt zum Tor hergestellt war. Ein Roboter, der nur einen Kompass nutzte, schoss mehrfach gegen die Kante des Tores, weil die Ausrichtung zu ungenau war.

Während der finalen Runden des Turniers spielten alle verbliebenen Roboter völlig unberechenbar. Der Einfall der Nachmittagssonne hatte offenbar die IR-Sensorik massiv gestört. Es wurde Wänden ausgewichen, wo keine waren und auch der Ball wurde oft ignoriert. Diese Einwirkung war ein Desaster, da der Verlauf der Spiele eher zufällig entschieden wurde. So blieb für eine Weiterentwicklung die Frage, in wie fern die Sensorfilterung noch verbessert werden könnte oder ob eine Steigerung der Zuverlässigkeit nur durch den Einsatz von hochwertigeren oder andersartigen Empfängern möglich wäre. So gab es Überlegungen, Sonar einzusetzen und der gesamten Problematik von Lichtinterferenzen auszuweichen. Diese Idee wurde jedoch verworfen, da die verfügbaren Ultraschallsensoren nicht auf die Messung von sehr kurzen Distanzen (erst ab ~ 30cm)

ausgelegt sind. Der Roboter sollte jedoch auch in der Lage sein, nahe liegende Hindernisse wahrzunehmen.

Wie bereits angesprochen verkeilten sich die Roboter im Kampf um den Ball sehr häufig miteinander. Dieser Zustand konnte jedoch nicht erkannt werden, zum einen, da die sehr löchrigen Lego-Chassis der Roboter zu wenig Reflektionsfläche für die Abstandssensoren boten. Zum anderen war keine Odometrie implementiert, um den Stillstand des Roboters zu bemerken. Die entscheidende Frage ist jedoch, wie der Roboter hätte reagieren sollen, wäre er mit dieser Technik ausgerüstet gewesen.

Angenommen, beide Roboter hätten kurz nach dem Start einer Runde versucht, den Ball zu bekommen, blockieren sich nun aber gegenseitig. Durch die Odometrie stellt das Programm fest, dass eine Bewegung nach vorn oder zur Seite nicht möglich ist. Die einzige Handlung wäre nun, die Situation durch einen Rückzug nach hinten aufzulösen. Auch wenn das Spiel dadurch sinnvoll weitergeführt werden kann, so würde dem gegnerischen Roboter der Zugriff auf den Ball freigegeben werden. Sinnvoller erscheint es hier, auf diese zusätzliche Technik zu verzichten und den Schiedsrichter beide Roboter in eine neutrale, gleichwertige Lage bringen zu lassen.

Ein zweiter Ansatz wäre, Zusammenstöße zu vermeiden, indem einfache Spielzüge programmiert würden. Voraussetzung hierfür wäre die Fähigkeit, den gegnerischen Roboter beispielsweise über eine ausgefeilte IR-Phalanx oder Ultraschallsensoren wahrnehmen zu können. Erst dies würde ermöglichen, Kontrahenten gezielt zu umfahren, statt blind in jene hineinzufahren.

Solche programmierten Verhaltensmuster würden jedoch die Komplexität des Programms und den damit verbundene Zeitaufwand um ein vielfaches erhöhen. Damit sie funktionieren, müsste nicht nur der gegnerische Roboter als bewegliches Ziel interpretiert werden, sondern auch eine Einschätzung des Weges getroffen werden. Ansonsten könnte es passieren, dass der Roboter sich vom Kontrahenten in eine Ecke oder vom Tor wegdrängen lässt. Vielleicht wäre es in gewissen Situationen sogar der erfolgreichste Weg gewesen, wieder auf alle Taktiken zu verzichten und den Gegenspieler einfach beiseite zu schieben.

Zusammenfassend kann gesagt werden, dass der Roboter gemessen an der Entwicklungszeit und im Vergleich mit den anderen Entwicklungen hervorragende Ergebnisse erzielte. An der Konstruktion fehlte in erster Linie eine Schussvorrichtung, um Geschwindigkeitsdefizite auszugleichen und mehr Manöver mit dem Ball zu erlauben.

Das Programm funktionierte ohne kritische Fehler, Weiterentwicklungen waren nur im Sinne von Erweiterungen, nicht aber in Änderungen oder gar Fehlerbehebung zu sehen. Einziger Kritikpunkt war die formale Strukturierung, für zukünftige Projekte sollte ein modular-orientierter Aufbau dienen.

4. Das Nachfolgemodell

Aus den Turniererfahrungen ging hervor, dass ein neues, überarbeitetes Robotermodell die Schwächen des Vorgängers beheben und die Aktionsmöglichkeiten erweitern sollte.

Es wurde entschieden, das vorhandene Modell auszubauen und zu erweitern, da die komplette Neukonstruktion einen immensen Zeitaufwand bedeutet hätte. Allerdings hatte die Überarbeitung nicht nur Vorteile. Es musste auf beschränkende Gegebenheiten des Vorgängers eingegangen werden. Allein aufgrund seiner Größe durften zusätzliche Anbauten auch nicht über seine Grundfläche hinausragen.

Die Software-Seite des Nachfolgers sollte ebenfalls optimiert werden. Komplette Modularisierung, Strategieverbesserungen und neue Aktionsvielfalt durch erweiterte Hardware waren die Ansatzpunkte in diesem Bereich.

4.1 Konstruktion

Der Roboter sollte die vorhandenen Sensoren und Aktoren weiterhin benutzen können, es sollten aber neue hinzukommen. In den folgenden Unterkapiteln werden die vorgenommenen Änderungen detailliert erläutert.

4.1.1 Neue Sensoren

Die Ballsensoren an der Front des Roboters sind weiterhin der Hauptbestandteil zur Bestimmung der Position des Balls, sie wurden aber durch fünf weitere Infrarotsensoren im Heck erweitert, von denen einer als zweiter Umgebungslichtsensor fungiert (s. Abb.3). So ist es möglich, den Ball auch wahrzunehmen, wenn er sich hinter dem Roboter befindet. Dann kann sich der Roboter den Ball über eine schnelle 180°-Drehung anvisieren und eventuell einfangen. Diese Änderung war notwendig, weil der Schiedsrichter im letzten Turnier im Falle einer Verkeilung zweier Roboter diese mit der Front gegen die Außenwand des Spielfeldes abgestellt hat, um den Ball zwischen ihnen zu platzieren. Auf diese Situation kann so nun entsprechend reagiert werden.

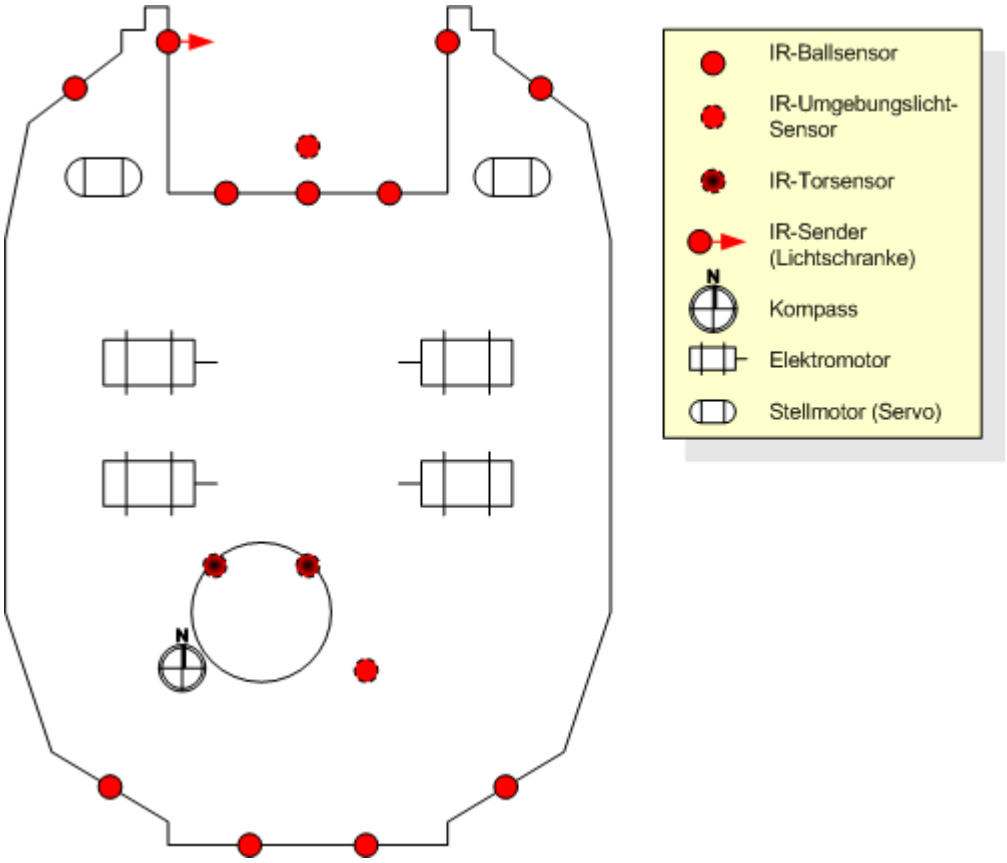

Abbildung 3: Schematische Anordnung der Aktoren und Sensoren

4.1.2 Stellmotoren und Ballführung

Der Roboter sollte mit einer Schussvorrichtung für den Ball ausgestattet werden. Außerdem sollte es ermöglicht werden, den Ball auch zu erreichen, wenn er direkt an einer Außenwand des Spielfeldes befindet. Zu diesem Zweck wurde ein durch zwei Stellmotoren nach oben und unten beweglicher Arm konstruiert, der beide Anforderungen gleichzeitig erfüllen soll. Er wurde, wie auf Abbildung 4 zu erkennen, über der alten Ballhalterung angebracht.

Der Ball kann in der Halterung eingeklemmt werden, indem sich der Arm von der oberen in die Mittelstellung bewegt und dort verharrt. Eventuell ist es so sogar möglich, mit dem gefangenen Ball rückwärts zu fahren. Zur Ausführung eines Schusses muss sich der Arm in der unteren Stellung befinden. Sobald sich der Ball in der Halterung befindet, muss der Arm schnell nach oben bewegt werden, um den Ball mit einem kräftigen Stoß zu versehen, der ihn vorwärts bewegt.

Zuerst wurde nur ein einzelner Servomotor an das Chassis angebracht, der mit einer 2:3-Übersetzung an der Achse des Arms wirkte. Nach einigen Probedurchläufen ergab sich, dass

zwar die Kraft genügte, aber die Abstoßgeschwindigkeit nicht ausreichte. Der Ball bewegte sich leider sehr langsam vorwärts und erreichte so eine Rollstrecke von nur etwa 40 bis 50cm. Deshalb wurde die Übersetzung auf 3:1 umgestellt und ein zweiter Servomotor wurde auf der anderen Seite des Chassis angebracht, um auch die nötige Schusskraft zu gewährleisten. Die Befestigung der Stellmotoren stellte sich übrigens als sehr problematisch heraus. Sie konnten durch die Konstruktion des Vorgängermodells nicht im Inneren des Chassis fest angebracht werden. Deshalb wurden sie an zwei vorderen Trägern mit Teppichklebeband befestigt und mit Kabelbindern fixiert. Leider war dieses Ergebnis noch nicht ausreichend, denn stieß der Arm während einer Bewegung auf einen festen Widerstand, so drehten sich die Kunststoffträger durch die angelegte Kraft der Stellmotoren zur Seite und die treibenden Zahnräder sprangen aus dem Getriebe. Als Behelf wurden dicke Querstreben an den Seiten des Roboters angebracht, die die Bewegung der Träger verhinderten. Da sich in der weiteren Entwicklungsphase keine weiteren Alternativen boten, wurde die recht gut funktionierende Notlösung beibehalten und bleibt auch im finalen Design erhalten. Mit den durchgeführten Änderungen ist es nun möglich, den Ball über Strecken von weit mehr als der halben Spielfeldlänge zu befördern.

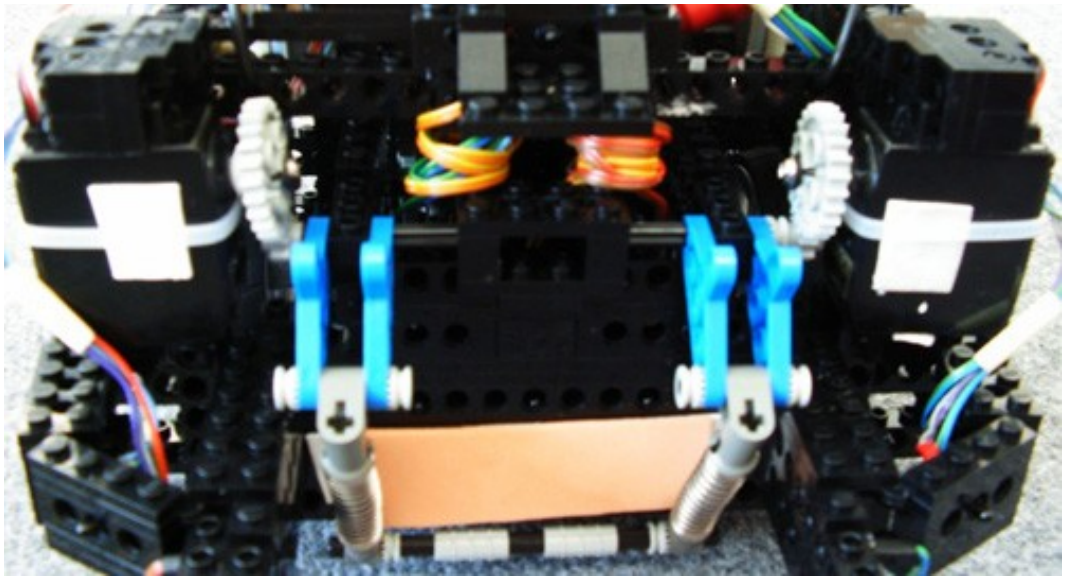

Abbildung 4: Die Stellmotoren

4.1.3 Tragevorrichtung und Halterung für den Nachlaufball

Um eine bessere Handhabung des Roboters z.B. durch den Schiedsrichter zu gewährleisten, wurden ein Tragegriff und eine Halterung für den Nachlaufball zusätzlich angebracht.

Die Tragevorrichtung besteht aus zwei Stück stabilen Drahts und einem Lego-Kunststoffteil, das als Griff dient. Die Drähte wurden durch Löcher des Kunststoffteils gezogen und auf einfache Weise an vier Eckpunkten des Chassis befestigt. Anschließend wurden sie so zurechtgebogen, dass sie keinen der Sensoren behindern, trotzdem aber für eine ausgewogene Gewichtverteilung sorgen und gut erreichbar sind.

Der Nachlaufball des Roboters wurde durch eine zusätzliche Halterung zwischen zwei Achsen eingeklemmt und, wie auf Abbildung 5 zu sehen, so unter dem Chassis fixiert. Das hat den Vorteil, dass er beim Anheben des Geräts nicht auf der Unterlage liegen bleibt, sondern immer in der richtigen Position verankert ist. Leider ist hiermit auch ein Nachteil verbunden. Da der Tischtennisball eingeklemmt ist, kann er sich nur noch sehr schwerfällig drehen und führt so zu einer leichten Reduktion der Höchstgeschwindigkeit.

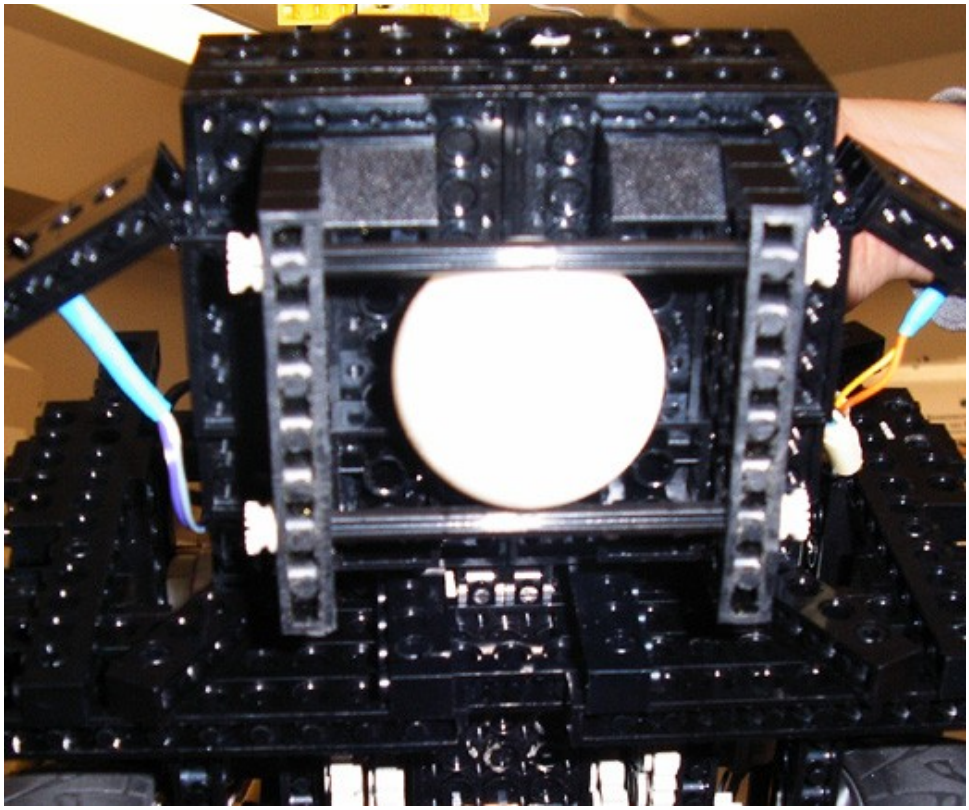

Abbildung 5: Befestigung des Nachlaufballes

4.1.4 Zusätzliche Wartungsarbeiten

Bei einigen Testdurchläufen wurde festgestellt, dass eines der Getriebe für den Antrieb schwergängig und somit der Geradeauslauf des Roboters sehr beeinträchtigt war. Daraufhin wurde entschlossen, das Getriebe komplett zu demontieren, um die Schwachstelle ausfindig zu machen und den Fehler zu beheben. Hierzu wurde das entsprechende Rad entfernt und jedes einzelne Zahnrad von unten nach oben aus der Halterung genommen. Dabei wurde nach jedem Zahnrad überprüft, ob die Schwergängigkeit noch bestand. Bei der Untersuchung des letzten Zahnrades wurde festgestellt, dass das Problem nicht durch das Getriebe verursacht wurde, sondern dass einer der beiden anliegenden Elektromotoren blockierte. Dieser wurde daraufhin durch einen neuen ersetzt. Das Getriebe wurde wieder Stück für Stück eingesetzt, wobei einige unnötige Kleinteile weggelassen und eine zu lange Achse durch eine kürzere ersetzt wurde. Das führte insgesamt zu etwas weniger Reibungsverlust, woraufhin das zweite Getriebe auf die gleiche Art und Weise abgeändert wurde. Das Resultat ist ein deutlich verbesserter Geradeauslauf und ein etwas geringeres Geräuschaufkommen bei der Fortbewegung. Abbildung 6 zeigt die überarbeiteten Getriebe nach der Fertigstellung der Wartung.

Des Weiteren wurden viele nebenläufige Kabelstränge, die die Sensoren und Aktoren mit dem AKSEN-Board verbinden, an einigen Stellen mit Klebeband umwickelt, um sie zu Kabelbäumen zusammenzufassen. So ließen sie sich besser am Chassis verlegen und es wird ausgeschlossen, dass ein einzelnes Kabel zwischen zwei Zahnräder gerät. Jeder einzelne Kabelstrang wurde außerdem markiert, um ihm dem entsprechenden Port auf dem Board und bei Uneindeutigkeit, wie z.B. bei den Antriebsmotoren, die Polung zuzuordnen.

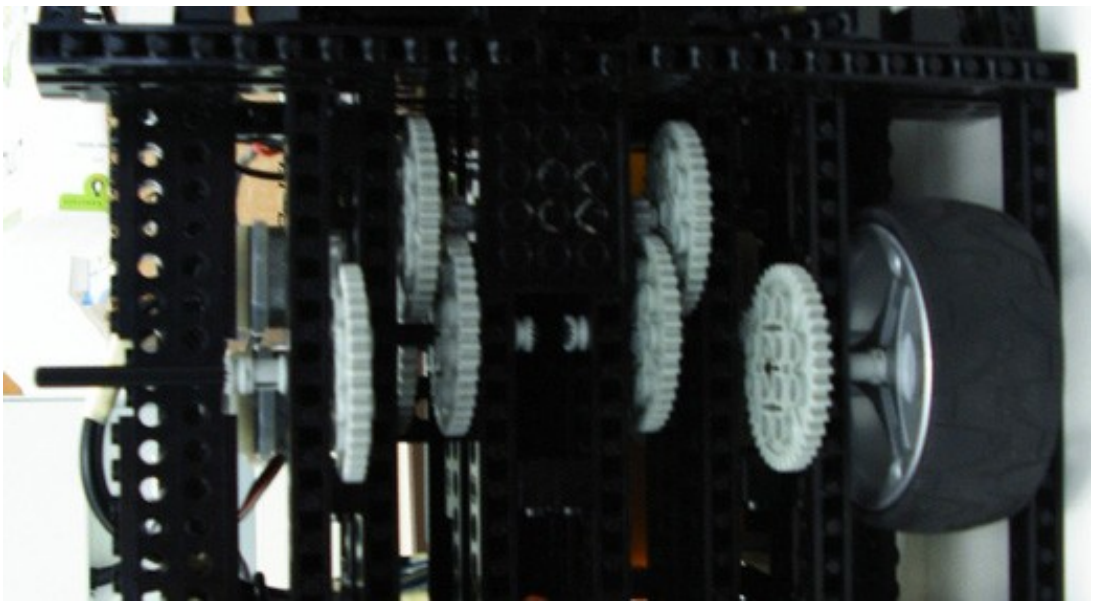

Abbildung 6: Gewartetes Getriebe

4.2 Softwarearchitektur

Es ist zu bemerken, dass sich die Software für den Roboter zum Zeitpunkt der Niederschrift dieser Dokumentation noch in der tiefsten Entwicklung befindet. Fast alle Ansätze dieses Kapitels sind somit theoretischer Natur und bisher in der Praxis noch nicht erprobt. Es kann also nicht vorbehaltlos davon ausgegangen werden, dass das finale Steuerungsprogramm mit den hier aufgeführten Ideen absolut übereinstimmt.

Durch die Änderungen an der Hardware des Roboters muss auch die Software angepasst und darauf ausgerichtet werden. Die neuen Sensoren müssen hierzu in die Balldetektion mit integriert und die Reaktionen entsprechend angepasst werden. Die Ansteuerung des Arms soll in mehreren Zuständen Beachtung finden. Ein komplett neuer Zustand muss in den endlichen Automaten eingearbeitet werden, der den Ballschuss kontrolliert. Des Weiteren wird hierbei ein besonderes Augenmerk auf die Modularisierung des Programms gelegt.

4.2.1 Modularisierung

Bei dem Quelltext des Vorgängerroboters handelt es sich um eine einzige Datei mit C-Code, in der alle Routinen zur Beschreibung des endlichen Zustandsautomaten, die Verarbeitung aller Sensoren, Aktoren und Zustandswechsel enthalten sind.

Um die Übersicht und gute Wartbarkeit der Quellen aufrechtzuerhalten, wurden mehrere Header-Dateien mit zugehöriger Implementierung erzeugt, die die gesamte Verarbeitung in Funktionsgruppen unterteilen. Daraufhin wurde auch das Makefile angepasst, da die ursprüngliche Version nur für die Kompilierung einer einzigen Datei ausgelegt war.

Die `tools`-Quellcodedateien enthalten grundlegende Funktionalitäten, wie einfache Berechnungen und die Definition des logischen Datentyps `boolean`, der ursprünglich nicht in den C-Programmbibliotheken für das AKSEN-Board vorhanden ist.

Mit den `sensor`-Dateien wird das Abrufen von den Daten aller angeschlossenen Sensoren ermöglicht. Dazu gehören alle möglichen Ball- und Umgebungslichtdetektoren, die Abstands- und Torsensorik, der Kompass, sowie die Lichtschranke zur Lagebestimmung des Balls. Außerdem werden eine Vielzahl von Schwellwerten und einige Strukturen zur Aufbereitung der Sensordaten definiert.

In den `aktor`-Dateien erlauben den Zugriff auf alle an das Board angeschlossenen Aktoren, die Antriebsmotoren und den durch Stellmotoren beweglichen Führungsarm.

Die `statemachine`-Quellcodes definieren die Struktur des endlichen Automaten. Die Verarbeitung jedes einzelnen Zustandes findet hier statt, die Wechsel zwischen den Zuständen werden unter den gegebenen Bedingungen ausgelöst.

Zu guter letzt steht die Datei `robocup.c`. Sie enthält die `main()`-Funktion und stellt somit den Eintrittspunkt der Applikation dar. Hierin finden einige Initialisierungen statt und die Hauptschleife des Programms wird ausgeführt.

4.2.2 Einbezug der neuen Sensoren und Aktoren

Die neuen Ballsensoren am Heck des Roboters werden, wie schon beim Vorgängermodell, mit Hilfe eines Umgebungslichtsensors normalisiert. Sie sind nur für den Einsatz im Zustand „Ballsuche“ gedacht. Programmtechnisch wird der Ablauf so realisiert, dass die Hecksensoren ebenfalls in der entsprechenden Routine abgefragt werden, und, falls einer von ihnen anspricht, der Roboter sich sofort um 180° um die eigene Achse dreht. Die anschließend nötige Nachjustierung der Ausrichtung läuft dann auf die ursprüngliche Art und Weise ab.

Der neue Aktor, der Führungsarm an der Front des neuen Robotermodells, hat auch einen Teil seiner Hauptfunktionalität im der Ballsuche zu absolvieren. Beim Eintritt in diesen Zustand wird der Arm in die obere Position gebracht, damit er, sobald der Ball die Lichtschranke

passiert, nach unten in die Mittelstellung bewegt werden kann, um den Ball einzufangen (s. Abb. 7). In dieser Situation ist es sogar denkbar, dass der fixierte Ball aus einer Ecke rückwärts herausmanövriert werden kann. Jedenfalls kann anschließend in den Torsuche-Zustand gewechselt werden.

Die zweite Aufgabe des Führungsarms besteht darin, den Ball zum Torschuss in Bewegung zu setzen. Dieser Ablauf ist in einem neuen Zustand realisiert, der anschließend ausführlich behandelt wird.

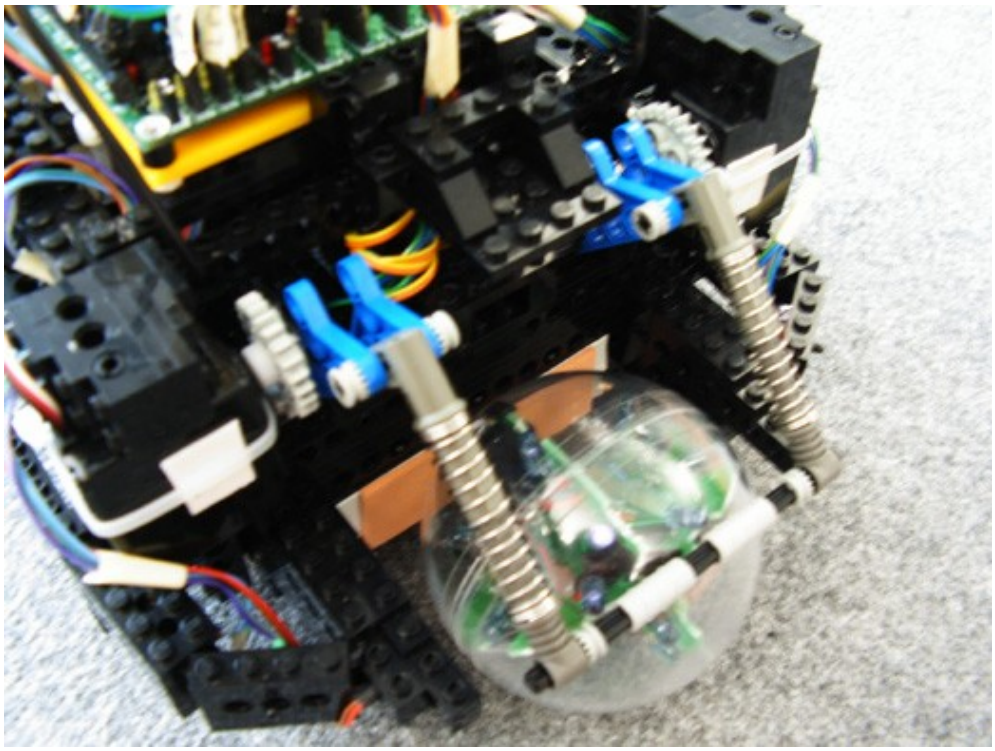

Abbildung 7: Der Ballkäfig

4.2.3 Der neue Zustand – Schuss des Balls

Der endliche Zustandsautomat, der das Hauptverhalten des Roboters darstellt, wird um einen Zustand erweitert: den Schuss des Balls, der einen komplexen Bewegungsablauf darstellt. Abbildung 8 zeigt die Anpassung des Automaten. Hier ist auch zu sehen, dass dieser Zustand auf zwei Weisen zu erreichen ist. Zum einen wird er aktiv, wenn während der Torsuche, bei der der Roboter in Ballbesitz ist, das Tor über die entsprechenden Sensoren wahrgenommen wird. Dies ist sicher die einleuchtendere Variante. Zum anderen wird versucht, einen so genannten Direktschuss zu implementieren. Dieser soll ausgelöst werden, wenn der Roboter auf Ballsuche ist, den Ball direkt vor sich aufspürt und zur gleichen Zeit das Zieltor in Sicht

gerät. Dann soll der Ball nicht erst mit dem Führungsarm in Gewahrsam genommen werden, sondern es soll sofort versucht werden, ihn auf das Tor zu schießen.

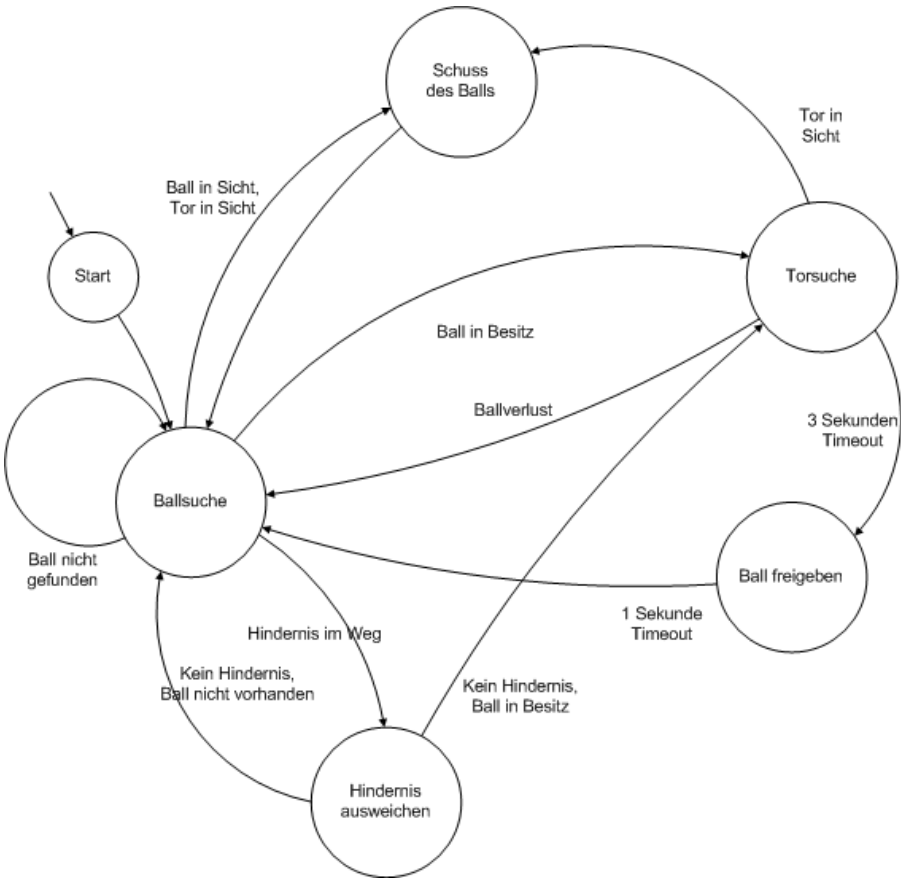

Abbildung 8: Neuer Zustandsautomat

Auf dem Programmablaufplan in Abbildung 9 ist die genaue Folge der Bewegungen zu sehen, die im Schusszustand durchlaufen werden sollen. Über die Bedingungsverzweigung am Anfang des Algorithmus wird festgestellt, aus welchem der Urzustände der Übergang stattgefunden hat, denn der Ablauf der Bewegungen ist anschließend ein wenig unterschiedlich. Ist der Ball im Besitz des Roboters, dann befindet sich der Führungsarm in Mittelstellung und der Ball ist darunter eingekeilt. Deshalb muss er in dieser Situation aus der Umklammerung befreit werden, indem der Arm nach oben ausgerichtet wird und der Roboter sich anschließend für einige hundert Millisekunden rückwärts bewegt. Anschließend entspricht die Situation derer, als wenn der Roboter zu einem Direktschuss ansetzt, deshalb geht der alternative Anweisungszweig wieder in den primären über. Der Arm wird für den Abstoß gesenkt, der Roboter fährt mit Maximalgeschwindigkeit vorwärts, um den Ball bei

Kontakt mit der vorn angebrachten Lichtschranke über eine Aufwärtsbewegung des Armes mit einen kräftigen Impuls zu versehen.

Nach der Beendigung dieses Ablaufs findet ein Zustandswechsel statt, es wird wieder auf die Ballsuche umgestellt.

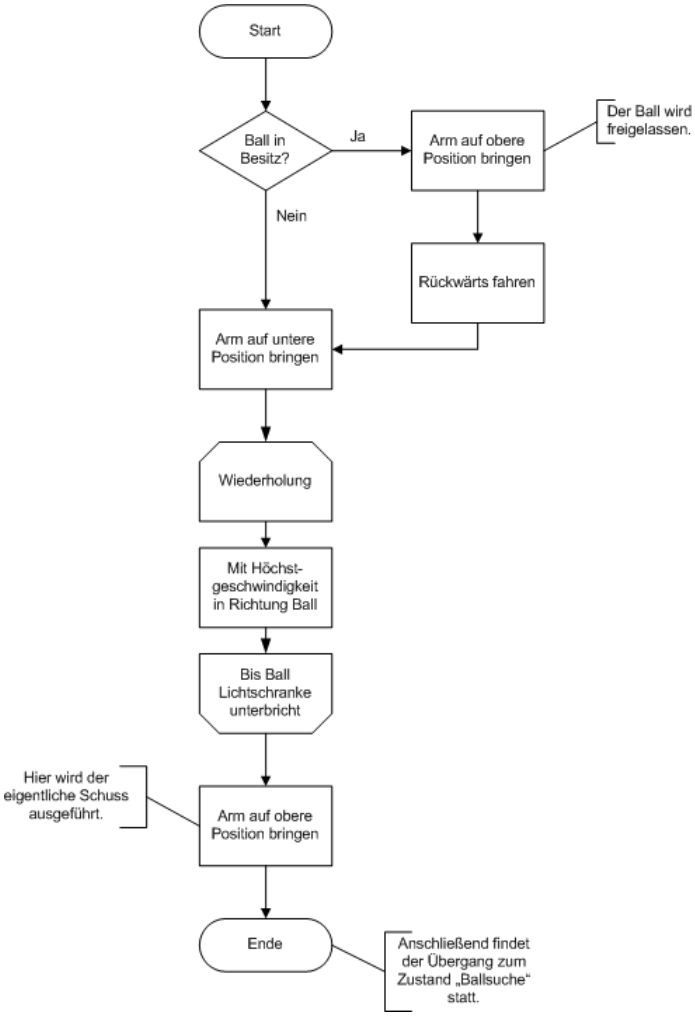

Abbildung 9: PAP des Schussablaufes

5. Anwendung von Methoden der Künstlichen Intelligenz

Beim Entwurf eines autonomen Systems steht grundsätzlich auch die Frage im Vordergrund, welche Methoden der Wissensverarbeitung angewendet werden können, um dessen Leistungsfähigkeit und Zuverlässigkeit zu steigern. Das Angebot an Möglichkeiten reicht hier von Fuzzy-Steuerung und Baysschen Netzen zum Umgang mit wagen und unsicheren Informationen bis hin zu lernenden Systemen, wie die der Genetischen Algorithmen. Dieses Kapitel soll eine genauere Einordnung des entworfenen Roboters in das Umfeld der Künstlichen Intelligenz bieten und verschiedene Ansätze zur Programmierung diskutieren.

Der Roboter, so wie er für das Turnier programmiert wurde, unterliegt dem Subsumptionsmodell nach Rodney Brooks [Brooks]. Hierbei handelt es sich um ein reines Reflexsystem, bei dem Sensorwerte direkt in Reaktionen der Aktorik umgesetzt werden. Diese Verhaltensmuster werden durch einfache endliche Zustandsautomaten beschrieben und einer Prioritätshierarchie unterworfen. Das Ziel dieses Ansatzes ist es, komplexe Abläufe durch einfache Handlungen zu erreichen, ohne auf eine interne Repräsentation der Umwelt zurückzugreifen. Dadurch werden insbesondere die Forderungen an ein Echtzeitsystem erfüllt, da keine aufwändigen Verarbeitungsschritte zwischen den Zuständen stattfinden. Auf der anderen Seite kann jedoch nicht von einem intelligenten System gesprochen werden, da weder Lernfähigkeit implementiert ist, noch die Möglichkeit mit der Umwelt zu planen oder gar optimale Lösungen zu kreieren.

Auf zusätzliche Methoden der Wissensverarbeitung wurde beim Bau des Roboters gänzlich verzichtet. Zum einen, da die zur Verfügung stehende Zeit grade einmal reichte, um Grundfunktionalitäten robust umzusetzen. Zum anderen wurde entschieden, dass der Aufwand der Implementierung und eventuell auftauchende Leistungsprobleme des AKSEN-Boards der Aufgabenproblematik nicht angemessen wären.

Beispielsweise hätte Fuzzy-Steuerung eingesetzt werden können, um die Navigation des Roboters um Hindernisse und zum Tor hin zu verfeinern. Selbst auf dem AKSEN wäre der damit verbundene Rechenaufwand vertretbar gewesen, da voraussichtlich einfache Dreiecksfunktionen für die Abstandshaltung zur Wand und die Neigung zu einem IR-Ziel ausgereicht hätten. Allerdings war in diesem geringen Vorteil kein erkennbarer Mehrwert für das derzeitige Spielniveau zu erkennen.

Auch der Einsatz von Baysschen Netzen scheint in der Subsumptionsarchitektur durchaus sinnvoll, da konkurrierende und unsichere Sensoreindrücke gegeneinander hätten abgewägt werden können. Bei der geringen Komplexität des Roboters fand diese Methode jedoch keine Anwendung, da für eine Aktion auch nur jeweils eine Sensorgattung gleichzeitig zum Einsatz kam.

Zuletzt blieb die Erwägung, lernende Strukturen einzusetzen. Ein Neuronales Netz zum Beispiel hätte wenig Sinn gemacht, da bis auf einfache Sensorwerte keine Informationen gesammelt werden, die hätten klassifiziert werden können: Weder Bildinformationen über eine Kamera, noch komplexe Spielzüge oder Gegneranalysen.

6. Ausblick

Zusammenfassend kann gesagt werden, dass der Roboter, wie er in dieser Arbeit vorgestellt wird, sein Ziel erfolgreich erreichte. Der Schwerpunkt innerhalb der Entwicklung lag deutlich auf der robusten Umsetzung einfacher Grundfunktionen und weniger auf der Einbindung komplexer Methoden der Künstlichen Intelligenz. Dadurch ergaben sich zwangsläufig nur primitive Verhaltensmuster und einfache Spielsituationen.

Um das Projekt nun auf die nächste Ebene heben zu können, müssen große Änderungen und eine Annäherung an die Techniken des offiziellen Robocups vorgenommen werden. Dies bedeutet, mehr Aspekte des Spieles wahrzunehmen, etwa den gegnerischen Roboter oder Veränderungen der Umgebungsbedingungen (Lichteinfall, Odometrie etc). Die größere Herausforderung liegt jedoch in der Erschaffung echter Interaktion. Bisweilen spielten alle Roboter, als ob sie allein auf dem Spielfeld agiert hätten. Ziel sollte es sein, Situationen zu erkennen, zu klassifizieren und nach einem Plan darauf zu reagieren. Ein Roboter könnte so entscheiden, von welcher Seite er auf das Tor schießt oder welcher Weg am geschicktesten am Kontrahenten vorbeiführt. Diese Taktiken sind mit den vorhandenen Möglichkeiten allerdings nur ansatzweise umsetzbar. Der Blick auf die großen RoboCup-Turniere zeigt, dass bildverarbeitende Systeme Pflicht sind, um das gesamte Spielfeld zu überblicken und Bewegungsrichtungen vorauszuschätzen. Ebenso müssten auf Seiten der Hardware Spezialanfertigungen für das Chassis genutzt werden (omnidirektionale Räder, stärkere Schussvorrichtungen etc), um einen deutlichen Vorteil zu gewähren.

7. Anhang

7.1 Quellenverzeichnis

[RoboCup]

RoboCup Official Site

http://www.robocup.org/

Stand des Abrufs: 2006-01-30

[RoboCupFH2]

KI-Labor, Studentenprojekte / RoboCupFH2

http://ots.fhbrandenburg.de/wiki/index.php/Studentenprojekte/RoboCupFH2

Stand des Abrufs: 2006-01-30

[Brooks]

Rodney Brooks, Subsumption Architecture

http://ai.eecs.umich.edu/cogarch0/subsump/index.html

Stand des Abrufs: 2006-01-30

7.2 RoboCup-Regeln

Regeln

Vorläufige Regeln für WS 05/06 - Allgemein

Grundregeln

(eigentlich jedem klar)

- Was nicht verboten ist, ist erlaubt.
- Der Tisch bleibt unberührt

- Es dürfen keine externen, absichtlich beigefügten Kräfte auf den Roboter einwirken (Wind etc.)
- Der gegnerische Roboter darf nicht (absichtlich) zerstört werden

Spielregeln

- Spieldauer: 2 x 90 Sek. mit Seitenwechsel

Irreführungen

- Es dürfen keine absichtlich verlorenen Teile auf dem Tisch platziert werden
- Das Signal des Balles darf nicht simuliert werden
    - Da die Roboter aus Brandenburg als Abstandssensoren mit Infrarotsendern arbeiten, heisst das hier: zusätzliche Infrarotquellen müssen gepulst werden.
- Die Torsignale dürfen nicht simuliert werden

Vorläufige Regeln für WS 05/06 - Speziell

1. Allgemeine Grundlage sind die RoboCup? Junior 2004 SOCCER Regeln (http://www.robocup.org)
2. Es wird „1 gegen 1“ gespielt.
3. Es gibt 2 Halbzeiten mit je 90 Sekunden. Die Pause zwischen den Halbzeiten beträgt maximal eine Minute.
4. Falls nach 2x90 Sek. kein Sieger ermittelt wurde, wird 90 Sekunden um ein Golden Goal gespielt, wird auch hier keine Entscheidung gefunden, folgen abwechselnd 5 Elfmeterversuche von den 5 neutralen Punkten, dabei gilt ein Zeitlimit von 20 Sekunden pro Versuch. Dabei wird der Roboter jeweils vom Schiedsrichter an beliebiger Position auf der Mittellinie gestellt – jeweils ausgerichtet auf das gegnerische Tor. Gibt es auch nach dieser Elfmeterrunde keine Entscheidung, wird gelost.
5. Jedes Team hat vor dem Spiel, sowie in der Halbzeit, max. eine Minute Zeit, den Roboter zu initialisieren, dabei gibt der Schiedsrichter an, welcher Roboter auf welches Tor schießt. Bei Spielstart wird jeder Roboter von einem Teammitglied gestartet. Das Startsignal kommt vom Schiedsrichter.
6. Beim Start und nach jedem Tor wird der Ball durch den Schiedsrichter in die Mitte des Feldes, die Roboter durch je ein Teammitglied direkt vor das eigene Tor gestellt.
7. Während des Spiels darf nur der Schiedsrichter über die Spielfeldumrandung greifen, Ausnahme sind die Initialisierung, das Starten des Roboters am Anfang jeder Halbzeit

und Neustart nach einem Tor. Greift ein Teammitglied trotzdem ein, hat die gegnerische Partei das Spiel automatisch gewonnen.

8. Die Größe der Roboter im Urzustand ist auf die Fläche einer DIN-A4-Seite beschränkt, dieses gilt auch für die beweglichen Teile – wenn ein klappbarer Greifarm installiert ist, darf der Greifarm nach dem Ausfahren maximal 5cm über das DIN A4 Blatt herausragen.
9. Über jedem Tor befindet sich ein IR-Beacon (100 Hz bzw. 125 Hz unterschiedlich für die beiden Tore). Die IR-Beacon sind jeweils 25 cm über dem Tor angebracht.
10. Bei einem Foul, Verhaken der beiden Roboter oder ähnlichem stellt der Schiedsrichter beide Roboter auf der Höhe der Verkeilung mit der Front zur Wand (sie stehen also mit dem Rücken zueinander). Der Ball wird genau zwischen die beiden Roboter platziert. Nun wird das Spiel fortgeführt.
11. Bei aktueller Langeweile darf der Schiedsrichter nach freiem Ermessen den Ball, einen oder beide Roboter auf den nächstgelegenen neutralen Punkt setzen.
12. Definition Tor: Der Ball muss die Rückwand der Toreinbuchtung berühren
13. Ist ein Roboter nicht mehr funktionsfähig, bleibt er auf dem Spielfeld stehen, bis die aktuelle Halbzeit vorüber ist.
14. Drohnen sind nicht erlaubt.
15. Zur kontrollierten Ballführung sind zwei Mechanismen erlaubt:
    - An der Front dürfen zwei Kabelbinder oder Legoleisten angebracht sein, um den Ball besser kontrollieren zu können, diese dürfen maximal 75% des Balldurchmessers lang sein. Der innere Abstand zwischen den Kabelbindern / Legoleisten darf sich nach vorne nicht verjüngen, es sind ein offener Winkel bzw. parallele Abstände erlaubt. Zusätzlich dürfen die Leisten seitlich gesehen nur so viel verdeckten, dass der Ball immer noch mindestens 50% frei sichtbar ist. Mit dieser Art von Ballführung ist eine Fahrt mit Ball erlaubt. Jede weitere Massnahme, den Ball festzuhalten (z.B. Tape), gilt als „Ballkäfig“.
    - Es darf an der Front ein Ballkäfig installiert werden, den der Roboter senken darf (und somit den Ball am Roboter sichert). Wird der Ballkäfig vom Ball entfernt, muss sich zunächst der Roboter etwa 10 cm (einen Handbreit) vom Ball entfernen, um somit dem gegnerischen Roboter die Möglichkeit zu geben, den Ball unter Kontrolle zu bringen, bevor auf das Tor geschossen wird. Wird der Ball zu lange kontrolliert, wird der Roboter ohne Ball vom Schiedsrichter willkürlich auf einen neutralen Punkt gesetzt. Der Ball bleibt am Platz liegen.

- Sobald der Roboter den Ball im Ballkäfig hat, kann dieser sich mit dem Ball maximal für 3 Sekunden frei bewegen. Nach Ablauf der 3 Sekunden muss dieser sich vom Ball eine Handbreit entfernen und 1 Sekunden stehend warten, bevor zum Schuss angesetzt wird.

7.3 Sourcecode

```
Robocup.c
#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>
#include <math.h>
#include "statemachine.h"

/**
* main routine. inits the goal frequency and schedules the
state
* machine.
*/
void AksenMain(void)
{
     initGoalSensors();

    lcd_cls();


     //state = SEARCH_BALL;
     state = SHOOT;

     hasBall = FALSE;


     while (1)
     {
```

```
            hasBall = hasBallContact();

            lcd_cls();

            switch (state)
            {
                 case AVOID_OBSTACLE:
                      lcd_puts("avoid");
                      avoidObstacle();
                      break;

                 case SEARCH_BALL:
                      lcd_puts("searchB");
                      searchBall();
                      break;

                 case SEARCH_GOAL:
                      lcd_puts("searchG");
                      searchGoal();
                      break;

                 case SHOOT:
                      lcd_puts("shooting");
                      shoot();
                      break;

                 default:
                      break;
            }
       }
}
```

Aktor.h

```
#ifndef AKTOR_H
#define AKTOR_H
```

```
#include <regc515c.h>
#include <stub.h>

// port numbers of the servos for the right/left liftarm
#define ARM_L_SERVONO        0
#define ARM_R_SERVONO        1
// servo positions for lowering boths arms
#define ARM_L_POSITION_DOWN  121
#define ARM_R_POSITION_DOWN   81
// servo positions for raising boths arms
#define ARM_L_POSITION_UP    98
#define ARM_R_POSITION_UP    101
// servo positions of the liftarms for holding the ball
#define ARM_L_POSITION_BALL 105
#define ARM_R_POSITION_BALL  92

#define FORWARD         1                  // motor direction
forward
#define BACKWARD    0                   // motor direction
backward

void setMotorDirection(unsigned char, unsigned char);
void setMotorPower(unsigned char, unsigned char);

void moveArmUp();
void moveArmDown();
void moveArmCatchBall();

#endif
```

Aktor.c

```
#include "aktor.h"

/**
* set of functions for controlling all actors
```

```
* ------------------------------------------
*/

/**
* lets the left and right motors change pairwise their
diretion.
* the directions are defined by the constans FORWARD and
BACKWARD
*/
void setMotorDirection(unsigned char left, unsigned char
right)
{
     motor_richtung(0, left);
     motor_richtung(1, left);
     motor_richtung(2, right);
     motor_richtung(3, right);
}



/**
* sets the power of the left and right motorpairs. the range
is
* between 0 (no movement) and 10 (maximum power).
*/
void setMotorPower(unsigned char left, unsigned char right)
{
     motor_pwm(0, left);
     motor_pwm(1, left);
     motor_pwm(2, right);
     motor_pwm(3, right);
}



/**
* raises the liftarms
*/
void moveArmUp()
```

```
{
     servo_arc(ARM_L_SERVONO, ARM_L_POSITION_UP);
     servo_arc(ARM_R_SERVONO, ARM_R_POSITION_UP);
}


/**
* lowers the liftarms
*/
void moveArmDown()
{
     servo_arc(ARM_L_SERVONO, ARM_L_POSITION_DOWN);
     servo_arc(ARM_R_SERVONO, ARM_R_POSITION_DOWN);
}


/**
* brings the liftarms to a position where they can hold the
ball
*/
void moveArmCatchBall()
{
     servo_arc(ARM_L_SERVONO, ARM_L_POSITION_BALL);
     servo_arc(ARM_R_SERVONO, ARM_R_POSITION_BALL);
}
```

Sensor.h

```
#ifndef SENSOR_H
#define SENSOR_H

#include <regc515c.h>
#include <stub.h>
#include "tools.h"

// maximum ir error cycles
#define IR_MAX_ERROR                    4
```

```
// frequency of both goal beacons
#define GOAL_FREQUENCY_100              5
#define GOAL_FREQUENCY_125              4

// ir thresholds for assuming an obstacle to be far or near to
the robot
#define LEFT_OBSTACLE_FAR                   12
#define LEFT_OBSTACLE_CLOSE                 80
#define RIGHT_OBSTACLE_FAR                  12
#define RIGHT_OBSTACLE_CLOSE            80
// ir thresholds for assuming a goal/wall to be detected
#define GOAL_DETECTION_THRESHOLD        4
#define WALL_DETECTION_THRESHOLD        12
// time threshold for goal contact laziness factor
#define GOAL_CONTACT_TIMEOUT            100

// port numbers for all ir sensors
// .. front ball sensors
#define BALL_SENSOR_LEFT                0
#define BALL_SENSOR_FRONTLEFT           4
#define BALL_SENSOR_FRONTCENTER         5
#define BALL_SENSOR_FRONTRIGHT          6
#define BALL_SENSOR_RIGHT               1
#define BALL_SENSOR_AMBIENT_FRONT           7
// .. rear ball sensors
#define BALL_SENSOR_REAR_LEFT           13
#define BALL_SENSOR_REAR_CENTERLEFT         12
#define BALL_SENSOR_REAR_CENTERRIGHT    11
#define BALL_SENSOR_REAR_RIGHT              10
#define BALL_SENSOR_AMBIENT_REAR            9

// ir thresholds used for assuming a ball contact
#define BALL_SIDE_THRESHOLD             4
#define BALL_FRONT_THRESHOLD            5
#define BALL_CONTACT_THRESHOLD              25
// time threshold used for ball contact laziness factor [ms]
#define BALL_CONTACT_DELAY                  250
```

```
// all possible ball positions
enum goalPosition
{
     NONE  = 0,
     FRONT = 1,
     LEFT  = 2,
     RIGHT = 4
};

// all posiible compass directions
enum compassDirection
{
     SOUTHWEST = 0,
     WEST      = 1,
     NORTHWEST = 2,
     NORTH     = 3,
     NORTHEAST = 4,
     EAST      = 5,
     SOUTHEAST = 6,
     SOUTH     = 7
};

// all possible ball positions
enum ballPosition
{
     NONE        = 0,
     LEFT        = 1,
     FRONTLEFT   = 2,
     FRONTCENTER = 4,
     FRONTRIGHT  = 8,
     RIGHT       = 16,
     REAR      = 32
};

// all possible obstacle positions
```

```
enum obstaclePosition
{
     NONE  = 0,
     LEFT  = 1,
     RIGHT = 2
};



void initGoalSensors(unsigned char irFrequency);
void getIR(int, int, unsigned char*, unsigned char*);
enum goalPosition getGoalSensor();
enum compassDirection getCompassDirection();
enum ballPosition getBallPosition();
boolean isObstacleDetected();
boolean hasBallContact();

extern enum goalPosition lastGoalContact;
extern boolean hasBall;
extern unsigned long goalContactTime;
extern unsigned long ballContactTime;
extern unsigned long ballContactTime3SR;

extern enum obstaclePosition obsPosition;
extern unsigned char obsDistance;


#endif
```

Sensor.c

```
#include "sensor.h"

/**
* provides access to all used sensors. the functions already
```

```
* interpret the data in a context, like ball/goal/obstacle
* detecting, calculating the ambient light, using the compass
* etc.
* ------------------------------------------------------------
*/



// variable for remembering the last position of a goal
contact
enum goalPosition lastGoalContact  = NONE;
// flag indicating whether the robot has the ball or not
boolean hasBall                          = TRUE;
// counter watching the time since the last goal contact
unsigned long goalContactTime            = 0;
// counter watching the time since the last ball contact
unsigned long ballContactTime            = 0;
// counter watching the time since the ball was catched
unsigned long ballContactTime3SR   = 0;
// indicating the position of a registered obstacle
enum obstaclePosition obsPosition  = NONE;
// indicating the distance of an obstacle
unsigned char obsDistance                = 0;



/**
* inits the goal sensors by setting the frequency
(GOAL_FREQUENCY_100 /
* GOAL_FREQUENCY_125) and setting their maximum error cycles
to default
* (IR_MAX_ERROR).
* the current goal is chosen by dip pin 0 (0->FREQUENCY_100,
1->FREQUENCY_125)
*/
void initGoalSensors()
{
```

```
    unsigned char irFrequency = (dip_pin(0) == 0) ?
GOAL_FREQUENCY_100 : GOAL_FREQUENCY_125;

    mod_ir0_takt(irFrequency);
    mod_ir1_takt(irFrequency);
    mod_ir3_takt(irFrequency);

    mod_ir0_maxfehler(IR_MAX_ERROR);
    mod_ir1_maxfehler(IR_MAX_ERROR);
    mod_ir3_maxfehler(IR_MAX_ERROR);
}

/**
* sends a IR signal from ports <port1> and <port2> and
measures their reflection.
* the results are mapped to the pointers <*port1Value> and
<*port2Value>, taking
* the ambient light into account
*/
void getIR(int port1, int port2, unsigned char *port1Value,
unsigned char *port2Value)
{
    unsigned char analogLight1  = 0, analogLight2  = 0;
    unsigned char ambientLight1   = 0, ambientLight2 = 0;

    ambientLight1 = analog(port1);
    ambientLight2 = analog(port2);
    led(port1, 1);
    led(port2, 1);
    sleep(10);
    analogLight1 = analog(port1);
    analogLight2 = analog(port2);
    led(port1, 0);
    led(port2, 0);
    sleep(10);

    *port1Value = ( ambientLight1 > analogLight1 ) ?
ambientLight1 - analogLight1 : 0;
```

```
     *port2Value = ( ambientLight2 > analogLight2 ) ?
ambientLight2 - analogLight2 : 0;
}



/**
* returns the position of the goal or NONE if no goal was
detected.
* it is only reported with an intended laziness if the contact
to
* the goal gets lost (GOAL_CONTACT_TIMEOUT). this way the
robot still
* advances towards the goal even if the contact is interrupted
for
* a short time.
*/
enum goalPosition getGoalSensor()
{
     unsigned char left, right;
     enum goalPosition position = NONE;

     left  = mod_ir1_status();
     right = mod_ir3_status();



     if (left > GOAL_DETECTION_THRESHOLD)
          position |= LEFT;

     if (right > GOAL_DETECTION_THRESHOLD)
          position |= RIGHT;

     if (position != NONE)
     {
          goalContactTime = akt_time();
          lastGoalContact = position;
     }

     lcd_setxy(1, 8);
```

```
    lcd_puts("G:");

    // no goal contact but the GOAL_CONTACT_TIMEOUT is not
reached yet
    // => assume the last goal position to be actual
    if ((position == NONE) && (goalContactTime > 0) &&
(akt_time() - goalContactTime < GOAL_CONTACT_TIMEOUT))
    {
        if( lastGoalContact != NONE )
            lcd_puts("1");
        else
            lcd_puts("-");

        return lastGoalContact;
    }


    if( position != NONE )
        lcd_puts("1");
    else
        lcd_puts("-");


    return position;
}


/**
* returns the actual state of the compass (8 directions).
* the default NORTH direction is calibrated at startuptime by
* a hardware trigger.
*/
enum compassDirection getCompassDirection()
{

    unsigned char compassDir = analog(8);
```

```
    if ((compassDir > 56) && (compassDir <= 76))
        return NORTH;

    if ((compassDir > 76) && (compassDir <= 98))
        return NORTHEAST;

    if ((compassDir > 98) && (compassDir <= 119))
        return EAST;

    if ((compassDir > 119) && (compassDir <= 138))
        return SOUTHEAST;

    if ((compassDir > 138) && (compassDir <= 163))
        return SOUTH;

    if ((compassDir > 163) && (compassDir <= 184))
        return SOUTHWEST;

    if ((compassDir > 184) && (compassDir <= 200))
        return WEST;

    if ((compassDir > 30) && (compassDir <= 56))
        return NORTHWEST;

    return NORTH;
}



/**
* returns the current position of the ball or NONE if the ball
can not
* be registered. to get better results and to eliminated light
interferences
* the ambient light is considered using a separate IR-Sensor.
*/
enum ballPosition getBallPosition()
```

```
{
    enum ballPosition pos = NONE;
    unsigned char ambientFront, left, frontLeft, frontCenter,
frontRight, right;
    //unsigned char ambientRear, rearLeft, rearRight,
rearCenterLeft, rearCenterRight;

    // get all sensor values
    ambientFront     = analog(BALL_SENSOR_AMBIENT_FRONT);
    left           = analog(BALL_SENSOR_LEFT);
    frontLeft      = analog(BALL_SENSOR_FRONTLEFT);
    frontCenter    = analog(BALL_SENSOR_FRONTCENTER);
    frontRight     = analog(BALL_SENSOR_FRONTRIGHT);
    right          = analog(BALL_SENSOR_RIGHT);

    // TODO: rear sensors!!!!

    // normalize values
    left        = (ambientFront > left) ? ambientFront - left
: 0;
    frontLeft   = (ambientFront > frontLeft) ? ambientFront -
frontLeft : 0;
    frontCenter = (ambientFront > frontCenter) ? ambientFront
- frontCenter : 0;
    frontRight  = (ambientFront > frontRight) ? ambientFront -
frontRight : 0;
    right       = (ambientFront > right) ? ambientFront -
right : 0;

    // check all values
    if (left > BALL_SIDE_THRESHOLD)
        pos |= LEFT;

    if (frontLeft > BALL_FRONT_THRESHOLD)
        pos |= FRONTLEFT;

    if (frontCenter > BALL_FRONT_THRESHOLD)
        pos |= FRONTCENTER;
```

```
     if (frontRight > BALL_FRONT_THRESHOLD)
          pos |= FRONTRIGHT;

     if (right > BALL_SIDE_THRESHOLD)
          pos |= RIGHT;


     return pos;
}



/***
* returns true, if an obstacle can be registered by the
sensors
* or false otherwise
*/
boolean isObstacleDetected()
{
     unsigned char leftDetectionLevel  = 0;
     unsigned char rightDetectionLevel = 0;

     getIR(0, 1, &leftDetectionLevel, &rightDetectionLevel);

     obsPosition = NONE;
     obsDistance = 0;

     if (leftDetectionLevel > LEFT_OBSTACLE_FAR)
     {
          obsPosition |= LEFT;
          obsDistance = max(obsDistance, leftDetectionLevel);
     }

     if (rightDetectionLevel > RIGHT_OBSTACLE_FAR)
     {
          obsPosition |= RIGHT;
```

```
            obsDistance = max(obsDistance, rightDetectionLevel);
        }


        if (obsPosition > NONE)
            return TRUE;

        return FALSE;
}


/**
* returns true if the ball (or at least something else) is
registered by the
* IR-barrier in the front bay, otherwise false. to get a more
accurate result
* from the sensors, the ambient light is taken into account.
* to avoid an oscillating sensor contact, a laziness-timer is
started every
* time the ball gets lost (ballContactTime)
*/
boolean hasBallContact()
{
        unsigned char analogLight  = 0;
        unsigned char ambientLight = 0;


        ambientLight = analog(14);

        led(2, 1);
       sleep(10);
        analogLight = analog(14);
        led(2, 0);
        sleep(10);
```

```
     // ir barrier is interrupted
     if (absoluteValue(analogLight - ambientLight) <=
BALL_CONTACT_THRESHOLD)
     {
          ballContactTime = akt_time();

          if (ballContactTime3SR == 0)
               ballContactTime3SR = akt_time();

          return TRUE;
     }
     // ir barrier is not interrupted, but the ballContactTime
is still used
     else if ((ballContactTime > 0) && (akt_time() -
ballContactTime < BALL_CONTACT_DELAY))
     {
          return TRUE;
     }
     // no contact at all
     else
     {
          ballContactTime3SR = 0;

          return FALSE;
     }
}
```

Statemachine.h

```
#ifndef STATEMACHINE_H
#define STATEMACHINE_H

#include <regc515c.h>
#include <stub.h>
#include "tools.h"
#include "sensor.h"
#include "aktor.h"
```

```
// all possible states of the state machine
enum machineState
{
     NONE = 0,
     AVOID_OBSTACLE,
     SEARCH_BALL,
     SEARCH_GOAL,
     SHOOT
};


void avoidObstacle();
void searchBall();
void searchGoal();
void shoot();

// the actual machine state
extern enum machineState state;


#endif
```

Statemachine.c

```
#include "statemachine.h"

/**
* Provides all important behaviours of the soccer robot,
* the interpretation of sensor data, the control of all
* actors as well as strategical issues
* ------------------------------------------------
*/
```

```
// the actual state
enum machineState state            = NONE;


/**
* defines how the robot shall avoid an obstacle like walls
* or other robots. main behaviour is to take the mirrored
* direction vector as new course.
* if no obstacle is detected the robot searches the ball
* (hasBall flag) or tries to shoot a goal (!hasBall flag)
*/
void avoidObstacle()
{
     switch (obsPosition)
     {
          case LEFT:
               if (obsDistance >= LEFT_OBSTACLE_CLOSE)
               {
                    setMotorDirection(FORWARD, BACKWARD);
                    setMotorPower(8, 8);
               }
               else
               {
                    setMotorDirection(FORWARD, FORWARD);
                    setMotorPower(8, 4);
               }

               break;

          case RIGHT:
               if (obsDistance >= RIGHT_OBSTACLE_CLOSE)
               {
                    setMotorDirection(BACKWARD, FORWARD);
                    setMotorPower(8, 8);
               }
```

```
                else
                {
                     setMotorDirection(FORWARD, FORWARD);
                     setMotorPower(4, 8);
                }

                break;

           case LEFT | RIGHT:
                setMotorDirection(BACKWARD, BACKWARD);
                setMotorPower(8, 8);

                break;
     }

     if (isObstacleDetected() == FALSE)
          state = (hasBall) ? SEARCH_GOAL : SEARCH_BALL;
}


/**
* lets the robot search the ball. if no obstacle is detected
* and the ball is not registered by the sensors, the robot
* drives forward at full speed. if the ball gets into range
* the robot turns towards it.
*/
void searchBall()
{
     enum ballPosition ballPos;

     ballPos = getBallPosition();


     if ((hasBall == FALSE) && (isObstacleDetected() == TRUE))
     {
```

```
          if (!(((obsPosition & LEFT == LEFT) && (ballPos ==
LEFT)) || ((obsPosition & RIGHT == RIGHT) && (ballPos ==
RIGHT))))
          {
               state = AVOID_OBSTACLE;
               return;
          }
     }


     if (hasBall == TRUE)
     {
          state = SEARCH_GOAL;
          return;
     }



     if ((ballPos & FRONTCENTER) > 0)
     {
          setMotorDirection(FORWARD, FORWARD);
          setMotorPower(10, 10);
          lcd_puts("FC");
     }
     else if ((ballPos & FRONTLEFT) > 0)
     {
          setMotorDirection(FORWARD, FORWARD);
          setMotorPower(7, 10);
          lcd_puts("FL");

     }
     else if ((ballPos & FRONTRIGHT) > 0)
     {
          setMotorDirection(FORWARD, FORWARD);
          setMotorPower(10, 7);
          lcd_puts("FR");
     }
```

```
    else if ((ballPos & LEFT) > 0)
    {
        setMotorDirection(BACKWARD, FORWARD);
        setMotorPower(5, 5);
        lcd_puts("left");
        sleep(150);
    }
    else if ((ballPos & RIGHT) > 0)
    {
        setMotorDirection(FORWARD, BACKWARD);
        setMotorPower(5, 5);
        lcd_puts("right");
        sleep(150);
    }
    else
    {
        setMotorDirection(FORWARD, FORWARD);
        setMotorPower(10, 10);
        lcd_puts("X");
    }


}


void searchGoal()
{

    enum goalPosition pos;
    pos = getGoalSensor();


    if( isObstacleDetected() == TRUE )
    {
        if( (hasBall == FALSE) /*|| (pos == NONE)*/ )
        {
```

```
                state = AVOID_OBSTACLE;
                return;
           }

     }


     if (hasBall == FALSE)
     {
           state = SEARCH_BALL;
           return;
     }


     if ((dip_pin(3) == 0) && (ballContactTime3SR > 0) &&
(akt_time() - ballContactTime3SR > 3000))
     {
           setMotorDirection(BACKWARD, BACKWARD);
           setMotorPower(10, 10);
           sleep(600);

           setMotorDirection(FORWARD, FORWARD);
           setMotorPower(0, 0);
           sleep(1000);

           ballContactTime3SR = 0;
           state = SEARCH_BALL;
           return;
     }



     switch (pos)
     {
           case LEFT:

                setMotorDirection(FORWARD, FORWARD);
```

```
                setMotorPower(0, 5);

                break;

           case RIGHT:

                setMotorDirection(FORWARD, FORWARD);
                setMotorPower(5, 0);

                break;

           case LEFT | RIGHT:

                setMotorDirection(FORWARD, FORWARD);
                setMotorPower(10, 10);

                break;

           case NONE:
           {
                enum compassDirection dir =
getCompassDirection();

                // rechts
                if (dir < NORTH)
                {
                     setMotorDirection(FORWARD, FORWARD);
                     setMotorPower(5, 0);
                }
                // links
                else if (dir > NORTH)
                {
                     setMotorDirection(FORWARD, FORWARD);
                     setMotorPower(0, 5);
                }
                else
                {
```

```
                    setMotorDirection(FORWARD, FORWARD);
                    setMotorPower(10, 10);
               }

               break;
          }

     } // end switch(pos)


     if( pos != NONE )
          mod_ir_an();
     else
          mod_ir_aus();

}



void shoot()
{

/*
     if( hasBallContact() )
     {
          // arm up, move backwards
          moveArm(ARM_POSITION_UP);
          sleep(1500);

          setMotorDirection(BACKWARD, BACKWARD);
          setMotorPower(10, 10);
          sleep(1500);

          // staying, arm down
          setMotorPower(0,0);
          moveArm(ARM_POSITION_DOWN);
```

```
          sleep(1500);

          // speeding forwards
          setMotorDirection(FORWARD, FORWARD);
          setMotorPower(10, 10);

          // continue as long as ball is visible but not yet
cached
          while( !hasBallContact() && getBallPosition() !=
NONE)
          {
               sleep(50);     // necessary?
          }

          moveArm(ARM_POSITION_UP);

     }

*/

     // shooting test


     while(1==1)
     {
          setMotorPower(0, 0);

          moveArmDown();
          sleep(1500);

          setMotorDirection(FORWARD, FORWARD);
          setMotorPower(10, 10);
          sleep(1500);


          moveArmUp();
          sleep(1500);
```

```
            setMotorDirection(BACKWARD, BACKWARD);
            setMotorPower(10, 10);
            sleep(1500);

        }

}
```

Tools.h

```
#ifndef TOOLS_H
#define TOOLS_H

// values used for boolean data type
#define TRUE        1
#define FALSE       0

typedef unsigned char boolean;     // emulating boolean
datatype

unsigned char min(unsigned char, unsigned char);
unsigned char max(unsigned char, unsigned char);
unsigned char absoluteValue(int);

#endif
```

Tools.c

```
#include "tools.h"

/**
* A set of tool arithmetic functions
* ----------------------------------
*/
```

```
/**
* returns the minimum of the two given numbers
*/
unsigned char min(unsigned char value1, unsigned char value2)
{
     return (value1 < value2) ? value1 : value2;
}


/**
* returns the maximum of the two given numbers
*/
unsigned char max(unsigned char value1, unsigned char value2)
{
     return (value1 > value2) ? value1 : value2;
}

/**
* returns the absolute value of the given number
*/
unsigned char absoluteValue(int value)
{
     if (value < 0)
          return (unsigned char) -value;
     else
          return (unsigned char) value;
}
```