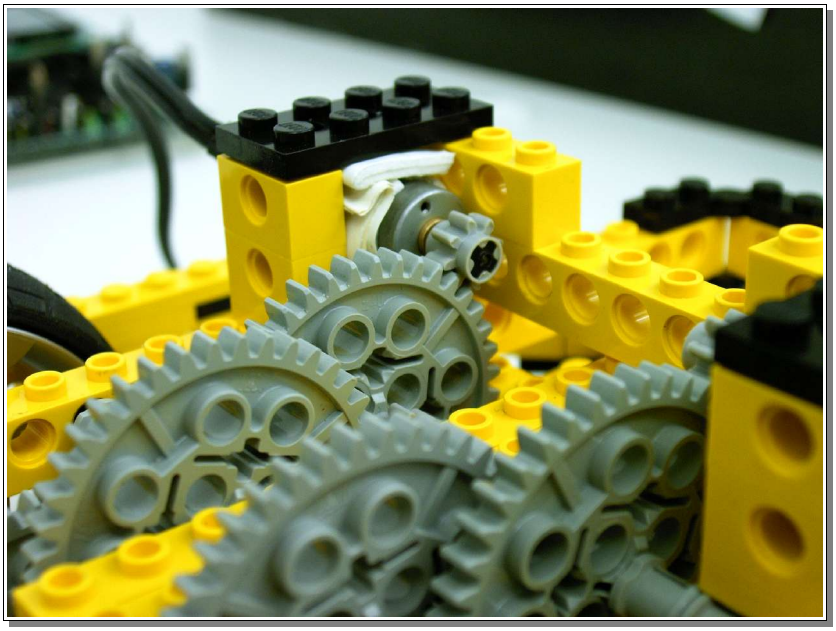

Dokumentation

„Konstruktion und Programmierung eines autonomen Fußballspielers für das Testareal im KI-Labor“

Erstellt von: Katja Orlowski (Intelligente System, 7. Semester)
Regina Wehren (Intelligente System, 7. Semester)

Fach: Applikation intelligenter Systeme

Betreuer: I. Boersch
Prüfer: I. Boersch
Prof. Heinsohn
Prof. Mündemann
Prof. Loose
C. Lemke

Semester: WiSe 2005/2006

Fachhochschule Brandenburg
Fachbereich Informatik und Medien

Inhaltsverzeichnis

1 Einleitung

Im vergangenen Semester hatten wir uns entschieden, an dem Projekt „Autonome mobile Roboter“ teilzunehmen. In diesem Projekt bauten wir einen autonomen Fußballspieler aus Legosteinen. Am Ende des Semesters, während der Projektbegehung, mussten sich dann alle Roboter in einem kleinen Wettkampf bewähren. Mit einigen kleinen Schwächen konnte sich unser Roboter namens „Yellow Panther“ gegen die anderen Roboter durchsetzen und gewann den Wettkampf. Da dieses Projekt an mehreren Hochschulen gleichzeitig läuft, wollten die Betreuer das Semester mit einem nationalen Ausscheid abschließen. Dazu kamen die Studenten der HAW Hamburg und der FH Dortmund zu uns nach Brandenburg. Die beiden Brandenburger Roboter „Böse Giraffe“ und „Yellow Panther“ konnten anfangs mit den anderen Roboter gut mithalten, jedoch reichte es bei beiden Teams nicht für das Finalspiel. Im Spiel um Platz 3 mussten die beiden Brandenburger dann gegeneinander antreten und „Yellow Panther“ besiegte „Böse Giraffe“ um Haaresbreite. Da wir von den Hamburgern zu einer Revanche eingeladen wurden, lag uns viel daran den Roboter noch stabiler zu bauen und aus den Fehlern zu lernen. In den folgenden Kapiteln werden wir die Änderungen vorstellen und genau erläutern, welche Fehler den vorgenommenen Änderungen zu Grunde liegen.

Im Folgenden werden die Aufgabe analysiert, die geltenden Randbedingungen sowie die gewonnenen Erkenntnisse aus dem vergangenen Semester beschrieben.

1.1 Analyse der Aufgabenstellung

Aufgabe ist es, einen autonomen Fußballspieler für das Testareal im KI-Labor zu konstruieren und zu programmieren. Die Programmierung des Roboters steht im Vordergrund der Aufgabe. Hierbei soll vor allem überlegt und analysiert werden, inwiefern KI-Methoden wie Fuzzy-Logik, Neuronale Netze, Bayesche Netze, Künstliche Evolution o.a. für die Lösung der Aufgabe verwendet werden können.

Der Roboter ist mit dem AKSEN-Board ausgestattet, welches so programmiert werden muss, dass der Roboter die geforderten Aufgaben erfüllen kann. Zusammenfassend kann man sagen, dass die Aufgabe des Roboters darin besteht, den Ball auf dem Feld aufzuspüren und unter Einhaltung aller Regeln diesen in das dafür vorgesehene Tor zu „transportieren“. Auf seinem Weg zum Ball bzw. zum Tor sollte er möglichst allen Hindernissen, wie beispielsweise den Wänden, ausweichen.

Für die Programmierung steht ein handelsüblicher PC zur Verfügung. Zum Kompilieren der Programme wird Cygwin genutzt und um das Programm auf das AKSEN-Board übertragen zu können, wird der AKSEN-Flasher verwendet.

1.2 Geltende Randbedingungen

Wie auch im vergangenen Jahr gelten die Regeln des Robocup (FH). Das Wiki auf der Internetseite ots.fh-brandendeburg.de steht zur Verfügung, um mit den Studenten und Betreuern aus Hamburg und Dortmund über eventuelle Probleme zu kommunizieren. Die Regeln für das Wintersemester 05/06 befinden sich im Anhang A.

1.3 Gewonnene Erkenntnisse

Da während der Wettkämpfe Videoaufzeichnungen entstanden sind, bestand im nachhinein die Möglichkeit diese auf mögliche Fehler zu untersuchen. Dies wurde ausgiebig genutzt und nach dem Anschauen und der Analyse der Wettkampfvideos ist man zu folgenden Erkenntnissen gekommen:

- Notwendigkeit von Kalibrierung: Der Roboter soll vor jedem Spiel lernen was ein Ball ist, d.h. der Schwellwert für die Ballerkennung muss dynamisch sein
- es wird ein solider Tragegurt benötigt
- Tischtennisball, welcher als Spielrad verwendet wird, darf beim Anheben nicht verloren gehen
- Ball darf Käfig nicht verlassen
- Die Bodenbeschaffenheit auf dem Spielfeld in Hamburg unterscheidet sich von der in Brandenburg. Es müssen Tests auf verschiedenen Untergründen durchgeführt werden. Erkenntnis vordergründig wichtig für den Wettkampf in Hamburg und nicht für die Prüfung
- Warum verhält sich der Roboter so? (Fährt gegen Hindernis, dreht sich weg und fährt wieder in Richtung Hinderniss – Hin- und Hergezuckel?) - Frage statischer Schwellwert
- Roboter benötigt Ballsensoren hinten (schnelle Drehung)
- Sensorik am Tragegurt, so dass Roboter bemerkt, wenn er durch Jury bewegt wird – Situation ist dann klar – Roboter steht mit Rücken zum Ball (nach Kollision / Verhaken der Roboter)
- Start im Tor -> nur hier wird Reset gedrückt -> Ball in Front -> Sprint zum Ball
- Einhalten der 3 Sekundenregel

Einige Dinge, die während der Videoanalyse aufgefallen sind, können auch dieses Mal nicht umgesetzt werden, weil einerseits die Möglichkeiten für die Umsetzung (Sensorik am Tragegurt) fehlen und andererseits der Zeitfaktor eine erhebliche Rolle spielt und somit die Umsetzung verhindert wird. Welche Punkte bei der Konstruktion und bei der Programmierung des neuen Roboters verwirklicht werden konnte, wird im Einzelnen in den Kapitel 2 und 3 näher erläutert.

2 Aufbau

Dieses Kapitel befasst sich mit der Konstruktion des Roboters sowie mit der verwendeten Sensorik und Aktorik zur Lösung der gestellten Aufgabe. Bei der Beschreibung der Konstruktion wird vor allem darauf eingegangen, wie sich der Roboter von seinem Vorgängermodell unterscheidet und warum diese Änderungen vorgenommen wurden. In Abschnitt 2.2 wird dann vertiefend auf die Veränderungen der Schussanlage eingegangen. Der Abschnitt 2.3 gewährt Einblicke in die verwendeten Sensoren und Aktuatoren. Ihre Einsatzgebiete sowie ihre Rolle bei der Bewältigung von Teilaufgaben werden erläutert. Bei den Teilaufgaben geht es um die Ball-, Wand- und Torerkennung. Die Vorgehensweise beim Lösen dieser Teilaufgaben wird ausführlich beschrieben.

2.1 Änderungen gegenüber dem Vorgängermodell und ihre Gründe

Nach der Analyse der Wettkampfvideos wurde entschieden, den Nachfolgeroboter ein wenig um zu gestalten. Zu den gravierendsten Designänderungen zählte vor allem, dass der neue Roboter etwas kürzer und schmaler gebaut werden musste, da sich die Größe am Regellimit bewegte. Um die Größe zu verringern, war es notwendig, die Zahnräder des Getriebes anders anzuordnen. Im Vorgängermodell waren die Zahnräder in einer Ebene angebracht worden. Um hier nun Platz einsparen zu können, musste versucht werden, über mehrere Ebenen die Zahnräder anzubringen.

Bei dem neuen Roboter sollte außerdem die Schussanlage ausgetauscht werden, weil auch die sehr viel Platz in Anspruch genommen hat. Welche Gedanken und Fragen bei der Entscheidung über eine andere Schussanlage auftauchten und wie die Entscheidung nach Abwägung aller Vor- und Nachteile der

verschiedenen Möglichkeiten ausgegangen ist, wird in Abschnitt 2.2 erläutert.
Ähnlich wie im Vorgängermodell befinden sich die Akkupacks in einem dafür vorgesehenen Korb. Der Unterschied zum Vorgänger ist allerdings, dass die Akkus hingestellt werden, um Platz zu sparen. Unter dem Akkukorb befindet sich das Nachlaufrad in Form eines Tischtennisballs. Dies war beim Vorgängermodell auch so, jedoch wird durch eine Konstruktion erreicht, dass der Ball beim Anheben des Roboters nicht verloren geht.
Eine Regel besagt, dass der Roboter einen Griff für das Umsetzen des Roboters durch den Schiedsrichter haben muss. Der Griff des Vorgängers war sehr instabil und wurde deshalb durch einen stabileren ersetzt. Hierzu wird ein Kabel und ein längerer Legostein genutzt. Das Kabel wurde sinnvoll sowohl durch die Löcher im hinteren Teil, im vorderen Teil des Roboters als auch durch den Legostein gezogen. Der Legostein wird zum Anfassen genutzt.
Da das Torfinden, neben dem Ballfinden, die zweitwichtigste Sache beim Fußballroboter ist, werden bei dem Nachfolgermodell zusätzlich zum Kompass 3 IR-Fernsteuerungempfänger genutzt. Um diese anbauen zu können, wurde aus Sperrholz eine trichterförmige Sicht- und Halterungsanlage gebaut und oberhalb des AKSEN-Boards und unterhalb des Kompasses fixiert. Grund für die Nutzung der Empfänger ist, dass der Kompass allein nicht zuverlässig genug die Torrichtung bestimmen kann. Der Kompass verfügt nämlich nicht annähernd über eine Genauigkeit von 5°. Mit den IR-Fernsteuerungsempfängern ist der Roboter dann in der Lage, sich genau vor dem Tor zu positionieren. Empfängt der mittelste dieser Empfänger von dem Torbeacon ein Signal, dann und nur dann befindet sich das Tor in nördlicher Richtung und der Roboter steht mittig vor dem Tor. Grund dafür ist die Anordnung der 3 Empfänger und der Aufbau der trichterförmigen Sicht- und Halterungsanlage.
Weiterhin wurde die Umrandung des Roboters mit Legosteinen noch ausgebaut, so dass sichergestellt werden kann, dass bei Verkeilungen mit anderen Roboter die Kabel und Lötstellen geschützt werden.

2.2 Veränderung der Schussanlage

Die Notwendigkeit der Nutzung einer anderen Platz sparenden Schussanlage wurde bereits im vorhergehenden Abschnitt erwähnt. In den folgenden Abschnitten 2.2.1 bis 2.2.3 werden die verschiedenen Möglichkeiten analysiert.

2.2.1 Variante A - Schussanlage des Vorgängers

Die Schussvorrichtung des Vorgängers konnte aufgrund von Designänderungen nicht umgesetzt werden, da beim Yellow Panther2 Platz sparend gebaut werden sollte. Weitere Gründe, die gegen das Verwenden der alten Schussvorrichtung sprachen, sind:

- beim Verkeilen mit anderen Robotern, konnte es passieren, dass der Servomotor ausgehebelt wurde, weil die verwendeten Legoteile nicht die Stabilität liefern, die bei großer Kraft benötigt wird
- weiterhin wurde beim Verkeilen das Signal der Lichtschranke ausgelöst und die Schussanlage reagierte mit einem Schuss, weil nicht zwischen Ball, Wandecke und Roboter unterschieden werden konnte. Da beim Schuss gegen die Wand oder gegen den anderen Roboter der Widerstand höher war als beim Schuss gegen den Ball, war der Stromverbrauch auch höher.
- Kette zum Verbinden der Zahnräder nicht sehr stabil

2.2.2 Variante B - Ballhalte- und Schussanlage

Zum Aufbau dieser Art von Ballhalte- und Schussanlage werden zwei Räder und ein Motor (eventuell Getriebe) benötigt. Zum Fangen und Festhalten des Balles sollten die Räder vorwärts bewegt werden, während für den Schuss die Drehrichtung des Motors umgekehrt werden musste (Räder drehen rückwärts). Hierfür wurden gummierte, luftbefüllte Räder benutzt und es wurden Tests mit Legomotoren (LM), Servomotoren (SM) und Modellbaumotoren (MBM) durchgeführt. Bei den Tests ist aufgefallen, dass keiner der verwendeten Motoren annähernd zufrieden stellende Ergebnisse liefern konnte. Folgende Probleme sind mit den einzelnen Motoren aufgetreten:

Modellbaumotor

- es ist ein Getriebe notwendig, um die nötige Kraft für den Schuss zu erzeugen
- Platzmangel
- grobe Stückelung der Zahnräder

Legomotor

- nötige Kraft, um den Ball schießen zu können, nicht vorhanden

Servomotor

- Getriebe im Servo vorhanden
- zeichnet sich als kräftigster Motor unter den Getesteten aus
- vorhandene Kraft reicht nicht zum Bewegen des Balls aus (Beachte Masse (109g) und Trägheit des Balls)

Nach einigen Tests mit verschiedene Motoren ist man zu dem Schluss gekommen, dass es nicht möglich ist, in akzeptabler Zeit eine gute und effektive Lösung für diese Art von Schussanlage zu finden. Nach einigen Überlegungen über eine andere Art von „Schussanlage“ hat sich das Team entschieden, die im Folgenden Abschnitt (2.2.3) Ballfixierungs- und Schusseinheit für den Roboter zu verwenden.

2.2.3 Variante C - Ballfixierungs- und Schusseinheit

Es handelt sich hierbei um eine Art Greifarm, der den Ball fixiert und auch in Richtung Tor bewegen kann. Die Ausgangsstellung bei der Initialisierung sieht so aus, dass der Greifarm nach oben geklappt ist und somit der Käfig geöffnet ist. Das Öffnen und Schließen des Käfigs durch den Greifarm wird durch einen Servomotor realisiert. Der Schuss wird durch Heranfahren des Roboters an den Ball sowie einer gleichzeitig stattfindenden öffnenden Bewegung des Greifarms erzielt. Das bedeutet, dass bei einem Schuss der Ball aus dem Käfig freigegeben, der Käfig wieder geschlossen und dann bei der Annäherung an den Ball der Käfig wieder geöffnet wird, um den Ball in Richtung Tor zu bewegen.

Folgende Situationen können auftreten und sollten wie folgt behandelt werden:

Situation	Reaktion
Ball wird nicht erkannt	Roboter sucht Ball
Ball wird erkannt	Annähern an Ball und Fixierung mittels Greifarm
Ball im Käfig, Tor wird nicht erkannt	Mit Ball im Käfig Tor suchen – 3-Sekunden-Regel
Ball im Käfig, Tor wird erkannt	Vom Ball entfernen, warten und Annähern mit Schuss

Tabelle 1 vorstellbare Situationen und ihre Reaktionen

Während der Analyse der einzelnen Schussanlagen ist die Frage aufgetreten, ob die Lichtschranke auch bei dieser Variante benötigt wird. Nach einigen Überlegungen ist man zu dem Schluss gekommen, dass die

Lichtschranke benötigt wird, weil nur mit ihr beurteilt werden kann, ob der Ball in der Nähe des Roboters ist und ob der Käfig geschlossen werden kann. Sie ist Auslöser zum Schließen des Käfigs. Natürlich können hier ähnliche Probleme wie bei der Variante A auftreten. Dazu zählt beispielsweise, dass der gegnerische Roboter oder eine Wandecke das Schließen des Käfigs bewirken kann. Das Eckenproblem wird mit Hilfe weiterer Sensoren für die Wanderkennung gelöst (vgl. 2.3.2).

Nach Abwägung des Nutzens gegenüber dem eben erwähnten Nachteil hat sich das Team für das Verwenden dieser Variante entschieden.

Wichtig für die Programmierung sind im Zusammenhang mit der Schussanlage zwei Regeln, die unbedingt berücksichtig werden müssen.

- Ball maximal 3 Sekunden festhalten (TIMER benutzen – aktTime())
- nach 3 Sekunden 10cm (handbreit) vom Roboter entfernen, 1 Sekunde warten und erst mit dem Schießvorgang beginnen

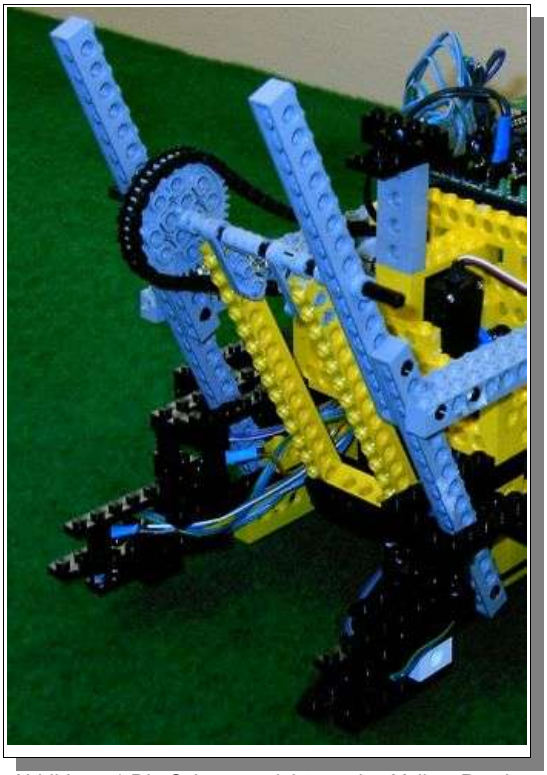

Abbildung 1 Die Schussvorrichtung des Yellow Panther

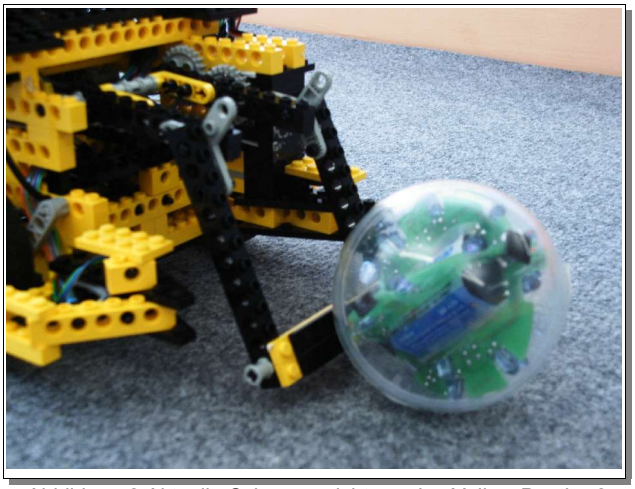

Abbildung 2 Aktuelle Schussvorrichtung des Yellow Panther2

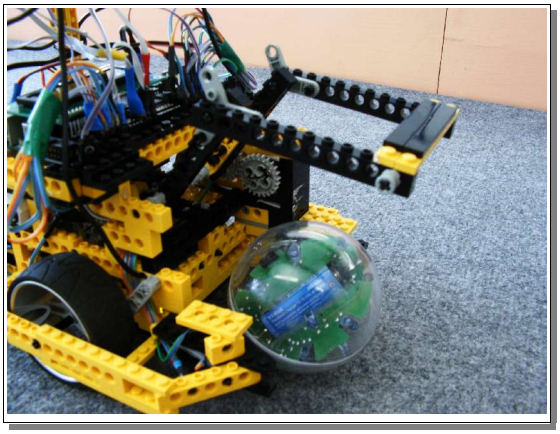

Abbildung 3 Schussvorrichtung dient auch als Käfig

2.3 Verwendete Sensorik und Aktorik

Der Abbildung kann entnommen werden, welche Sensoren und Aktuatoren verwendet werden. Außerdem gibt die Abbildung Aufschluss darüber, wo die einzelnen Sensoren und Aktuatoren angebracht wurden, welche Anschlüsse genutzt werden und wofür speziell die Sensoren (IR-Empfänger und IR-Fernsteuerungs-empfänger) eingesetzt werden.

Für die Ballerkennung werden Infrarot-Empfänger genutzt. Wie die Ballerkennung funktioniert, wird im Abschnitt 2.3.1 beschrieben. 3 IR-Empfänger und 3 LEDs kommen für die Wanderkennung zum Einsatz. Mehr Informationen zu der Wanderkennung sind im Abschnitt 2.3.2 zu finden. Für das Erkennen und Finden des eigenen Tores wird zum einen ein Kompass und zum anderen 3 IR-Fernsteuerungsempfänger benutzt. Weitere Details zum Thema Torerkennung werden in Abschnitt 2.3.3 erläutert. Wie bereits in Abschnitt 2.2.3 beschrieben, wird für die Schussanlage ein Servomotor verwendet. Für das Fortbewegen des Roboters werden zwei Modellbaumotoren und ein Getriebe aus 2 verschiedenen Zahnradgrößen verwendet.

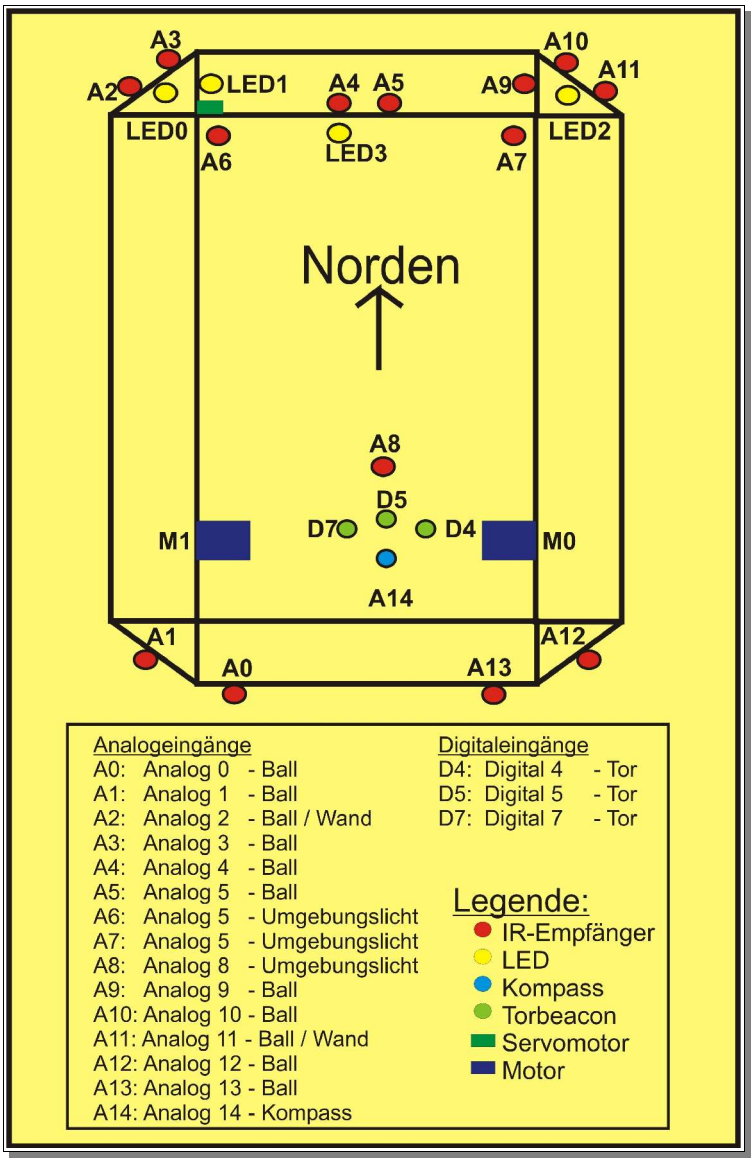

Abbildung 4 Anordnung der Sensoren und Aktutoren

2.3.1 Ballerkennung

Die 13 mit roten Kreisen dargestellten Infrarotempfänger werden für die Ballerkennung genutzt (Ausnahme A9). Am Roboter wurden an der Rückseite 4 zusätzliche IR-Empfänger angebracht. Grund dafür ist eine Regel die sagt, dass bei Verkeilungen und/oder akuter Langeweile die Roboter Rücken an Rücken positioniert werden und der Ball zwischen die Roboter gelegt wird.

Die Empfänger mit den Namen A6, A7 und A8 sind für den Empfang des Umgebungslichts verantwortlich, d.h. diese sollten möglichst von allen anderen IR-Sendern wie Ball oder LEDs abgeschirmt werden. Beim Vorgängermodell wurde für den Empfang des Umgebungslichtes nur ein Empfänger genutzt. Der aktuelle Roboter benutzt 3 Empfänger, weil bei diversen Tests aufgefallen ist, dass der Roboter in Richtung Fenster blickend Bälle erkennt, wo keine sind. Nach Analyse des Problems stand fest, dass das Ermitteln des Umgebungslicht mit einem Empfänger nicht ausreicht und somit wurden 2 zusätzliche Empfänger angebracht. Normalerweise bräuchte der Roboter auch hinten Empfänger für das Umgebungslicht, aber die konnten aus Mangel an analogen Eingängen nicht mehr angebracht werden. Die drei Empfänger sind so ausgerichtet, dass A8 (mitte) das Umgebungslicht für die Sensoren in der Mitte und hinten nutzt, während die beiden anderen etwas tiefer angebracht sind und einmal links vorn und rechts vorn das Umgebungslicht ermitteln.

Um dieses Problem zu beheben, gab es mindestens noch eine Alternative, die jedoch aus Zeitmangel nicht umgesetzt werden konnte. Denn unter Umständen besteht die Möglichkeit, die Werte von sogenannten Photosensoren auf das Umgebungslicht abzubilden. Wie in kurzen Tests herausgefunden wurde, reagieren diese Sensoren zwar auf Helligkeitsänderungen, aber sie werden nicht von Infrarotlicht abgelenkt.

Vorn links, vorn rechts, vorn mittig und hinten mittig sind jeweils zwei Sensoren angebracht worden. Das ist durchaus sinnvoll, da der Ball nicht komplett mit IR-Sendern ausgestattet ist, sondern nur an bestimmten Stellen die Sender befestigt sind. Deshalb kann es vorkommen, dass nur einer der beiden Sensoren den Ball erkennt, während der andere kein Signal vom Ball empfängt. Aus diesem Grund wird der Maximalwert von beiden genommen und somit kann sichergestellt werden, dass der Ball auch erkannt wird, wenn nur ein Sensor auf ihn reagiert. Bevor jedoch der Maximalwert bestimmt wird, wird der ermittelte Wert der Sensoren mit dem Umgebungslicht verglichen. Vergleichen heißt in diesem Fall, das der Wert des Sensorsvon dem Wert des Umgebungslichtsensors (z.B. A8) subtrahiert wird. Ist der Ball nicht zu sehen, dann sind beide Werte identisch und das Ergebnis der Subtraktion ist null. Wird der Ball detektiert, ist der Wert des Sensors, der ihn erkennt, kleiner als der Wert des Umgebungslichts. Die Subtraktion liefert somit ein Ergebnis größer als Null. Je größer der Wert ist, desto näher ist der Ball am Roboter. Nach dieser Subtraktion wird dann der Maximalwert der beiden nebeneinander liegenden Sensoren ermittelt. Übersteigt dieser Wert einen Schwellwert von 8, kann man davon ausgehen, dass der Ball ganz in der Nähe ist. Je nachdem welche Sensoren behaupten den Ball zu erkennen, muss sich der Roboter bewegen, d.h. vor fahren oder sich drehen.

Natürlich muss man davon ausgehen, dass es sich nur um eine Behauptung der Sensoren handelt, weil die Empfänger von anderen Sendern in der Umgebung abgelenkt werden können und nicht zwischen Ball und anderen Störquellen unterschieden werden kann. Dies stellt auch die größte Fehlerquelle dar. Um diese Fehler auf ein Minimum zu reduzieren, könnte man beispielsweise Fenster abdunkeln.

2.3.2 Wanderkennung

3 von 12 mit roten Kreisen dargestellten Infrarotempfänger (A3, A4 und A10) sowie 3 LEDs (LED0, LED3 und LED2) werden für die Wanderkennung verwendet. Im Gegensatz zum Vorgängermodell gibt es keine separaten Empfänger für die Wanderkennung, weil wie bereits erwähnt die analogen Eingänge am AKSEN-Board begrenzt sind. Es werden dieselben Sensoren wie für die Ballerkennung genutzt. Um der Wand ausweichen zu können, muss der Roboter in der Lage sein, die Entfernung zur Wand einigermaßen abschätzen zu können. Mit Hilfe der IR-Empfänger kann keine richtige Entfernungsschätzung in Zentimeter abgegeben werden, vielmehr handelt es sich hierbei um den Vergleich mit einem in Tests ermittelten Schwellwert.

Die Schätzung der Entfernung läuft wie folgt ab: von den IR-Empfängern A2, A4 und A11 werden die Werte eingelesen, während die LEDs ausgeschalten sind. Bei den Werten handelt es sich ähnlich wie bei der Ballerkennung um das Umgebungslicht. Im nächsten Schritt werden die LEDs angeschaltet und die Werte IR-Empfänger (Wert2) werden erneut abgefragt. Ist das Umgebungslicht größer als der Wert bei eingeschalteten LEDs wird auch hier eine Subtraktion durchgeführt (Umgebungslicht – Wert2), andernfalls ergibt sich eine 0.

Bei der Wanderkennung wird das Prinzip der Reflexion ausgenutzt. Wenn die LEDs angeschaltet sind und sich in der Nähe des Roboters eine Wand befindet, wird das von den LEDs ausgestrahlte Licht an der Wand reflektiert und kann vom IR-Empfänger aufgefangen werden. Ist der Roboter weit von der Wand entfernt, kommt von dem reflektierten Licht nur sehr wenig beim Empfänger an und der Wert der Subtraktion ist nur geringfügig größer als 0. Je näher der Roboter an der Wand steht, desto mehr reflektiertes Licht wird empfangen und desto kleiner ist der Wert2 im Gegensatz zum Umgebungslicht. D.h. je höher der Wert der Subtraktion ist, desto näher befindet sich der Roboter an der Wand.

Beim Testen der Sensoren ist aufgefallen, dass der Wert, welcher bei der Subtraktion entsteht, zwischen 20 und 25 den optimalen Abstand zwischen Wand und Roboter liefert. Es muss natürlich beachtet werden, dass man den Abstand nicht zu klein wählt, da sonst ein Ausweichen doch zum Crash führen kann. Der Roboter darf mit seiner Größe beim Wenden / Ausweichen nicht an den Ecken hängen bleiben. Wird der Abstand jedoch zu groß gewählt, kann es auch vorkommen, dass der Roboter den Ball in der Ecke oder nah an der Wand nie erreichen kann. Diesem Problem kann man mit Hilfe der Programmierung aus dem Weg gehen, wenn dem „Ball holen“ eine höhere Priorität als dem „Wand ausweichen“ zugeordnet wird.

Im Gegensatz zum Vorgängermodell werden zusätzlich noch eine LED und ein IR-Empfänger eingesetzt, die bewirken sollen, dass der Roboter nicht frontal gegen die Wand fährt bzw. sich an den Torecken die Schussvorrichtung kaputt fährt (siehe Abbildung). Es wird nicht mehr zugelassen, dass der Roboter zu dicht an die Ecken fährt und die Lichtschranke ausgelöst wird. Das Auslösen der Lichtschranke in dieser Situation führte dazu, dass sich der Servomotor aushebelte und die Funktionstüchtigkeit eingeschränkt wurde.

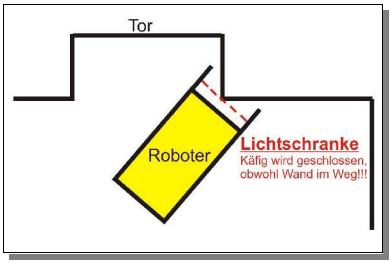

Abbildung 5 Grafische Darstellung des Eckenproblems

Für die Wanderkennung wurden zusätzlich Sharp-Sensoren getestet. Leider reichte die Zeit für umfangreiche Tests nicht aus und somit konnten diese Sensoren nicht eingesetzt werden. Ein weiterer Grund, warum diese Sensoren nicht verwendet wurden, ist ihre Größe und die damit verbundene Schwierigkeit sie im vorderen Bereich des Roboters unterzubringen.

2.3.3 Torerkennung

Wie bereits erwähnt wird für die Torerkennung zum einen ein Kompass benutzt. Die Genauigkeit des Kompasses für die Positionierung des Roboters ist nicht ausreichend, wie die beiden folgenden Abbildungen und die zugehörigen Erläuterungen zeigen werden.

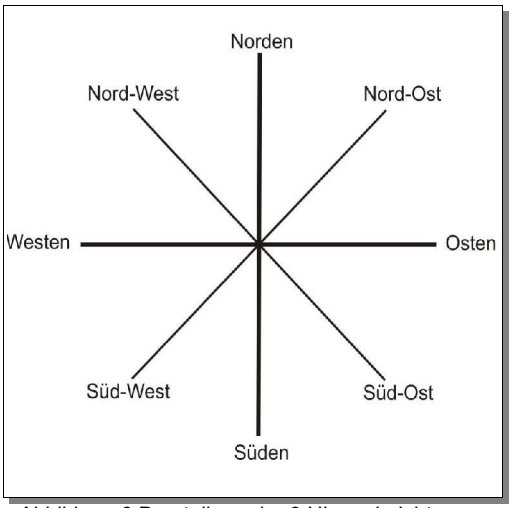

Abbildung 6 Darstellung der 8 Himmelsrichtungen

Himmelsrichtung	Wert
Norden	65
Nord-Ost	86
Osten	126
Süd-Ost	126
Süden	151
Süd-West	176
Westen	190
Nord-West	47

Abbildung 7 ermittelte Kompasswerte

In der linken Abbildung werden die Himmelsrichtungen gezeigt, die für die Positionierung genutzt werden können. Weiterhin werden in der rechten Abbildung die Werte aufgezeigt, die der Kompass für die jeweilige Himmelsrichtung angibt. Alle möglichen Ausrichtungen, die der Roboter vor dem Ausführen des Torschusses annehmen könnte, sind nicht genau genug, wie an dem folgenden Beispiel gezeigt wird: Ist die Ausrichtung des Roboters nicht direkt nach Süden, aber auch nicht nach Osten, sondern blickt der Roboter irgendwo in süd-östlicher Richtung, liefert der Kompass immer den Wert 126 zurück. Das wiederum bedeutet, dass es nicht möglich ist, den Roboter präzise genug auf das Tor auszurichten, zumal der Wert für Osten und Süd-Ost identisch ist.

Deshalb bestand zum anderen die Notwendigkeit IR-Fernsteuerungsempfänger zu verwenden. Diese empfangen von den über dem Tor angebrachten Beacons ein moduliertes Signal von 100 oder 125 Hz. Wichtig bei der Nutzung dieser Empfänger ist die sinnvolle Anordnung. Wie man der folgenden Abbildung entnehmen kann, sieht die Konstruktion wie eine Art Trichter aus.

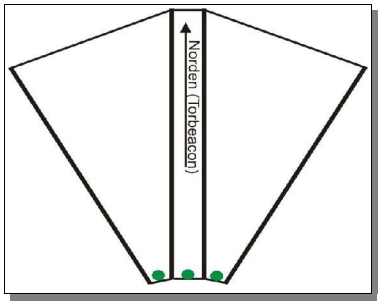

Abbildung 8 Konstruktion aus Sperrholz für die IR-Fernsteuerungsempfänger

Die Konstruktion wurde aus Sperrholz gefertigt und mit schwarzer Farbe lackiert. An der kleinen Öffnung hinten sind die 3 Empfänger fixiert. Jeder Empfänger ist mittels Trennwand von den anderen abgeschottet. Jeder der Empfänger kann nur bei einer bestimmten Ausrichtung des Roboters das Signal des Torbeacons empfangen. Sieht der linke Empfänger (D7) das Tor, muss der Roboter solange nach links drehen bis der mittlere Empfänger (D5) das Signal empfängt. Sieht der rechte Empfänger (D4) das Tor, muss der Roboter solange nach rechts drehen bis der D5 das Signal empfängt. D.h. das der Roboter nur dann korrekt ausgerichtet ist und es sich lohnt einen Schuss abzugeben, wenn der mittlere Empfänger das modulierte Signal empfängt.

2.3.4 Motor und Getriebe

Für den Antrieb des Roboters werden 2 handelsübliche Modellbaumotoren genutzt. Das Getriebe ist aus 3 großen und 3 kleinen Zahnrädern aufgebaut. Um Platz zu sparen, wurden die Zahnräder anstatt auf zwei Ebene auf drei Ebenen angebracht. Dies ist in der nachfolgenden Abbildung zu erkennen. Auf die Welle des Motors ist ebenfalls ein Zahnrad der kleinsten Größe fixiert.

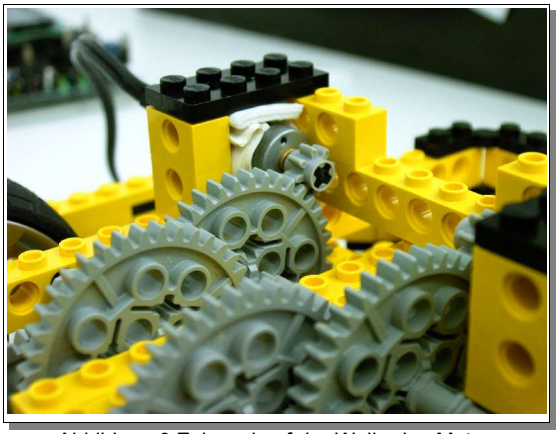

Abbildung 9 Zahnrad auf der Welle des Motors

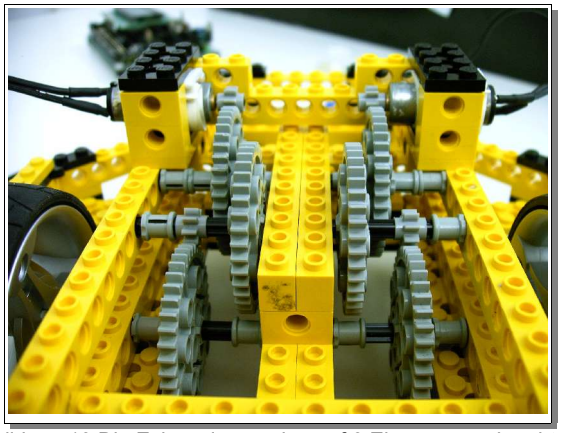

Abbildung 10 Die Zahnräder wurden auf 3 Ebenen angebracht

Das Anbringen der Zahnräder auf mehrere Ebenen hat zu großen Problemen und zu häufigen Umbauten geführt. Der erste Entwurf des Getriebes, der in der rechten oberen Abbildung gezeigt wird, hatte eine Übersetzung von 125 : 1. Leider war hier jedoch das Problem, dass das letzte kleine Zahnrad nicht stark genug in das große Zahnrad gefasst hat und somit das ein oder andere Mal sich die Räder nicht gedreht haben. Bei den folgenden Tests ist man nicht zu einer Anordnung gelangt, die die gleiche Übersetzung hatte. Deshalb wurde versucht, das letzte kleine Zahnrad durch ein etwas größeres Zahnrad zu ersetzen. Dadurch hat sich aber die Übersetzung auf 125 : 2 verringert. Weiterhin ist aufgefallen, dass der Roboter zu schnell ist und nicht angemessen schnell auf seine Umwelt reagieren kann. Aus diesem Grund sollte die Übersetzung erhöht werden, so dass der Roboter an Geschwindigkeit verliert und an Kraft gewinnt. Hierzu wurden zwei verschiedene Getriebe gebaut. Das eine Getriebe hatte eine Übersetzung von 625 : 1 und das zweite eine Übersetzung von 375 : 1. Leider waren diese Übersetzungen nicht zufrieden stellend und die verschiedenen Anordnungsmöglichkeiten wurden weiter getestet, weil die Geschwindigkeit zum Ausführen des Schusses nicht mehr ausreichte.

Letztendlich wurde doch noch eine Variante gefunden, die wie oben beschrieben 4 kleine und 3 große Zahnräder besitzt und eine Übersetzung von 125 : 1 hat. Die nachfolgende Abbildung zeigt, wie die Übersetzung berechnet wird.

Das anfängliche Problem, dass der Roboter seine Umwelt nicht wie gewünscht wahrnimmt, lag auch zum Teil an der Programmierung, die durch Fehleranalyse geändert wurde.

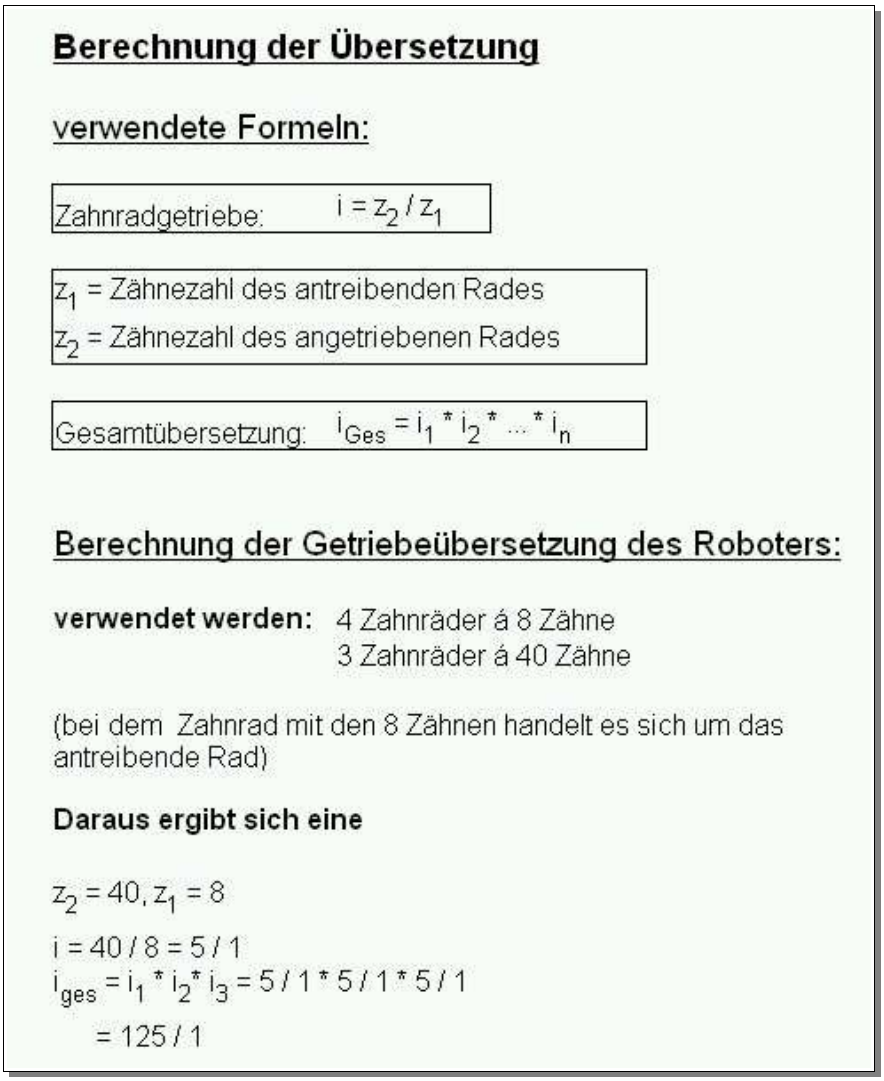

Abbildung 11 Berechnung der Übersetzung

3 Programmierung

In diesem Kapitel wird speziell auf die Programmierung eingegangen. Die Programme werden in C geschrieben. Für die Programmierung eines fußballspielenden Roboters hat sich die Verwendung einer Zustandsmaschine angeboten. Im nachfolgenden Abschnitt werden ein paar allgemeine Fakten genannt und im Abschnitt 3.2 wird dann die Zustandsmaschine und ihre Umsetzung in C beschrieben. In einem weiteren Abschnitt wird die Verwendung von KI-Methoden diskutiert.

3.1 Allgemeine Fakten

Im Gegensatz zu der Programmierung des Vorgängermodells wird bei dieser Entwicklung der Programme auf Übersichtlichkeit großen Wert gelegt. Deshalb wurden die verschiedene Funktionen aus der Main.c ausgelagert und auf die entsprechenden C-Dateien verteilt. Folgende C-Dateien sind neben der Main.c entstanden: BallDetection.c, Common.c, GoalDetection.c, Global.c, Motor.c, Shooter.c und WallDetection.c. Um die C-Dateien benutzen zu können, wurden folgende Header-Dateien angelegt, die neben den Funktionsdeklarationen, Konstanten oder globale Variablen enthalten. Dazu zählen: BallDetection.h, Common.h, Constants.h, Global.h, GoalDetection.h, Motor.h, Shooter.h und WallDetection.h. In der Main.c ist die Zustandsmaschine enthalten, welche die Zustände SEARCH_BALL, SEARCH_GOAL, TROUBLE und SHOT beinhaltet.

3.2 Verwendung einer Zustandsmaschine

Für die Verwendung einer Zustandsmaschine wurden 4 Zustände definiert. Folgende Zustände werden verwendet: SEARCH_BALL, SEARCH_GOAL, TROUBLE und SHOT. Nur in der Main.c werden die Zustandsänderungen vorgenommen. Es gibt jedoch eine Ausnahme. Das ist der Wechsel von SEARCH_GOAL zu SHOT, wenn das Tor detektiert wurde. Dieser Zustandswechsel findet direkt in der GoalDetection.c statt.

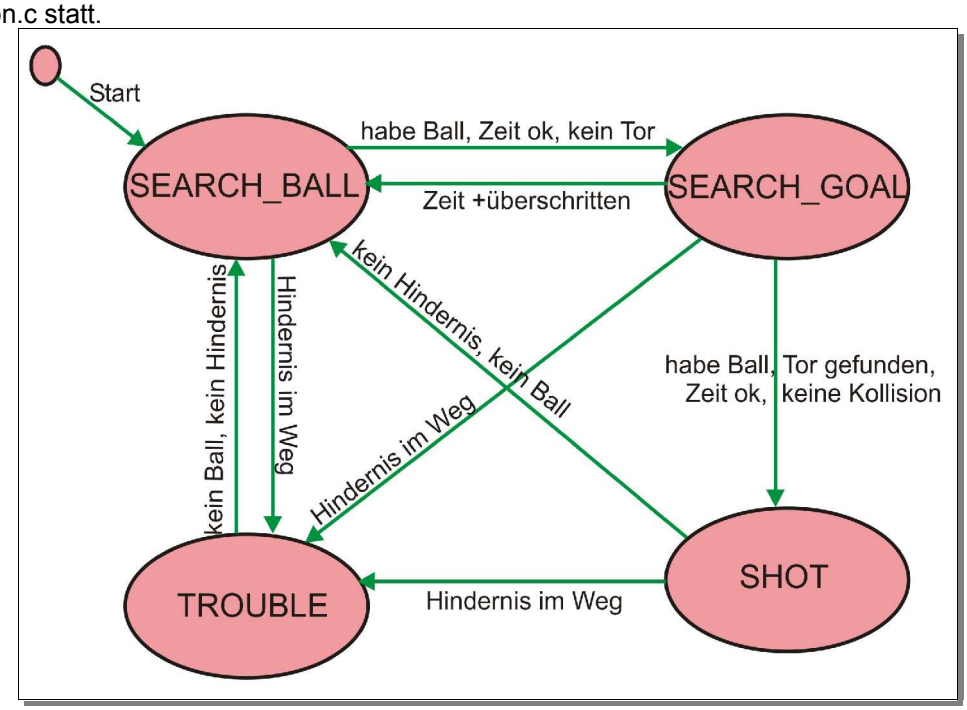

Abbildung 12 Zustandsmaschine mit 4 Zuständen

Die Namen der Zustände sind selbst erklärend. Bei dem Zustand SEARCH_BALL geht es darum, dass der Roboter sich auf dem Spielfeld bewegt und dabei mit Hilfe der IR-Sensoren Ausschau nach dem Ball hält. Bei Beginn des Programms befindet sich die Zustandsmaschine in diesem Zustand. Wenn der Roboter den Ball im Käfig hat, dann sucht er unter Verwendung des Kompasses und der 3 IR-Fernsteuerungsempfänger das Tor. Hat er das Tor gefunden, findet ein Zustandswechsel in den Zustand SHOT statt. In diesem Zustand ist es wichtig, sich an die Regeln zu halten, bevor der Schuss abgegeben werden kann. Dazu zählt vor allem, dass sich der Roboter vor dem Schuss eine Hand breit von dem Ball entfernt und dann erst schießt (vgl. Regel 15 im Anhang). Taucht irgendwo ein Hindernis in Form einer Wand auf, dann erfolgt ein Wechsel in den Zustand TROUBLE und eine passende Maßnahme wird eingeleitet. Mehr Einzelheiten zu der Programmierung sind in den nachfolgenden Abschnitten 3.2.1 und 3.2.2 zu finden. In diesen Abschnitten werden die Funktionen beschrieben. Weiterhin können zur unterstützenden Erklärung der einzelnen Funktionen die Programmkommentare im Anhang B hinzu gezogen werden. Mit Hilfe der nachfolgenden Abbildung soll der Programmablauf schematisch dargestellt werden.

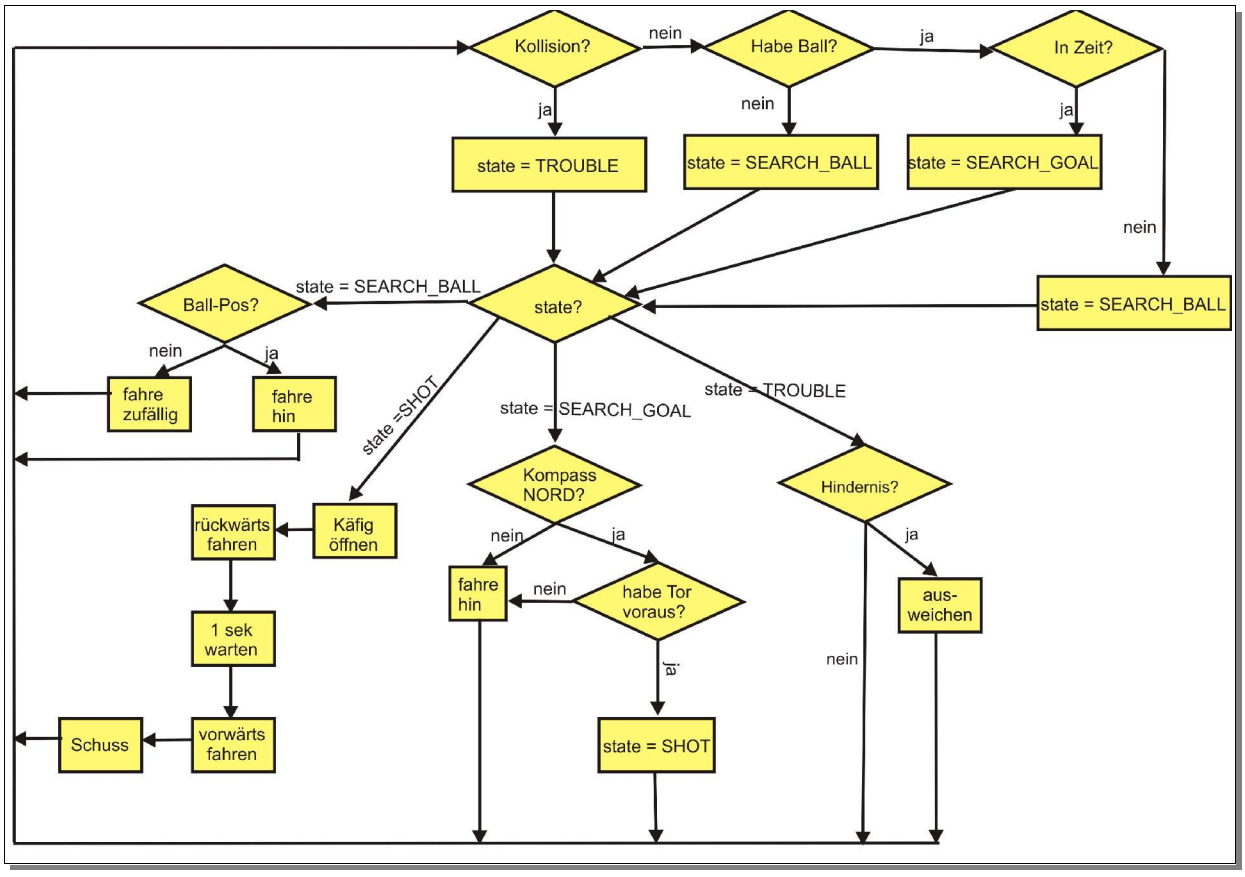

Abbildung 13 Programmablaufplan (PAP)

3.2.1 Programmdateien

In diesem Abschnitt wird auf die einzelnen Header- und C-Dateien eingegangen. Dabei werden die verwendeten Datentypen und die generelle Arbeitsweise der Funktionen beschrieben. Der Quellcode mit Kommentaren ist im Anhang B zu finden.

3.2.1.1 Constants.h

In dieser Header-Datei sind alle Konstanten definiert, welche in den verschiedenen C-Dateien zur

Verwendung kommen. Die Konstanten sind in fünf Gruppen unterteilt.

- Parameter (Schwellwerte)
- LED Parameter (LED Ports)
- Motor Parameter (Motor Ports, Bewegungsrichtung, max. und min. Geschw.)
- Servo Motor Parameter (Servo Port, Winkel)
- Analoge Sensor Parameter (Analoge Ports)
- Digitale Sensor Parameter (Digitale Ports, Torfrequenzen, Fehler- und Schwellwert)

3.2.1.2 Global.h und Global.c

Die Global-Dateien definieren globale Variablen sowie bestimmte Datentypen (Enums).

enum boolean {TRUE, FALSE}

In C existiert kein Datentyp boolean wie in C++, um boolsche Rückgabewerte von Funktionen zu erhalten, und deshalb wurden die beiden Werte TRUE und FALSE definiert.

enum states {SEARCH_BALL, SEARCH_GOAL, TROUBLE, SHOT}

Dieser Datentyp definiert die verschiedenen Zustände, welche die Zustandsmaschine einnehmen kann.

- SEARCH_BALL→ Suche Ball
- SEARCH_GOAL→ Suche Tor
- TROUBLE→ Hindernis
- SHOT→ Ausführen des Schusses

enum compassDirection {NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST}

In diesen Enum sind die acht Kompassrichtungen definiert. Dieser Typ wird bei der Torsuche mit dem Kompass benutzt.

enum objectPosition {NONE, LEFT, RIGHT, CENTER}

Dieser Datentyp wird bei der Wanderkennung verwendet und enthält Werte, die angeben in welchem Bereich sich das Hindernis befindet.

- NONE→ kein Hindernis
- LEFT→ Hindernis ist auf der linken Seite
- CENTER→ Hindernis ist vor einem
- RIGHT→ Hindernis ist auf der rechten Seite

enum ballPosition {NONE, FRONTLEFT, FRONTCENTER, FRONTRIGHT, BACKCENTER, BACKLEFT,BACKRIGHT}

Dieser Datentyp kommt bei der Ballsuche zum Einsatz, die Werte beschreiben die Lage des Balles zum Roboter. Jedem Wert aus diesem Typ wurde über die Zahl ein Bitmuster zugeordnet, um die Beschreibungsmöglichkeit der Balllage breit zu Fächern.

- NONE = 0 → kein Ball in Sichtweite

- FRONTLEFT = 1 → Ball liegt vorne links
- FRONTCENTER = 2 → Ball liegt vorne mittig
- FRONTRIGHT = 4 → Ball liegt vorne rechts
- BACKCENTER = 8 → Ball liegt hinten mittig
- BACKLEFT = 16 → Ball liegt hinten links
- BACKRIGHT = 32 → Ball liegt hinten rechts

enum goalPosition {NONE, LEFT, CENTER, RIGHT}

Bei der Torsuche werden drei IR-Fernsteuerungsempfänger benutzt. Für diese Empfänger wurden vier Zustände definiert.

- NONE → keiner sieht das Tor
- LEFT → der linke Empfänger sieht das Tor (D7)
- CENTER → der mittlere Empfänger sieht das Tor (D5)
- RIGHT → der rechte Empfänger sieht das Tor (D4)

3.2.1.3 Common.h und Common.c

Die Common-Dateien definieren allgemeine Funktionen, Hilfsprogramme und die Initialisierungsfunktion.
In der Initialisierungsfunktion (init()) werden die DIP-Schalter des AKSEN Boards abgefragt und entschieden, ob das Hauptprogramm laufen soll oder ein Hilfsprogramm gestartet wird. Weiterhin werden die IR-Fernsteuerungsempfänger auf die entsprechende Torfrequenz eingestellt.
Zu den allgemeinen Funktionen zählen Funktionen, die den Minimum-, Maximum oder Absolutwert zurückgeben. Außerdem sind Funktionen enthalten, die die Differenzen bilden zwischen dem Umgebungslicht und dem empfangenen Licht (compareToSurround(), compareToSurroundPort() - Ballerkennung) bzw. zwischen dem reflektierten und dem empfangenen Licht (getIR() - Wanderkennung).
Hilfsprogramme sind Ausgaben von verschiedenen Sensorwerten oder Testprogrammen wie das Testen der Schussanlage.

3.2.1.4 BallDetection.h und BallDetection.c

Die BallDetection-Dateien bestehen aus drei Funktionen findBall(), getBallPosition und hasBall().
Die Funktion findBall() bekommt über den Aufruf der Funktion getBallPosition() die Lage des Balles zum Roboter zurück und reagiert darauf, in dem der Roboter sich dreht oder vorwärts fährt.
Die Funktion getBallPosition() gibt Bitmuster aus einen oder verschiedenen Werten aus dem **enum ballPosition** zurück. Dieses Bitmuster beschreibt die Ballposition zum Roboter.
Die Funktion hasBall() ermittelt über eine Lichtschranke (LED1, A9), ob der Ball im Ballkäfig ist und liefert einen Wert aus dem Datentyp **enum boolean** zurück. Des Weiteren wird der Käfig geschlossen und der Timer zurückgesetzt, sobald der Ball in den Käfig kommt.

3.2.1.5 GoalDetection.h und GoalDetection.c

In den beiden GoalDetection-Dateien sind drei Funktionen definiert findGoal(), getCompassDirection() und getGoalSensor().
Die Funktion findGoal() ermittelt über den Aufruf der beiden anderen Funktionen die Position des Tores und fährt bzw. dreht den Roboter dorthin.

Die Funktion getCompassDirection() bekommt von dem Kompass (A14) eine ungefähre Richtung des Tores (Kapitel 2.3.3). Der Rückgabewert ist ein Wert aus dem Datentypen **enum compassDirection**.
Die Funktion getGoalSensor() dient der Feinausrichtung des Roboters zum Tor. Es wird ermittelt, ob das Tor von einem der drei IR-Fernsteuerungssensoren gesehen wird und liefert einen Wert aus **enum goalPosition** zurück.

3.2.1.6 WallDetection.h und WallDetection.c

In diesen beiden Dateien sind nur zwei Funktionen definiert findWall() und hasCollision().
Die Funktion findWall() versucht, mit dem gesetzten Wert der globalen Variable ***object*** (**enum objectPosition**) dem Hindernis durch verschiedenes Steuern der Motoren auszuweichen. Das Auswerten dieser globalen Variablen erfolgt auch über die Auswertung von Bitmustern (vgl. 3.2.1.4).
Die Funktion hasCollision() ermittelt, ob sich ein Hindernis vor den Wanderkennungssensoren (A3, A4, A10) befindet und liefert einen Wert aus dem Datentyp **enum boolean** zurück. Des weiteren wird beim Finden eines Hindernisses die globale Variable ***object*** (**enum objectPosition**) auf den Wert LEFT, RIGHT oder CENTER gesetzt.

3.2.1.7 Motor.h und Motor.c

Diese beiden Dateien definieren Funktionen zum gerichteten Steuern beider Motoren, wobei nur die Geschwindigkeit variabel ist.

- moveForward() → vorwärts fahren, Driften durch verschiedene Geschwindigkeiten möglich
- moveBackward() → rückwärts fahren, Driften durch verschiedene Geschwindigkeiten möglich
- turnLeft() → Linksdrehung auf der Stelle
- turnRight() → Rechtsdrehung auf der Stelle
- fullStop() → beide Motoren reduzieren ihre Geschwindigkeit auf Null

3.2.1.8 Shooter.h und Shooter.c

Die Shooter-Dateien beinhalten Funktionen für die Schussanlage und die Schussabfolge. Die zwei Funktionen shooterDown() und shooterUp() veranlassen den Arm der Schussanlage sich zu senken bzw. zu heben. Währenddessen ist in der Funktion shot() der Ablauf des Schusses beschrieben, wie er in Regel 15 im Anhang festgelegt ist. Außerdem existiert eine Funktion namens isInTime(), welche einen Wert des Typs **enum boolean** zurückliefert. Diese Funktion bestimmt, ob das Zeitlimit für das Festhalten des Balles überschritten ist. Ist das Zeitlimit überschritten, wird der Ball freigeben.

3.2.1.9 Main.c

Die C-Datei Main enthält die Hauptfunktion des Programms, in der die Zustandsmaschine beschrieben ist. Der erste Schritt in der Hauptfunktion ist das Initialisieren bestimmter Komponenten (siehe Kapitel Common). Danach kommt das Programm in eine Endlosschleife, in der ständig die verschiedenen Bedingungen abgefragt, die Zustände gesetzt und aufgerufen werden.
Dabei wird als erstes geprüft, ob irgendwo vor dem Roboter ein Hindernis ist, wenn ja, wechselt der Roboter in den Zustand TROUBLE und versucht dem Hindernis auszuweichen. Wenn kein Hindernis vorhanden ist, wird überprüft, ob der Ball im Ballkäfig ist. Ist dies der Fall, wird in den Zustand SEARCH_GOAL übergegangen und der Roboter sucht das Tor.

Wenn kein Ball im Käfig ist wechselt der Zustand in SEARCH_BALL über und der Roboter versucht, den Ball auf dem Spielfeld zu lokalisieren.

3.2.2 Verwendung der Dip-Schalter

Während des Projektes kamen immer wieder Programme zum Einsatz, welche z.B. die Ausgabe von Sensorwerten oder das Testen von Sequenzen ermöglichen. Durch die verschiedenen Einstellungsmöglichkeiten der DIP-Schalter des AKSEN Boards, ist es möglich die verschiedenen Programme in der Initialisierungsphase zu starten.

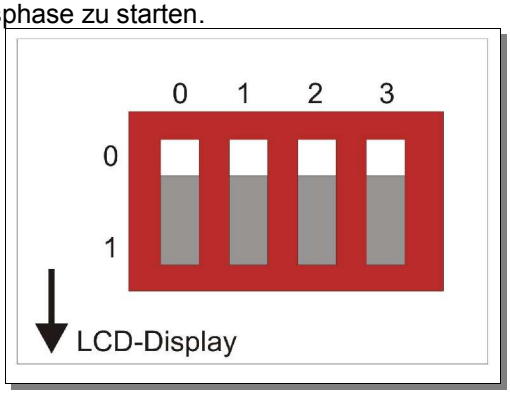

Abbildung 14 Dip-Schalter

Der DIP-Schalter 0 ist für die Torfrequenz reserviert. Die IR-Fernsteuerungsempfänger werden je nach der Position des DIPs eingestellt. Befindet er sich auf der Position 0, so empfangen die IR-Fernsteuerungsempfänger 100 Hz, ansonsten 125 Hz.

DIP 0 - 0 → 100 Hz
- 1 → 125 Hz

Die möglichen Programme, welche über die DIP-Schalter benutzt werden können, sind in der nachfolgenden Tabelle aufgelistet.

Dip 1	***Dip 2***	***Dip 3***	
0	0	0	→ starten des Hauptprogrammes
0	0	1	→ Ausgabe der Ballsensoren
0	1	0	→ Ausgabe der Umweltsensoren
0	1	1	→ Ausgabe der Wandsensoren
1	0	0	→ Ausgabe der Torsensoren und des Kompass
1	0	1	→ Schusstest
1	1	0	
1	1	1	

Tabelle 2 Dipschalterbelegung

3.3 Umsetzung von KI-Methoden

In der Diskussion für eine eventuelle Umsetzung von KI-Methoden wurden folgende Methoden in Betracht gezogen: Neuronale Netze, Fuzzy-Logik und Künstliche Evolution. Bei Neuronalen Netzen benutzt man einen Satz von Trainingsdaten, um das Netz zu trainieren. Da diese Trainingsdaten nicht zur Verfügung stehen und vorerst definiert werden müssten, ist diese KI-Methode nicht geeignet. Ein weiterer Grund ist, dass nicht bekannt ist, wie der Aufbau der Trainingsdaten für den speziellen Fall aussehen sollte.

Bei der Fuzzy Logik sieht es schon etwas anders aus. Hier benötigt man Fuzzy-Mengen um einen Fuzzy-Regler aufbauen zu können. Für die Ball- und Wanderkennung könnte man Fuzzy-Mengen definieren wie in den beiden nachfolgenden Abbildung gezeigt.

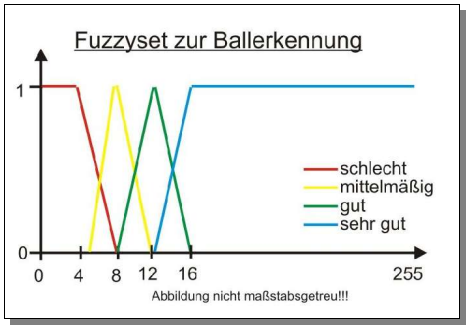

Abbildung 15 Mögliche Fuzzy-Menge für die Ballerkennung

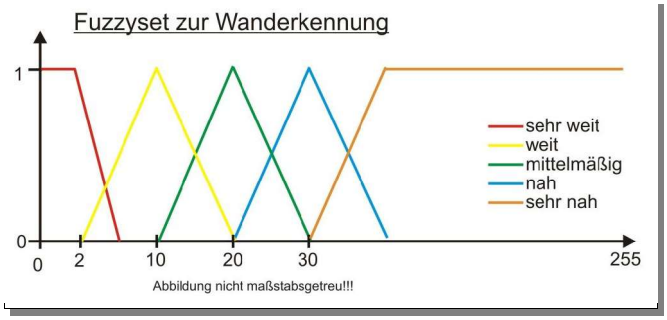

Abbildung 16 Mögliche Fuzzy-Menge für die Wanderkennung

Jedoch ist die Anwendung von Fuzzy-Logik nicht angemessen, da man sich auch mit einem einfachen If-Then-Else-Konstrukt behelfen kann und man mit Fuzzy Regeln kein anderes Ergebnis erzielen würde. Außerdem ist die Zeit für die Umsetzung zu knapp.

Die Methode Künstliche Evolution ist diesbezüglich vergleichbar mit den Neuronalen Netzen. Da der Zeitfaktor eine große Rolle spielt, sind Überlegungen zum Einsatz von Genetischen Algorithmen und Genetischem Programmieren sowie die Art und der Aufbau eines Individuums nicht mehr möglich.

Abschließend wird festgestellt, dass der Einsatz von KI-Methoden zur Lösung der gestellten Aufgabe nicht angebracht ist.

4 Zusammenfassung

In diesem Kapitel soll zusammenfassend ein Fazit gezogen werden. Es werden eine Einschätzung der entstandenen Lösung gegeben sowie auftretende Probleme erläutert.

4.1 Einschätzung der aktuellen Lösung

Der bisherige Entwicklungsstand des Fußballroboters „Yellow Panther2“ ist für die Entwickler noch nicht zufrieden stellend, weil nicht genügend Zeit in die Testphase investiert werden konnte. Grund dafür ist vorallem, dass viel Zeit für das ständige Umbauen des Getriebes genutzt und andere Entwicklungsschritte zurückgestellt werden mussten. Es wurden bisher nur die einzelnen Teilaufgaben, wie Ballsuchen, Wandausweichen und Torsuchen, getestet. Ein abschließender Test, bei dem die Teilaufgaben zusammengeführt werden, muss noch durchgeführt werden.

Darüber hinaus können auch noch Verbesserungen vorgenommen werden. Dazu zählt beispielsweise, dass wie in Abschnitt 1.3 beschrieben für die Ballerkennung dynamische Werte verwendet werden könnten, so dass nach jedem Programmstart die aktuellen Werte des Balles ermittelt werden. Dies könnte von Vorteil sein, wenn das Umgebungslicht stark von dem Licht in der Testumgebung abweicht. Es wäre auch denkbar, andere Sensoren zur Ermittlung des Umgebungslichts zu verwenden, deren Werte man auf das Umgebungslicht abbilden kann. Die zusätzlichen Sensoren sollten jedoch nicht so stark von der IR-Strahlung des Sonnenlichtes beeinflusst werden.

Das Design ist vom Aufbau sehr robust und der Roboter sollte auch nicht bei den ersten Verkeilungen auseinanderfallen. Der Roboter ist zwar einerseits sehr robust, aber andererseits auch ziemlich groß im Gegensatz zu den Modellen, die man von den Teams aus Hamburg und Dortmund kennt. Hier würde die Möglichkeit bestehen, den Roboter im Hinblick auf seine Größe noch weiter zu verbessern, in dem statt Modellbaumotoren Legomotoren eingesetzt werden und somit auf das Getriebe verzichtet werden kann.

4.2 Eventuelle auftretende Probleme

Kein Roboter bzw. keine Lösung ist perfekt und es können immer Probleme auftreten. Hierzu zählen zum einen Probleme und Fehler, mit denen man rechnet und zum anderen Probleme, die man nicht vorhersehen kann.

Ein Motorproblem zählt zu den Problemen, mit denen die Entwickler rechnen. Da Modellbaumotoren verwendet werden und die Achsen mit dem Zahnrad nur mit Klebstoff auf der Welle des Motors befestigt sind, kann es immer wieder passieren, dass sich die Achsen samt Zahnrad lösen.

Ein weiteres Problem entsteht, wenn der Servomotor durch ungewollte Krafteinwirkung aus seiner Verankerung gerissen wird. Der Grund für die beiden vorangegangenen Probleme liegt darin, dass es sich bei beiden Motoren nicht um Legoteile handelt und diese mit doppelseitigem Klebeband an den Legosteinen befestigt werden und die notwendigen Zahnräder auch nur provisorisch fixiert werden konnten.

Die Stromversorgung stellt auch eine Fehlerquelle dar. Wie bei Tests aufgefallen ist, reagiert der Roboter nicht mehr so zuverlässig, wenn die Akkupacks an Leistung verlieren. Aus diesem Grund werden für den Roboter bereits zwei Akkupacks verwendet, aber es sollte trotzdem immer darauf geachtet werden, dass für jedes Spiel (Wettkampf) neue Akkus benutzt werden. Da Akkus nicht linear an Leistung verlieren, ist vielleicht ratsam, auf Batterien umzusteigen.

Ein weiteres denkbares Problem wäre, dass während eines Spiels ein oder mehrere Empfänger ausfallen

oder vom gegenerischen Roboter beschädigt werden. Bei einem defekten Empfänger kann es vorkommen, dass das AKSEN-Board nicht mehr korrekt arbeitet bzw. durch einen Kurzschluss komplett ausfällt.

5 Ausblick

Für die Weiterentwicklung des Roboters könnten Technik wie Odometrie und Maus zum Einsatz kommen. Damit kann erkannt werden, ob der Roboter sich bewegt oder mit einem Hindernis kollidiert ist und die Räder durchdrehen.

Darüber hinaus wäre es von Vorteil, wenn Sensorik verwendet werden würde, die unempfindlicher gegen die äußeren Einflüsse ist.

6 Abbildungsverzeichnis

7 Tabellenverzeichnis

8 Literaturverzeichnis

- AKSEN-Handbuch
- Tafelwerk „Formeln und Tabellen“, Verlag: PAETEC, Erscheinungsjahr: 1999, Auflage: 6/7
- Quellcode des Vorgängermodells „Yellow Panther“
- „Wissensverarbeitung“, Autor: Heinsohn/Socher-Ambrosius, Verlag: Spektrum, Erscheinungsjahr: 1999

9 Anhang A – Regeln Robocup (FH) 2005/2006

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.

Offen:

Da Verkeilungen sehr häufig sind, schlagen wir vor, das die Roboter nach 3 Sekunden Blockade versuchen müssen, sich durch eine merkbare Reaktion zu befreien, anderenfalls werden sie verwarnt und nach 3 Verwarnungen disqualifiziert.

- Halte ich für eine sinnvolle Erweiterung die den Spielverlauf unter Umständen lebendiger gestalten könnte, allerdings nur mit nicht geringem Aufwand zu implementieren (Odemetrie oder Strommessung) (Daniel)

Reaktion von den Teams aus Brandenburg !!!

- Wir haben einstimmig beschlossen, dieser Regelung nicht zuzustimmen. Wir denken, dass der Aufwand für das relative geringe Ergebnis (Schiri muß nicht eingreifen) viel zu hoch ist. Immerhin würde es erfordern, zumindest Odometrie einzubauen und darüber hinaus

10 Anhang B – Quellcode

(Änderungen vorbehalten)

```
/*********************************************************************
 **                      RoboCup 2005/06
 **                      Roboter: Yellow Panther2
 **
 **
 **      Fach:           Applikation intelligenter Systeme
 **                      Abschlussaufgabe "Autonomer Fußballroboter"
 **      Autoren:        Regina Wehren
 **                      Katja Orlowski
 **      Betreuer:       Prof. Heinsohn
 **                      Ingo Boersch
 **                      Christiane Lemke
 **                      Prof. Loose
 **                      Prof. Mündemann
 **
 **      Bearbeitungszeitraum: 23.11.2005 - 31.01.2006
 **
 *********************************************************************/


/**Constants.h*********************************************************************/

/* Parameter *****************************************************/

#define LIGHTBAR_VALUE                  10
#define BALL_VALUE                      15
#define SENSOR_GOAL_VALUE               7

#define SENSOR_LEFT_WALL_VALUE          25
#define SENSOR_RIGHT_WALL_VALUE         20
#define SENSOR_CENTER_WALL_VALUE        20
#define SENSOR_BOTH_LEFT_WALL_VALUE     14
#define SENSOR_BOTH_RIGHT_WALL_VALUE    20

#define TIME_BALL_HOLD                  300
#define TIME_WAIT                       100

/* LED Parameter *************************************************/

#define LED_WALL_LEFT                   0
#define LED_WALL_RIGHT                  2
#define LED_WALL_CENTER                 3
#define LED_LIGHTBAR                    1

#define LED_ON                          1
#define LED_OFF                         0

/* Motor Parameter ***********************************************/

#define MOTOR_LEFT                      1
#define MOTOR_RIGHT                     0

#define MOTOR_STOP                      0
#define MOTOR_MAX_PWM_LEFT              10
#define MOTOR_MAX_PWM_RIGHT             10

#define MOTOR_DIRECTION_FORWARD         0
```

```
#define MOTOR_DIRECTION_BACKWARD		1

/* Servo Motor Parameter ********************************************/

#define SERVO_MOTOR				0
#define SERVO_UP				100
#define SERVO_DOWN				50

/* Analog Sensor Parameter ******************************************/

#define SENSOR_LIGHTBAR				9

#define SENSOR_FRONT_RIGHT_1			10
#define SENSOR_FRONT_RIGHT_2			11

#define SENSOR_FRONT_CENTER_1			4
#define SENSOR_FRONT_CENTER_2			5

#define SENSOR_FRONT_LEFT_1			2
#define SENSOR_FRONT_LEFT_2			3

#define SENSOR_BACK_LEFT			1
#define SENSOR_BACK_RIGHT			12

#define SENSOR_BACK_CENTER_1			0
#define SENSOR_BACK_CENTER_2			13

#define SENSOR_SURROUND				8
#define SENSOR_SURROUND_LEFT			7
#define SENSOR_SURROUND_RIGHT			6

#define SENSOR_COMPASS				14

/* Digital Sensor Parameters ****************************************/

#define GOAL_SENSOR_LEFT			7
#define GOAL_SENSOR_CENTER			5
#define GOAL_SENSOR_RIGHT			4

#define GOAL_FREQUENCY_100			5
#define GOAL_FREQUENCY_125			4
#define IR_MAX_ERROR				4
#define GOAL_DETECTION_TIE			4

/**BallDetection.h*************************************************************/

#ifndef _BALLDETECTION_H
#define _BALLDETECTION_H

	#include "Global.h"

	// Methode zum Aufspüren des Balls
	void findBall(void);
	// Methode zur Ermittlung der Ballposition
	enum ballPosition getBallPosition(void);
	// Methode zum Prüfen, ob Ball die Lichtschranke durchbricht
	enum boolean hasBall(void);

#endif
```

```
/**BallDetection.c*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "BallDetection.h"
#include "WallDetection.h"
#include "Constants.h"
#include "Global.h"
#include "Common.h"
#include "Motor.h"
#include "Shooter.h"

/*
 *  Methode zum Aufspüren des Balls
 *  wird aufgerufen, wenn die Zustandsmaschine sich im Zustand SEARCH_BALL befindet
 *
 *  Überprüfen, ob Ball Lichtschranke durchbricht - dann Zustandswechsel SEARCH_GOAL
 *  Ball nicht vorhanden, Ball aufspüren
 *  Ball nicht zu sehen, Ball suchen durch Bewegen auf dem Spielfeld
 */
void findBall(void)
{
	enum ballPosition pos;

	pos = getBallPosition();
	lcd_cls();
	switch(pos)
	{
		case NONE :
			lcd_puts("none");
			moveForward(10, 10);
			break;
		case FRONTLEFT :
			lcd_puts("fl");
			turnLeft(7, 7);
			break;
		case FRONTLEFT | FRONTCENTER :
			lcd_puts("flc");
			turnLeft(7, 7);
			break;
		case FRONTCENTER :
			lcd_puts("fc");
			moveForward(10, 10);
			break;
		case FRONTRIGHT :
			lcd_puts("fr");
			turnRight(7, 7);
			break;
		case FRONTRIGHT | FRONTCENTER :
			lcd_puts("frc");
			turnRight(7, 7);
			break;
		case BACKCENTER :
			lcd_puts("bc");
			turnRight(10, 10);
			break;
		case BACKLEFT :
			lcd_puts("bl");
```

```
                        turnLeft(7, 7);
                        break;
                case BACKCENTER | BACKLEFT:
                        lcd_puts("bcl");
                        turnLeft(7, 7);
                        break;
                case BACKRIGHT :
                        lcd_puts("br");
                        turnRight(7, 7);
                        break;
                case BACKCENTER | BACKRIGHT:
                        lcd_puts("bcr");
                        turnRight(7, 7);
                        break;
                case BACKCENTER | BACKRIGHT | BACKLEFT:
                        lcd_puts("bcrl");
                        turnRight(10, 10);
                        break;
                default :
                        lcd_puts("default");
                        moveForward(10, 10);
                        break;
        }
        sleep(20);
}

/*
 *  Methode zur Ermittlung der Ballposition Abfragen aller Analogports, Vergleich mit Umgebungslicht
 *  Vergleich der ermittelten Werte der Sensoren mit einem Schwellwert (BALL_VALUE)
 *   wird der Ball nicht erkannt, ist der ermittelte Wert der Sensoren 0
 *   je höher der Wert, desto besser wird der Ball erkannt und desto näher ist der Roboter am Ball
 *
 *  return Ballpostion (FRONTLEFT, FRONTCENTER, FRONTRIGHT, BACKRIGHT, BACKCENTER,
 *  BACKLEFT)
 */
enum ballPosition getBallPosition(void) {

        enum ballPosition pos = NONE;
        unsigned char valueFrontCenter, valueFrontLeft, valueFrontRight, valueBackLeft, valueBackRight,
                        valueBackCenter;

        valueFrontLeft = max(compareToSurroundPort(SENSOR_SURROUND_LEFT,
                         SENSOR_FRONT_LEFT_1),   compareToSurroundPort
                        (SENSOR_SURROUND_LEFT, SENSOR_FRONT_LEFT_2));
        valueFrontCenter = max( compareToSurround(SENSOR_FRONT_CENTER_1),
                                        compareToSurround(SENSOR_FRONT_CENTER_2));
        valueFrontRight = max(compareToSurroundPort(SENSOR_SURROUND_RIGHT,
                         SENSOR_FRONT_RIGHT_1),  compareToSurroundPort
                        (SENSOR_SURROUND_RIGHT, SENSOR_FRONT_RIGHT_2));
        valueBackLeft   = compareToSurround(SENSOR_BACK_LEFT);
        valueBackCenter = max(compareToSurround(SENSOR_BACK_CENTER_1), compareToSurround
                        (SENSOR_BACK_CENTER_2));
        valueBackRight = compareToSurround(SENSOR_BACK_RIGHT);

        if (5 < valueFrontCenter)
                pos |= FRONTCENTER;

        if (BALL_VALUE < valueFrontLeft)
                pos |= FRONTLEFT;
```

```
	if (BALL_VALUE < valueFrontRight)
		pos |= FRONTRIGHT;

	if (BALL_VALUE < valueBackLeft)
		pos |= BACKLEFT;

	if (BALL_VALUE < valueBackCenter)
		pos |= BACKCENTER;

	if (BALL_VALUE < valueBackRight)
		pos |= BACKRIGHT;

	return pos;
}

/*
 *  Methode zum Prüfen, ob Ball die Lichtschranke durchbricht und der Käfig geschlossen werden kann
 *
 *  return TRUE, wenn Ball Lichtschranke durchbricht
 *  return FALSE, wenn Ball Lichtschranke nicht durchbricht
 */
enum boolean hasBall(void) {

	enum boolean value = FALSE;

	value = (analog(SENSOR_LIGHTBAR) < LIGHTBAR_VALUE) ? FALSE : TRUE;

	if(value == TRUE)
	{
		shooterDown();
		if(timer == FALSE)
		{
			clear_time();
			timer = TRUE;
		}

	}
	else
		shooterUp();

	return value;
}


/**Common.h*************************************************************************/

#ifndef _COMMON_H
#define _COMMON_H


	// Methode zum Vergleich eines Analogportes mit dem Umgebungslicht
	unsigned char compareToSurround(unsigned char port1);
	// Methode zum Vergleich eines Analogportes mit dem Umgebungslicht an beliebigen Port
	unsigned char compareToSurroundPort(unsigned char surround, unsigned char port1);

	// Ermittelt das Maximum
	int max(unsigned char w1, unsigned char w2);
	// Ermittelt das Minimum
	int min(unsigned char w1, unsigned char w2);
	// ermittelt den absoluten Wert
	unsigned char absoluteValue(int value);
```

```
	// Methode zum Ermitteln der Werte der Wanderkennungssensoren
	void getIR(int port1, int port2, int port3, unsigned char *port1Value, unsigned char *port2Value,
			unsigned char *port3Value);

	// An- und Abschalten von LEDs
	void leds(unsigned char value);
	// Initialiserung des Hauptprogrammes oder von Hilfsprogrammen
	void init(void);

	// Ausgabe Ballsensoren
	void outputBallSensor(void);
	// Ausgabe Umweltsensoren
	void outputSurroundSensor(void);
	// Ausgabe Wandsensoren
	void outputWallSensor(void);
	// Ausgabe Torsensoren
	void outputGoalSensor(void);
	// Schussanlagentest
	void shooterTest(void);

#endif


/**Common.c***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "Common.h"
#include "Global.h"
#include "Constants.h"
#include "Shooter.h"

/*
 *  Methode zum Vergleich eines Analogportes für die Ballerkennung mit dem Umgebungslicht
 */
unsigned char compareToSurround(unsigned char port1)
{
	return compareToSurroundPort(SENSOR_SURROUND, port1);
}

/*
 *  Methode zum Vergleich eines Analogportes für die Ballerkennung
 *  mit dem Umgebungslicht an einen beliebbigen Port
 */
unsigned char compareToSurroundPort(unsigned char surround, unsigned char port1)
{
	return (analog(surround) > analog(port1)) ? analog(surround) - analog(port1) : 0;
}
/*
 *  Methode zum Ermitteln des Maximums von zwei übergebenen Werten
 */
int max(unsigned char w1, unsigned char w2)
{
	return (w1 > w2) ? w1: w2;
}
```

```
/*
 *  Methode zum Ermitteln des Minimums von zwei übergebenen Werten
 */
int min(unsigned char w1, unsigned char w2)
{
	return (w1 < w2) ? w1 : w2;
}

/*
 *  Methode zum Ermitteln des Absolutwertes eines eingebenen Wertes
 */
unsigned char absoluteValue(int value)
{
	return (value < 0) ? (unsigned char) -value : (unsigned char) value;
}

/*
 *  Methode zum An- bzw. Ausschalten der LEDs
 *  Notwendig für die Wanderkennung
 */
void leds(unsigned char value)
{
	led(LED_WALL_LEFT,  value);
	led(LED_WALL_RIGHT, value);
	led(LED_WALL_CENTER, value);
}

/*
 *  Methode zum Ermitteln der Werte der Wanderkennungssensoren
 *
 *  Werte der Sensoren (links und rechts) mit ausgeschalteten LEDs messen - Umgebungslicht
 *  Werte der Sensoren (links und rechts) mit angeschalteten LEDs messen - Wert2
 *
 *  wenn Umgebungslicht > Wert2, dann subtrahiere und übergebe Wert an port1Value/port2Value
 *  andernfalls port1Value/port2Value auf 0 setzen (keine Wand)
 */
void getIR(int port1, int port2, int port3, unsigned char *port1Value, unsigned char *port2Value, unsigned char
*port3Value)
{
	unsigned char analogLight1  = 0, analogLight2  = 0, analogLight3 = 0;
	unsigned char ambientLight1     = 0, ambientLight2 = 0, ambientLight3 = 0;

	ambientLight1 = analog(port1);
	ambientLight2 = analog(port2);
	ambientLight3 = analog(port3);
	leds(LED_ON);
	sleep(10);
	analogLight1 = analog(port1);
	analogLight2 = analog(port2);
	analogLight3 = analog(port3);
	leds(LED_OFF);

	*port1Value = ( ambientLight1 > analogLight1 ) ? ambientLight1 - analogLight1 : 0;
	*port2Value = ( ambientLight2 > analogLight2 ) ? ambientLight2 - analogLight2 : 0;
	*port3Value = ( ambientLight3 > analogLight3 ) ? ambientLight3 - analogLight3 : 0;
}
```

```
/*
 *  Methode zum Initialisieren
 *  Initialiserung des Hauptprogrammes oder von Hilfsprogrammen
 */
void init(void)
{
	// Initalisierung Torerkennung
	unsigned char irFrequence = (dip_pin(0) == 0) ? GOAL_FREQUENCY_100 :
GOAL_FREQUENCY_125;
	int dipValue;

	// Initialisierung der IR-Fernsteuerungsempfänger
	mod_ir0_takt(irFrequence);
	mod_ir1_takt(irFrequence);
	mod_ir3_takt(irFrequence);
	mod_ir0_maxfehler(IR_MAX_ERROR);
	mod_ir1_maxfehler(IR_MAX_ERROR);
	mod_ir3_maxfehler(IR_MAX_ERROR);

	// Starten von bestimmten Programmsequenzen
	do {
		dipValue = (dip_pin(1) << 2) + (dip_pin(2) << 1) + dip_pin(3);
		switch(dipValue){
			case 1 :
				outputBallSensor();
				break;
			case 2 :
				outputSurroundSensor();
				break;
			case 3 :
				outputWallSensor();
				break;
			case 4 :
				outputGoalSensor();
				break;
			case 5 :
				shooterTest();
				break;

			default:
				lcd_cls();
				lcd_puts("Dip-Wert: ");
				lcd_ubyte(dipValue);
				sleep(10);
			break;
		}
	}while(dipValue != 0);


	led(LED_LIGHTBAR, LED_ON);
	leds(LED_OFF);

}
```

```
/*
 * Ausgabe der Ballsensorik
 */
void outputBallSensor(void)
{
	unsigned char valueFrontCenter, valueFrontLeft, valueFrontRight, valueBackLeft, valueBackRight,
			valueBackCenter;

	valueFrontLeft		= max(compareToSurround(SENSOR_FRONT_LEFT_1),
				compareToSurround(SENSOR_FRONT_LEFT_2));
	valueFrontCenter	= max(compareToSurround(SENSOR_FRONT_CENTER_1),
				compareToSurround(SENSOR_FRONT_CENTER_2));
	valueFrontRight		= max(compareToSurround(SENSOR_FRONT_RIGHT_1);
				compareToSurround(SENSOR_FRONT_RIGHT_2));
	valueBackLeft		= compareToSurround(SENSOR_BACK_LEFT);
	valueBackCenter		= max(compareToSurround(SENSOR_BACK_CENTER_1),
				compareToSurround(SENSOR_BACK_CENTER_2));
	valueBackRight		= compareToSurround(SENSOR_BACK_RIGHT);

	lcd_cls();
	lcd_ubyte(valueFrontLeft);
	lcd_puts(" ");
	lcd_ubyte(valueFrontCenter);
	lcd_puts(" ");
	lcd_ubyte(valueFrontRight);
	lcd_setxy(1, 0);
	lcd_ubyte(valueBackLeft);
	lcd_puts(" ");
	lcd_ubyte(valueBackCenter);
	lcd_puts(" ");
	lcd_ubyte(valueBackRight);
	sleep(50);
}

/*
 * Ausgabe der Umweltsensorik
 */
void outputSurroundSensor(void)
{
	unsigned char leftSensor  = 0;
	unsigned char rightSensor = 0;
	unsigned char centerSensor = 0;

	getIR(SENSOR_FRONT_LEFT_2, SENSOR_FRONT_RIGHT_1, SENSOR_FRONT_CENTER_1,
		&leftSensor, &rightSensor, &centerSensor);

	lcd_cls();
	lcd_puts("Top: ");
	lcd_ubyte(analog(SENSOR_SURROUND));
	lcd_setxy(1, 0);
	lcd_puts("Left: ");
	lcd_ubyte(analog(SENSOR_SURROUND_LEFT));
	lcd_puts(" Right: ");
	lcd_ubyte(analog(SENSOR_SURROUND_RIGHT));
	sleep(50);

}
```

```
/*
 * Ausgabe der Wandsensorik
 */
void outputWallSensor(void)
{
	unsigned char leftSensor  = 0;
	unsigned char rightSensor = 0;
	unsigned char centerSensor = 0;

	getIR(SENSOR_FRONT_LEFT_2, SENSOR_FRONT_RIGHT_1, SENSOR_FRONT_CENTER_1,
		&leftSensor, &rightSensor, &centerSensor);

	lcd_cls();
	lcd_puts("l:  ");
	lcd_ubyte(leftSensor);
	lcd_puts(" r: ");
	lcd_ubyte(rightSensor);
	lcd_setxy(1, 0);
	lcd_puts(" c: ");
	lcd_ubyte(centerSensor);
	sleep(50);

}

/*
 * Ausgabe der Torsensorik
 */
void outputGoalSensor(void)
{
	lcd_cls();
	lcd_puts("L: ");
	lcd_ubyte(mod_ir3_status());
	lcd_puts(" M: ");
	lcd_ubyte(mod_ir1_status());
	lcd_puts(" R: ");
	lcd_ubyte(mod_ir0_status());
	lcd_setxy(1,0);
	lcd_puts("Kompass: ");
	lcd_ubyte(analog(SENSOR_COMPASS));
	sleep(50);

}

/*
 * Test für die Schussanlage
 */
void shooterTest()
{
	lcd_cls();
	lcd_puts("Down");
	shooterDown();
	lcd_cls();
	lcd_puts("up");
	shooterUp();
}
```

```
/**Global.h*************************************************************************/

#ifndef _GLOBAL_H
#define _GLOBAL_H


	enum boolean
	{
		TRUE		= 1,
		FALSE		= 0
	};

	enum states
	{
		SEARCH_BALL		= 0,
		SEARCH_GOAL		= 1,
		TROUBLE			= 2,
		SHOT		= 3
	};

	enum compassDirection
	{
		NORTH		= 0,		// Tor liegt vor dem Roboter
		NORTHEAST	= 1,		// Tor liegt vorne rechts vom Roboter
		EAST		= 2,		// Tor liegt rechts vom Roboter
		SOUTHEAST	= 3,		// Tor liegt hinten rechts vom Roboter
		SOUTH		= 4,		// Tor liegt hinterm Roboter
		SOUTHWEST	= 5,		// Tor liegt hinten rechts vom Roboter
		WEST		= 6,		// Tor links vom Roboter
		NORTHWEST	= 7		// Tor liegt vorne links vom Roboter

	};

	enum objectPosition
	{
		NONE		= 0,
		LEFT		= 1,	// Hindernis vorne links
		RIGHT		= 2,	// Hindernis vorne rechts
		CENTER		= 4	// Hindernis vorne mitte
	};

	enum ballPosition
	{
		NONE			= 0,	// Ball wird nicht gesehen
		FRONTLEFT		= 1,	// Ball liegt vorne links
		FRONTCENTER		= 2,	// Ball liegt vorne mittig
		FRONTRIGHT		= 4,	// Ball liegt vorne rechts
		BACKCENTER		= 8,	// Ball liegt hinten mittig
		BACKLEFT		= 16,	// Ball liegt hinten links
		BACKRIGHT		= 32	// Ball liegt hinten rechts
	};

	enum goalPosition
	{
		NONE		= 0,	// Kein Tor
		LEFT		= 1,	// Tor liegt leicht links vor dem Roboter
		CENTER		= 2,	// Tor liegt genau vor dem Roboter
		RIGHT		= 4	// Tor liegt leicht rechts vor dem Roboter
	};
```

```
 // Definieren der global Varibalen

        // Variable enthält den aktuellen Zustand
        extern enum states state;
        // Variable enthält die alte Ballposition
        extern enum ballPosition oldPosition;
        // Variable enthält die Position des Hindernisses
        extern enum objectPosition object;
        // Variable merkt sich ob der Timer aktiv ist
        extern enum boolean timer;
        // Geschwindigkeit für Motor rechts
        extern int speedRight;
        // Geschwindigkeit für Motor links
        extern int speedLeft;

#endif


/**Global.c*************************************************************************/

#include "Global.h"
#include "Constants.h"

//Deklarieren und Initialisieren der globalen Variablen

// Variable enthält den aktuellen Zustand
enum states state = SEARCH_BALL;
// Variable enthält die alte Ballposition
enum ballPosition oldPosition = NONE;
// Variable enthält die Position des Hindernisses
enum objectPosition object = NONE;
// Variable merkt sich ob der Timer aktiv ist
enum boolean timer = FALSE;

// Geschwindigkeit für Motor rechts
int speedRight = MOTOR_MAX_PWM_RIGHT;
// Geschwindigkeit für Motor links
int speedLeft  = MOTOR_MAX_PWM_LEFT;


/**GoalDetection.h*******************************************************************/

#ifndef _GOALDETECTION_H
#define _GOALDETECTION_H

        #include "Global.h"

        //Methode zum Aufspüren des Tores
        void findGoal(void);

        //Methode zum Auslesen des Kompasswertes
        enum compassDirection getCompassDirection(void);

        //Methode zum Auswerten der IR-Fernsteuerungsempfänger
        enum goalPosition getGoalSensor(void);

#endif
```

```
/**GoalDetection.c*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "GoalDetection.h"
#include "BallDetection.h"
#include "WallDetection.h"

#include "Global.h"
#include "Constants.h"
#include "Shooter.h"
#include "Motor.h"

/*
 *  Methode zum Aufspüren des Tors
 *  wird aufgerufen, wenn die Zustandsmaschine sich im Zustand SEARCH_GOAL befindet
 *
 *  Überprüfen, ob es eine Kollision gibt
 *  Überprüfen, ob Ball noch vorhanden ist - wenn Ball verloren wurde - Zustandswechsel SEARCH_BALL
 *
 *  Ball vorhanden und keine Kollision, Torsuchen aufspüren
 *  Je nach Torrichtung bewegen (vorwärts fahren, drehen)
 */
void findGoal(void)
{
        enum goalPosition pos;
        enum compassDirection goal;


        // ermitteln der Werte der IR-Fernsteuerungsempfänger
        // wenn pos = CENTER, dann Zustandswechsel,
        // andernfalls drehen bzw. Kompass abfragen (bei NONE)
        pos = getGoalSensor();
        goal = getCompassDirection();

        switch (goal)
        {
                case NORTH :
                        switch(pos)
                        {
                                case LEFT :
                                        moveForward(6, 10);
                                        break;
                                case LEFT | CENTER:
                                        state = SHOT;
                                        break;
                                case CENTER :
                                        state = SHOT;
                                        break;
                                case RIGHT | CENTER:
                                        state = SHOT;
                                        break;
                                case RIGHT :
                                        moveForward(10, 6);
                                        break;
                                case RIGHT | CENTER | LEFT:
                                        state = SHOT;
                                        break;
```

```
                                        case RIGHT | LEFT:
                                                state = SHOT;
                                                break;
                                        case NONE :
                                        {
                                                break;
                                        }
                                }
                                break;
                        case NORTHEAST :
                                turnLeft(8, 8);
                                break;
                        case EAST :
                                turnLeft(10, 10);
                                break;
                        case SOUTHEAST :
                                turnLeft(10, 10);
                                break;
                        case SOUTH :
                                turnRight(10, 10);
                                break;
                        case SOUTHWEST :
                                turnRight(10, 10);
                                break;
                        case WEST :
                                turnRight(10, 10);
                                break;
                        case NORTHWEST :
                                turnRight(8, 8);
                                break;
                        default :
                                lcd_puts("ERROR Goal");
                                break;
        }
}

/*
 *  Methode zum Auslesen des Kompasswertes
 *  Auswertung in welche Richtung der Roboter steht
 *  return Himmelsrichtung (NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST,
 *  NORTHWEST)
 */
enum compassDirection getCompassDirection(void)
{
        unsigned char compassDir = analog(SENSOR_COMPASS);

        if ((compassDir > 55) && (compassDir <= 75))
                return NORTH;

        if ((compassDir > 75) && (compassDir <= 105))
                return NORTHEAST;

        if ((compassDir > 105) && (compassDir <= 126))
                return EAST;

        if ((compassDir > 126) && (compassDir <= 145))
                return SOUTHEAST;

        if ((compassDir > 145) && (compassDir <= 165))
                return SOUTH;
```

```
	if ((compassDir > 165) && (compassDir <= 185))
		return SOUTHWEST;

	if ((compassDir > 185) && (compassDir <= 200))
		return WEST;

	if ((compassDir > 30) && (compassDir <= 55))
		return NORTHWEST;

	return NORTH;
}
/*
 *  Methode zum Auswerten der IR-Fernsteuerungsempfänger
 *  Notwendig für die Torschuss
 */
enum goalPosition getGoalSensor(void)
{
	unsigned char front, left, right;
	enum goalPosition pos = NONE;

	left  = mod_ir3_status();
	front = mod_ir1_status();
	right = mod_ir0_status();

	if(left > GOAL_DETECTION_TIE)
		pos = LEFT;

	if(right > GOAL_DETECTION_TIE)
		pos = RIGHT;

	if(front > GOAL_DETECTION_TIE)
		pos = CENTER;

	return pos;
}


/**Main.c*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "Constants.h"
#include "Global.h"
#include "Common.h"
#include "WallDetection.h"
#include "BallDetection.h"
#include "GoalDetection.h"
#include "Motor.h"
#include "Shooter.h"

/* Hauptprogrammroutine */
void AksenMain(void)
{
	init();
	while(1)
	{
		// wenn es Roboter zu an der Wand ist, dann ausweichen
		// Zustandswechsel
```

```
		if (state != SHOT)
		{
			if(TRUE == hasCollision())
				state = TROUBLE;
			else
			{
				// wenn Roboter im Besitz des Balles, dann Tor suchen
				// Zustandswechsel
				if(TRUE == hasBall())
				{
					if(TRUE == isInTime())
						state = SEARCH_GOAL;
					else
						state = SEARCH_BALL;
				}
				// wenn Roboter nicht im Besitz des Balles, dann Ball suchen
				// Zustandswechsel
				else
					state =SEARCH_BALL;
			}
		}

		switch(state)
		{
			case SEARCH_BALL :
				findBall();
				break;
			case SEARCH_GOAL :
				findGoal();
				break;
			case TROUBLE :
				findWall();
				break;
			case SHOT :
				shot();
				break;
		}
	}
}

/**Motor.h*************************************************************************/

#ifndef _MOTOR_H
#define _MOTOR_H

	//Methode, zum Vorwärtsfahren
	void moveForward(int pwmLeft, int pwmRight);

	//Methode, zum Rückwärtsfahren
	void moveBackward(int pwmLeft, int pwmRight);

	//Methode, zum Linksdrehen
	void turnLeft(int pwmLeft, int pwmRight);

	//Methode zum Rechtsdrehen
	void turnRight(int pwmLeft, int pwmRight);

	// Methode zum Stoppen der Motoren
	void fullStop(void);
#endif
```

```
/**Motor.c*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "Motor.h"
#include "Constants.h"
#include "Global.h"

/*
 *  Methode, zum Vorwärtsfahren
 */
void moveForward(int pwmLeft, int pwmRight)
{
	motor_richtung(MOTOR_LEFT,  MOTOR_DIRECTION_FORWARD);
	motor_richtung(MOTOR_RIGHT, MOTOR_DIRECTION_FORWARD);
	motor_pwm(MOTOR_LEFT,  pwmLeft);
	motor_pwm(MOTOR_RIGHT, pwmRight);
}

/*
 *  Methode, zum Rückwärtsfahren
 */
void moveBackward(int pwmLeft, int pwmRight)
{
	motor_richtung(MOTOR_LEFT,  MOTOR_DIRECTION_BACKWARD);
	motor_richtung(MOTOR_RIGHT, MOTOR_DIRECTION_BACKWARD);
	motor_pwm(MOTOR_LEFT,  pwmLeft);
	motor_pwm(MOTOR_RIGHT, pwmRight);
}

/*
 *  Methode, zum Linksdrehen
 */
void turnLeft(int pwmLeft, int pwmRight)
{
	motor_richtung(MOTOR_LEFT,  MOTOR_DIRECTION_BACKWARD);
	motor_richtung(MOTOR_RIGHT, MOTOR_DIRECTION_FORWARD);
	motor_pwm(MOTOR_LEFT,  pwmLeft);
	motor_pwm(MOTOR_RIGHT, pwmRight);
}

/*
 *  Methode, zum Rechtsfahren
 */
void turnRight(int pwmLeft, int pwmRight)
{
	motor_richtung(MOTOR_LEFT,  MOTOR_DIRECTION_FORWARD);
	motor_richtung(MOTOR_RIGHT, MOTOR_DIRECTION_BACKWARD);
	motor_pwm(MOTOR_LEFT,  pwmLeft);
	motor_pwm(MOTOR_RIGHT, pwmRight);
}
```

```
/*
 *  Methode, zum Stoppen der Motoren
 */
void fullStop(void)
{
	motor_pwm(MOTOR_LEFT,  MOTOR_STOP);
	motor_pwm(MOTOR_RIGHT, MOTOR_STOP);
}


/**Shooter.h*************************************************************************/

#ifndef _SHOOTER_H
#define _SHOOTER_H

	#include "Global.h"

	// Methode zum Schießen Balls
	void shot(void);
	// Methode zum Prüfen des Zeitlimits
	enum boolean isInTime();
	// Methode zum Öffnen des Käfigs
	void shooterUp();
	// Methode zum Schließen des Käfigs
	void shooterDown();

#endif


/**Shooter.c*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "Shooter.h"
#include "Constants.h"
#include "Global.h"
#include "BallDetection.h"
#include "Motor.h"

/*
 *  Methode, zum Schießen des Balles nach dem Regelwerk
 *  Ball freigeben, handbreiten Abstand zum Ball einnehmen
 *  1 Sekunde warten und schießen
 */
void shot(void)
{
	if(TRUE == hasBall())
	{
		// Käfig öffnen
		shooterUp();
		// Abstand einnehmen und Warten
		moveBackward(10,10);
		sleep(700);
		fullStop();
		sleep(TIME_WAIT);
		// Schießen
		shooterDown();
		moveForward(10,10);
```

```
                sleep(650);
                shooterUp();
                fullStop();
                sleep(300);
        }
        else
        {
                shooterUp();
        }
}

/*
 *  Methode, zum Prüfen des Zeitlimits
 */
enum boolean isInTime()
{
        enum boolean value = FALSE;

        // Zeitlimit überschritten und Timer ist aktiv
        if(TRUE == timer && akt_time() >= TIME_BALL_HOLD)
        {
                // dann gebe Ball frei
                value = FALSE;
                shooterUp();
                moveBackward(10,10);
                sleep(700);
                fullStop();
                sleep(TIME_WAIT);
                timer = FALSE;
        }
        else
                value = TRUE;

        return value;
}

/*
 *  Methode, zum Bewegen des Greifarms auf 100° (Öffnen des Käfigs)
 */
void shooterUp()
{
        servo_arc(SERVO_MOTOR, SERVO_UP);
        sleep(800);
}

/*
 * shooterDownMethode, zum Bewegen des Greifarms auf 50° (Schließen des Käfigs)
 */
void shooterDown()
{
        servo_arc(SERVO_MOTOR, SERVO_DOWN);
        sleep(800);
}
```

```
/**WallDetection.h*********************************************************************/

#ifndef _WALLDETECTION_H
#define _WALLDETECTION_H

	#include "Global.h"

	// Methode zur Hindernis erkennen und ausweichen
	void findWall(void);

	//  Methode zum Überprüfen, ob es eine Kollision gibt
	enum boolean hasCollision(void);

#endif


/****WallDetection.c*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <regc515c.h>
#include <stub.h>

#include "Constants.h"
#include "Global.h"
#include "Common.h"
#include "Motor.h"
#include "WallDetection.h"
#include "BallDetection.h"

/*
 *  Methode zur Hinderniserkennung
 *
 *  Überprüfen des aktuellen Status und Reaktion
 *
 *  Wand links erkannt           -> rechts drehen
 *  Wand rechts erkannt           -> links drehen
 *  Wand rechts und links erkannt -> rückwärts fahren
 */
void findWall(void)
{
	switch (object)
	{
		case LEFT :
			turnRight(8, 8);
			break;
		case RIGHT :
			turnLeft(8, 8);
			break;
		case LEFT | RIGHT :
			turnLeft(8, 8);
			break;
		case CENTER :
			fullStop();
			sleep(100);
			moveBackward(7, 9);
			sleep(1000);
			fullStop();
			sleep(100);
			break;
		case LEFT | CENTER :
			turnRight(8, 8);
```

```
                        break;
                case RIGHT | CENTER :
                        turnLeft(8, 8);
                        break;
                case LEFT | RIGHT | CENTER :
                        turnLeft(8, 8);
                        break;
                case NONE :
                        moveForward(10, 10);
                        break;
        }
}

/*
 *  Methode zum Überprüfen, ob es eine Kollision gibt
 *
 *  return TRUE, wenn es Kollision gibt - wo etwas im Weg ist, wird in der Variable object gespeichert
 *  return FALSE, wenn Weg frei ist
 */
enum boolean hasCollision(void)
{
        unsigned char leftSensor  = 0;
        unsigned char rightSensor = 0;
        unsigned char centerSensor = 0;
        unsigned char valueFrontLeft, valueFrontRight, valueFrontCenter;

        // Aufruf der Funktion getIR();
        getIR(SENSOR_FRONT_LEFT_2, SENSOR_FRONT_RIGHT_1, SENSOR_FRONT_CENTER_1,
                &leftSensor, &rightSensor, &centerSensor);

        // Ballsensoren überprüfen ob Hindernis der Ball ist
        valueFrontCenter= max(compareToSurround(SENSOR_FRONT_CENTER_1), compareToSurround
                                (SENSOR_FRONT_CENTER_2));
        valueFrontLeft   = max(compareToSurroundPort(SENSOR_SURROUND_LEFT,
                                SENSOR_FRONT_LEFT_1),   compareToSurroundPort
                               (SENSOR_SURROUND_LEFT, SENSOR_FRONT_LEFT_2));
        valueFrontRight = max(compareToSurroundPort(SENSOR_SURROUND_RIGHT,
                                SENSOR_FRONT_RIGHT_1), compareToSurroundPort
                               (SENSOR_SURROUND_RIGHT, SENSOR_FRONT_RIGHT_2));

        object = NONE;

        /*
         * Vergleich der ermittelte Werte der Wandsensoren mit vorgesehenen Schwellwert
         * je größer der Wert für leftSensor bzw rightSensor, desto näher ist der Roboter
         * an der Wand
         */
        if( BALL_VALUE >= valueFrontCenter)
                if (SENSOR_CENTER_WALL_VALUE < centerSensor)
                        object |= CENTER;

        if( BALL_VALUE >= valueFrontLeft)
                if (SENSOR_LEFT_WALL_VALUE < leftSensor)
                        object |= LEFT;

        if( BALL_VALUE >= valueFrontRight)
                if (SENSOR_RIGHT_WALL_VALUE < rightSensor)
                        object |= RIGHT;

        return (object != NONE) ? TRUE : FALSE;
}
```