Fachgebiet Simulation und Systemoptimierung
Fachbereich Informatik
Technische Universität Darmstadt

Hindernis- und Spielererkennung für Humanoidroboter beim RoboCup

Obstacle and Player Detection for Humanoid Robots at RoboCup

Diplomarbeit

von

Tobias Ludwig

Darmstadt, Oktober 2006

Aufgabenstellung: Prof. Dr. Oskar von Stryk
Betreuer: Dipl.-Inform. Dirk Thomas

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Oktober 2006 Tobias Ludwig

Zusammenfassung

Das Fachgebiet Simulation und Systemoptimierung der Technischen Universität Darmstadt beschäftigt sich mit der Forschung an mobilen autonomen Robotersystemen. Ein wichtiger Aspekt ist dabei die Entwicklung der Künstlichen Intelligenz zur Steuerung verschiedener Robotermodelle, von denen einige in regelmäßigen Abständen an Wettbewerben des RoboCups teilnehmen.

Diese Arbeit beschreibt die Entwicklung einer Hindernis- und Spielererkennung als Erweiterung der vorhandenen Steuerungssoftware *RoboCup06App*, die im Rahmen der Humaniodliga des RoboCups Anwendung findet. Eine ebenfalls entworfene Modellierung der neu gewonnenen Daten ermöglicht schließlich deren Verwendung bei der Verhaltensprogrammierung, was beispielhaft durch eine Hindernisvermeidung demonstriert werden konnte.

Abstract

The Simulation and Systems Optimization Group as a part of the Computer Science Department of Technische Universität Darmstadt deals with research regarding mobile, autonomous robotic systems. One major research aspect focuses on developing Artificial Intelligence in order to control miscellaneous robot models. Some of these models regularly participate in competitions of RoboCup. This paper describes the development of an obstacle and player detection as an extension to *RoboCup06App*, a robot control software which has already been employed in RoboCup *Humanoid League*. A modelling of the newly extracted data, also designed as part of this work, finally allows an application by the behavior layer. These functions are demonstrated using an obstacles avoidance behavior.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Ziel der Arbeit

Ziel der Diplomarbeit ist die Entwicklung einer Software, welche die Erkennung von Hindernissen ermöglicht. Die Software soll bei autonomen, im Rahmen des RoboCups[1] teilnehmenden, Humanoidrobotern Verwendung finden. Aufbauend auf der bereits vorhandenen Steuerungssoftware *RoboCup06App* wird mit der Implementierung einer Hinderniserkennung eine Verbesserung der Leistungsfähigkeit angestrebt. Von besonderem Interesse ist dabei die Möglichkeit, Roboterspieler von allgemeinen Hindernissen unterscheiden zu können.

Bevor der Roboter seine Umwelt berücksichtigen kann, muss er diese zunächst erfassen. Durch die Regeln des RoboCups eingeschränkt, erfolgt dies mittels eines Kamerasystems. Damit der Roboter zusätzlich in die Lage versetzt werden kann, auf Hindernisse und Spieler in seiner Umgebung zu reagieren, die sich nicht unmittelbar in seinem Blickfeld befinden, muss er sich seine zuvor gesehene Umwelt „gemerkt“ haben. Dazu ist die Implementation einer sinnvollen Modellierung der neuen Erkenntnisse notwendig.

1.2 Aufbau der Arbeit

Nachdem in **Kapitel 2** die Beweggründe für diese Arbeit erläutert werden, erfolgt eine Darstellung der Grundlagen, insbesondere der vorhandenen Steuerungssoftware (**Kapitel 3**). **Kapitel 4** beschreibt das Konzept der neu zu entwickelnden Komponenten und **Kapitel 5** geht detailliert auf deren Funktionsweise ein. Durch Tests gewonnene Resultate werden in **Kapitel 6** dargestellt. **Kapitel 7** fasst die Arbeit zusammen und gibt einen Ausblick über mögliche Weiterentwicklungen in der Zukunft.

[1] RoboCup `http://www.robocup.org`

2 Motivation

Der spielerische Kontext des RoboCups mag schnell über die Komplexität der Aufgabe, einen autonomen Roboter zum Fußballspielen zu befähigen, hinwegtäuschen. Zur Lösung der sich stellenden Aufgaben müssen die Roboter die Umwelt zunächst über ein Kamerasystem wahrnehmen, um situationsbedingt entscheiden zu können, wie sie darauf zu reagieren haben. Die Umsetzung der geplanten Vorgehensweise erfolgt anschließend durch entsprechenden Anweisungen an den Bewegungsapparat.

Die Humanoidliga, die den Rahmen dieser Arbeit bildet, ist eine vergleichsweise junge Disziplin und teilweise noch mit der Lösung von Grundaufgaben beschäftigt. Neben der Verbesserung von technischen Voraussetzungen, wie etwa der Koordination des Laufens auf zwei Beinen, besteht vor allem im Bereich der Steuerungssoftware Entwicklungsbedarf. Am Fachgebiet Simulation und Systemoptimierung wurde bereits eine solche Software entwickelt, die grundsätzlich ein Fußballspielen ermöglicht und mit Erfolg bei der RoboCup Weltmeisterschaft 2006 in Bremen zum Einsatz gelangte.

Die Humanoidroboter der teilnehmenden Teams verfügten jedoch nur teilweise über die Fähigkeit Hindernisse (d.h. Tore, Mitspieler, etc.) zu erkennen und entsprechend darauf zu reagieren.

Die Analyse der Zwei-gegen-Zwei-Spiele zeigt, dass eine Berücksichtigung von Hindernissen entscheidende Vorteile für die Weiterentwicklung des Spiels ergeben würde. Zunächst könnte eine Kollisionsvermeidung zum Einsatz kommen, um blockierenden Spielern auszuweichen. Dies wird zunehmend wichtiger, da in den nächsten Jahren mit einer Regelverschärfung und damit einhergehenden härteren Ahndung von Kollisionen (Fouls) gerechnet werden muss. Die Einbeziehung des Gegners könnte aber auch zu einem besseren Stellungsspiel führen und präzisere Torschüsse erlauben, die Erkennung von eigenen Spielern ein genaues Passspiel ermöglichen. Neben den Zwei-gegen-Zwei-Spielen gibt es noch weitere Aufgabenstellungen (z.B. Dribbling im Slalom), bei denen die genannten Möglichkeiten sinnvoll eingesetzt werden könnten.

Mit der in Zukunft sicherlich wachsenden Anzahl von Spielern eines Teams werden diese Aspekte bei den regulären Spielen in der Humanoidliga zunehmend an

Bedeutung gewinnen, wie man bei den Vier-gegen-Vier-Spielen der *Sony Four-Legged Robot League*[1] vergleichend feststellen kann. Die Vierbeinerliga verwendet entsprechend ihrer Regeln identische Robotermodelle und ist diesbezüglich als homogen zu bezeichnen. Dort lässt sich auch beobachten, dass es möglich ist auf Hindernisse zu reagieren. Zur Unterscheidung von Gegnern und eigenen Robotern gibt es bereits Arbeiten, auf die später noch eingegangen wird [2]. Die Konzepte der Vierbeinerliga sind aufgrund der Verwendung unterschiedlicher Roboter innerhalb der Humanoidliga (Heterogenität) allerdings nur bedingt übertragbar.

Es gilt also, die bestehende Robotersoftware *RoboCup06App* entsprechend zu erweitern. In einem ersten Schritt muss dazu eine Hinderniserkennung entwickelt werden. Die Umgebungsbedingungen, auf welche an späterer Stelle detailliert eingegangen wird, legen eine Aufteilung in eine allgemeine Hindernis- und eine besondere Spielererkennung nahe. Dem folgt ein Modellierungsteil, in dem die neu gewonnenen Informationen zusammengetragen und aufgearbeitet werden, so dass eine Berücksichtigung auf Verhaltensebene des Roboters stattfinden kann.

Durch den Entwurf eines Roboterverhaltens soll die Verwendung der neuen Erkenntnisse demonstriert werden.

[1] Sony Four-Legged Robot League `http://www.tzi.de/4legged`

3 Grundlagen

In der Einleitung ist bereits angeklungen, dass dieses junge Fachgebiet stark durch die Fußballwettkämpfe des RoboCups geprägt ist. Eine kurze Beschreibung dieser spielerischen Rahmenbedingungen soll einen Einblick in das Umfeld dieser Arbeit bieten. Die anschließende Erläuterung der Umgebungsbedingungen (z.B. Roboter, Spielfeld, Aufgaben) verdeutlicht die technischen Gegebenheiten, welche bei der Entwicklung der Software zu beachten sind. Es folgt die Beschreibung der Software, die bei der Entwicklung der Roboter an der Technischen Universität Darmstadt zum Einsatz kommt. Diese Diplomarbeit beschreibt die Entwicklung einer Erweiterung der bestehenden Steuerungssoftware. Daher kommt der Darstellung dieser Software eine besondere Bedeutung zu.

3.1 RoboCup

Der RoboCup ist ein Kooperationsprojekt von Universitäten und Forschungseinrichtungen, welches die Forschung und Bildung im Bereich der künstlichen Intelligenz, der Robotik und verwandten Disziplinen fördern möchte. Seit 1997 finden jährlich nationale und internationale Konferenzen und Wettbewerbe statt, bei denen sich Roboter und Softwareagenten messen. Dabei gibt es vier Bereiche, die sich durch unterschiedliche Themengebiete differenzieren. Der große Bereich *RoboCupSoccer* hat das Thema Fußball als Aufgabengebiet, weil dieser Aspekt genügend komplexe Anforderungen stellt und ein großes Maß an öffentlichem Interesse hervorruft. Zur speziellen Förderung von Jugendlichen gibt es den *RoboCupJunior*, der mit einfacheren Robotersystemen arbeitet und überschaubarere Aufgabenstellungen beinhaltet. *RoboCupRescue* untersucht den Einsatz von Robotern in Katastrophengebieten und *RoboCup@Home* beschäftigt sich mit neuen Anwendungen in Privathaushalten.

RoboCupSoccer gliedert sich weiter in fünf verschiedene Ligen (*Simulation League, Small-size League, Middle-size League, Four-Legged League, Humanoid League*), die sich durch ihre Rahmenbedingungen bezüglich der Feldabmessungen, Robotergröße und Antriebsart unterscheiden. Innerhalb dieser Ligen werden mehrere Wettbewerbe mit zum Teil unterschiedlichen Aufgabenstellungen ausgetragen. Als

plakatives Ziel des RoboCups wurde ausgegeben, bis 2050 eine humanoide Robotermannschaft zu entwickeln, die gegen den amtierenden Weltmeister des FIFA Worldcups gewinnen kann.

Die Technische Universität Darmstadt wird seit 2001 durch die Darmstadt Dribbling Dackles[1] in der *Sony Four-Legged Robot League* und seit 2003 durch die Darmstadt Dribblers[2] in der Humanoidliga vertreten. Nicht nur in der Gründungsphase konnte das Team der Zweibeiner von den Erfahrungen der Darmstadt Dribbling Dackles profitieren. Beide Teams können auf eine erfolgreiche Bilanz bei den vergangenen Veranstaltungen zurückgreifen.

3.2 Umgebungsbedingungen

Dieser Abschnitt fasst die wichtigsten Elemente der Umgebungsbedingungen zusammen. Die Regeln der Humanoidliga[3] gehen detaillierter auf die einzelnen Gesichtspunkte ein.

Die Umgebungsbedingungen sind von besonderer Wichtigkeit, da diese Vorgaben das einzige a priori Wissen überhaupt darstellen. Ohne die Kenntnis über die Größenverhältnisse und Anordnung des Spielfeldes wäre eine Orientierung im Raum nur sehr schwer möglich.

Farbe	**Verwendung**
gelb, blau	Tore, Segmente der Landmarken
orange	Ball
grün	Spielfläche
weiß	Feldlinien, Torpfosten
magenta, cyan	Teammarker der Roboter

Tabelle 3.1: Bedeutung der Farben des Spielfeldes

Die definierten Farben der Spielfeldelemente haben eine spezielle Bedeutung (Tabelle 3.1). Sie ermöglichen eine vereinfachte Erkennung, indem das Vorwissen um die Farben geschickt eingesetzt wird. Durch sie wird die Verwendung von weniger komplexen Algorithmen zur Objekterkennung möglich. Aufgrund der oft sehr stark eingeschränkten Rechenkapazität der Robotersteuerungen können meist keine aufwändigeren Verfahren eingesetzt werden. Als weiteres Problem stellt sich die

[1] Darmstadt Dribbling Dackles `http://robocup.informatik.tu-darmstadt.de`
[2] Darmstadt Dribblers `http://www.dribblers.de`
[3] Humanoid League Rules `http://www.humanoidsoccer.org/rules.html`

Störanfälligkeit gegenüber Helligkeitsveränderungen dar. Für die Erkennung über Farben muss die Beleuchtung möglichst konstant gehalten werden.

3.2.1 Roboter

Roboterkoordinaten Der Ursprung dieses dreidimensionale Koordinatensystem liegt im definierten Fußpunkt des Roboters (Lot des Schwerpunktes in Nullposition). Es handelt sich um ein rechtsgerichtetes System, bei dem die x-Achse nach vorne, die y-Achse nach links und die z-Achse nach oben zeigt.

In der Humanoidliga finden autonome, menschenähnliche (humanoide) Roboter Verwendung. Charakteristisch ist die Fortbewegung auf zwei Beinen. Autonomie bedeutet in diesem Zusammenhang, dass die Roboter selbstständig handeln können. Sie verfügen über eine unabhängige Stromversorgung und Recheneinheit, die sie mit sich führen; eine Steuerung von außen, etwa mittels einer Fernbedienung, ist untersagt. Es ist den Robotern eines Teams gestattet über WLAN miteinander zu kommunizieren. Die ausgetauschten Datenpakete (*Teammessages*) enthalten z.B. Informationen über die Ballposition oder die eigene Position auf dem Spielfeld.

3.2.1.1 Allgemeine Beschaffenheit

In der Humanoidliga werden zwei Größenklassen (KidSize Liga, TeenSize Liga) unterschieden. Trotz der grundsätzlichen Übertragbarkeit der hier entwickelten Konzepte, wird die Behandlung im Folgenden auf die KidSize Liga beschränkt, um Verwirrungen mit mehrdeutigen Größenangaben zu vermeiden.

Im Gegensatz zu anderen Ligen des RoboCups besitzen die Roboter der *Humanoid League* sehr unterschiedliche Bauformen. Bei der *Sony Four-Legged Robot League* gehen die Regeln sogar soweit, dass alle Teams ein und dasselbe kommerzielle Robotermodell benutzen müssen, wobei Modifikationen an der Hardware nicht gestattet sind. Da die Liga der Humanoiden erst seit wenigen Jahren besteht und es keine einheitliche Plattform gibt, auf der die Teams arbeiten können, werden überwiegend modifizierte Bausätze oder Eigenbauten verwendet.

Die Regeln der Humanoidliga schränken die Ausgestaltung der Hardware nur grob ein. Die Größe eines KidSize-Roboters muss zwischen 30 und 60 cm liegen. Es gibt weitere Restriktionen, z.B. für die Größe der Füße oder die räumliche

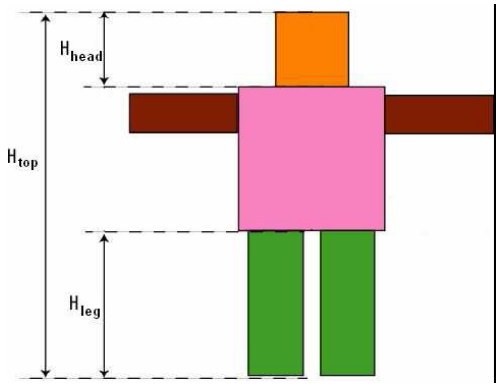

Abbildung 3.1: Schematischer Aufbau eines Humanoidroboters

Ausdehnung, welche in Abhängigkeit zur Roboterhöhe definiert sind (Abb. 3.1[4]).

Die Konstrukteure sind dazu angehalten, sich möglichst nahe am menschlichen Vorbild zu bewegen. Die Verwendung von aktiven Sensoren, die elektromagnetische Wellen zur Abstandsbestimmung aussenden und empfangen (z.B. Radar, aktives Sonar, Laserscanner), ist untersagt. Daher ist der Einsatz eines Kamerasystems als Hauptsensor die einzig sinnvolle Lösung, um auf die Umwelt reagieren zu können. Dieses sollte sich möglichst am Kopf des Roboters befinden, eine Montage an den Extremitäten ist sogar verboten. Zusätzlich erlaubt sind interne Sensoren zur Lagebestimmung und Beschleunigungsmessung sowie zur Messung der Gelenkwinkelpositionen.

Die Roboter müssen zu 90% schwarz sein und auf Anfrage des gegnerischen Teams farbliche Markierungen (cyan oder magenta) auf Brusthöhe tragen. Die 8x8 cm großen Markierungen sollten nach Möglichkeit von allen Seiten aus sichtbar sein.

3.2.1.2 Robotermodelle des Fachgebiets

Das Fachgebiet Simulation und Systemoptimierung hat verschiedene Robotermodelle im Rahmen des RoboCups eingesetzt. Abbildung 3.2 zeigt links den durch die Darmstadt Dribblers bei den GermanOpen 2005 verwendeten modifizierten KHR-1 Bausatz[5] [4, 8, 9]. Bei der Weltmeisterschaft 2005 in Osaka, Japan kam das Nachfolgemodell YDH zum Einsatz (mittleres Bild).

Das aktuelle Robotermodell HR 18 (rechts) ist in einer Kooperation mit dem

[4]Bildquelle: Humanoid League Rules `http://www.humanoidsoccer.org/rules.html`
[5]www.kondo-robot.com `KondoKagakuCo.,Ltd.`

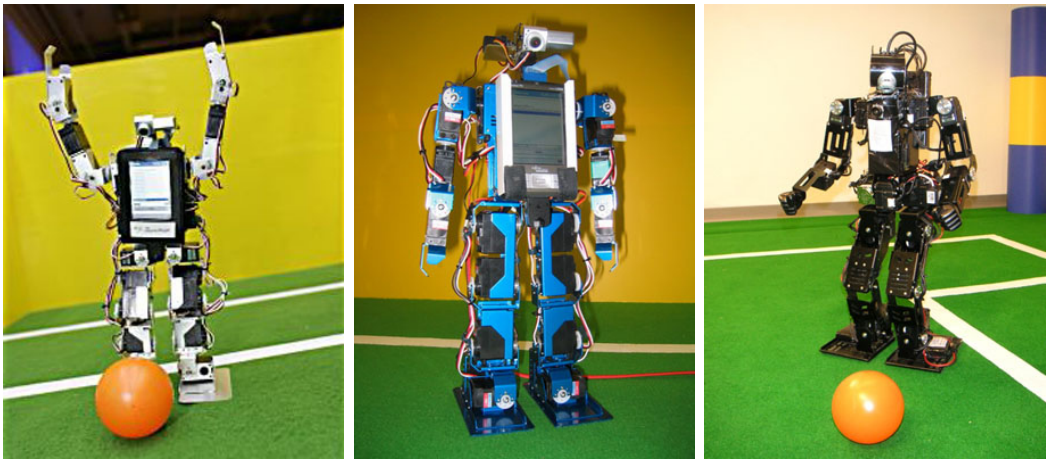

Abbildung 3.2: Robotermodelle des Fachgebiets

japanischen Roboterhersteller Hajime Sakamoto[6] entwickelt worden. Es besitzt insgesamt 21 Gelenke, die auch als Freiheitsgrade bezeichnet werden. Der Roboter hat eine Höhe von ca. 60 cm und verfügt über eine bewegliche Kamera im Kopf, eine Weitwinkelkamera auf Brusthöhe, sowie Beschleunigungs- und Lagesensoren im Oberkörper.

Aufgrund des geringen Gewichts dient ein handelsüblicher Pocket PC (PPC) mit einem Windows CE Betriebssystem als Steuerungseinheit. Daraus resultieren die erheblichen Ressourceneinschränkungen. Besonders ins Gewicht fallen dabei der langsamen Arbeitsspeicher und der fehlende mathematische Koprozessor zur Berechnung von Gleitkommazahlen (FPU[7]).

Eine effiziente Datenverarbeitung muss daher ein integrales Ziel der Programmierung sein. Dies gilt in verstärktem Maße für die Bildverarbeitung, da diese die größte Datenmenge zu bewältigen hat. Auf die Funktionsweise der Bildverarbeitung wird im Abschnitt 3.3.1.1 näher eingegangen.

[6]Hajime Research Institute, Ltd. `http://www.hajimerobot.co.jp`
[7]engl. Floating Point (Processing) Unit

3.2.2 Spielfeld

Spielfeldkoordinaten Das Spielfeld wird durch ein dreidimensionales kartesisches Koordinatensystem beschrieben, dessen Ursprung sich im Zentrum des Mittelkreises befindet. Dabei zeigt die x-Achse in Richtung des gegnerischen Tors; die y-Achse verläuft nach links und die z-Achse nach oben.

Das Spielfeld besteht aus einem rechteckigen Feld mit zwei Toren, vier Landmarken, Feldlinien und neun Aufsetzmarken (vgl. Abbildung 3.3[8]). Als Untergrund findet ein handelsüblicher grüner Teppich Verwendung. Die Feldlinien haben eine Dicke von 5 cm und bestehen in der Regel aus weißem Klebeband.

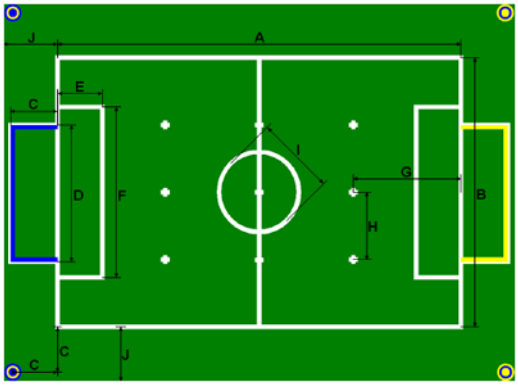

Abbildung 3.3: Spielfeldschema der Humanoidliga

Die Tore haben eine Höhe von 90 cm und sind an den drei Innenflächen blau respektive gelb markiert. Sie besitzen eine weiße Querlatte. Die Landmarken befinden sich außerhalb des eigentlichen Spielfeldes und sollen den Robotern bei der Orientierung auf dem Spielfeld helfen. Als Landmarken dienen runde Säulen mit einem Durchmesser von 20 cm und bestehen aus jeweils drei 30 cm hohen übereinander liegenden farblich markierten Segmenten. Auf der Seite des gelben Tors ist die Farbreihenfolge gelb-blau-gelb, auf der Seite des blauen blau-gelb-blau. Die Abmessungen des Spielfeldes sind detailliert Tabelle 3.2[9] zu entnehmen.

[8] Bildquelle: Humanoid League Rules `http://www.humanoidsoccer.org/rules.html`
[9] Quelle: Humanoid League Rules `http://www.humanoidsoccer.org/rules.html`

Strecke	Bezeichnung	Ausmaß
A	Feldlänge	450 cm
B	Feldbreite	300 cm
C	Tortiefe	50 cm
D	Torbreite	150 cm
E	Strafraumtiefe	50 cm
F	Strafraumbreite	190 cm
G	Strafstossdistanz	120 cm
H	Abstand der Aufsetzmarken	75 cm
I	Durchmesser des Mittelkreises	90 cm
J	Randbreite (mindestens)	60 cm

Tabelle 3.2: Spielfeldmaße

3.2.3 Aufgaben

In drei Wettbewerben können die Roboter der KidSize Liga antreten: ein Zwei-gegen-Zwei-Spiel, eine Strafstoß-Disziplin sowie eine „Technical Challenge“ mit drei verschiedenen Aufgaben. Zum einen gilt es eine unebene Fläche in möglichst kurzer Zeit zu überwinden. Eine weitere Problemstellung ist dabei das Dribbeln eines Balls im Slalom um drei Hindernissäulen. Bei der dritten Aufgabe sollen mehrere Pässe zwischen zwei Robotern ausgeführt werden, ohne dass diese sich zu nahe kommen.

Die Aufgaben können sich von Jahr zu Jahr ändern. Durch eine anspruchsvollere Gestaltung wird die Forschung zusätzlich vorangetrieben.

3.3 Software

Bei der Entwicklung der humanoiden Roboter werden unterschiedliche Softwarelösungen eingesetzt und weiterentwickelt. Den Hauptteil bilden dabei die eigentliche Steuerungssoftware des Roboters (*RoboCup06App*), sowie eine zugehörige Benutzeroberfläche (*RoboCup06GUI*). Beide Programme basieren auf *RoboFrame*, einem Framework für Robotersteuerungen, welches am Fachgebiet Simulation und Systemoptimierung im Rahmen einer Diplomarbeit entwickelt wurde [13].

Ein Framework ist ein Programmgerüst, welches eine Anwendungsarchitektur für eine festgelegte Domäne vorgibt. Erst durch die Implementierung und Registrierung von konkreten Klassen entsteht ein funktionstüchtiges Programm, so dass eine Ausgestaltung im Einzelfall erforderlich ist. Dabei werden Kontrollfluss (Auf-

ruf registrierter Softwarekomponenten) und Schnittstellen durch das Framework definiert.

RoboFrame stellt eine plattformunabhängige Rahmensoftware zur Steuerung von mobilen autonomen Robotern dar. Es regelt beispielsweise den Datenaustausch zwischen verschiedenen Softwarekomponenten und deren zeitliche Steuerung durch die Zuordnung zu Prozessen.

3.3.1 RoboCup06App

Bei *RoboCup06App* handelt es sich um die eigentliche Steuerungssoftware des Roboters. Sie wird in der Regel auf der Recheneinheit eines Roboters eingesetzt, kann aber zu Debugzwecken auch auf einem Windows PC laufen. Sie enthält bereits alle Komponenten, die den Roboter zum Fußballspielen befähigen. Der Aufbau der vorhandenen Software orientiert sich am Prinzip des „*Sense - Plan - Act*"- Zyklus. Dies bedeutet, dass eine logische Aufteilung der Softwarekomponenten in die drei Bereiche Wahrnehmung, Planung und Aktorik vorgenommen wird [5, 14].

Die im Rahmen dieser Arbeit entwickelte Software ist als Integration in *RoboCup06App* zu sehen und soll die bestehende Steuerung erweitern. In den folgenden Abschnitten werden daher bereits bestehende Komponenten der Steuerungssoftware beschrieben, auf welche bei den Neuerungen zurückgegriffen wird. Dazu gehört insbesondere die Bildverarbeitung, welche die Daten des wichtigsten Sensors, der Kamera, verarbeitet.

3.3.1.1 Bildverarbeitung

Bildkoordinaten Der Ursprung dieses zweidimensionalen kartesischen Koordinatensystems liegt in der oberen linken Ecke des Bildes. Die x-Achse zeigt nach rechts und die y-Achse nach unten.

Als Grundlage für die hier verwendete Bildverarbeitung dienen die farblichen Markierungen (vgl. 3.2). Die eindeutige Definition der Farben erleichtert die Erkennung.

Gegenüber dem menschlichen Auge ist die visuelle Sensorik eines Roboters eingeschränkt. Menschen können beispielsweise einen mit der Kamera aufgenommenen orangefarbenen Ball leicht erkennen. Für den Roboter ist das keine triviale Aufgabe. Bei genauerer Betrachtung setzt sich der Ball im Bild aus mehreren Bildpunkten unterschiedlicher Farbtöne zusammen. Dies wird besonders durch die vorherrschenden Lichtverhältnisse und die sich daraus ergebenden Helligkeitsschwan-

kungen und Unterschiede der Farbtemperaturen beeinflusst. Menschen können die verschiedenen Farbtöne und Helligkeitswerte kompensieren und erkennen so nicht nur eine orangefarbene Fläche, sondern den Ball als dreidimensionales Objekt.

Die Bildverarbeitung der Robotersoftware arbeitet in mehreren Schritten. Zunächst werden die verschiedenfarbigen Pixel eines Farbtons zu einer Farbklasse zusammengefasst. Alle Pixel, die keiner der sieben definierten Farben entsprechenden Klasse zugewiesen werden konnten, fallen in die zusätzliche Klasse „unknown", die schwarz dargestellt wird. Das Bild wird somit auf acht Farben beschränkt, wodurch der Speicherbedarf enorm abnimmt aber auch Informationen verloren gehen.

Anschließend sollen aus diesem „reduzierten" Bild Merkmale extrahiert werden, die verschiedene Teile des Spielfeldes beschreiben. Dabei kann nicht von einer echten Objekterkennung gesprochen werden, weil keine dreidimensionalen Gegenstände erfasst werden, sondern lediglich Merkmale, die diese rudimentär beschreiben. Beispielsweise wird ein erkanntes Tor nur durch eine Linie auf der Spielfläche beschrieben. Die Umrechnung von Bildkoordinaten in Roboterkoordinaten wird im letzten Abschnitt des Bildverarbeitungsteils behandelt.

Bildsegmentierung Die Zuordnung von Bildpunkten zu Klassen wird Segmentierung genannt. Die Literatur nimmt eine Unterscheidung in pixel-, kanten- und regionenorientierte Verfahren vor [7]. Pixelorientierte Verfahren treffen für jeden Bildpunkt die Entscheidung zu welchem Segment er gehört. Bei kantenorientierten Ansätzen wird im Bild nach Objektübergängen gesucht während regionenorientierte Vorgehensweisen Punktmengen betrachten. Die beiden letztgenannten Verfahren gehören dabei zur Kategorie der im Abschnitt 3.2 als aufwändig bezeichneten Algorithmen und können mit der aktuellen Steuerungshardware nicht eingesetzt werden. Durch die Ressourcenbeschränkung wird daher ein pixelorientierter Ansatz verwendet.

Die Zugehörigkeit eines Pixels zu einem Segment wird mit Hilfe einer so genannten Farbtabelle bestimmt. Da Farbwerte digitale Bilder in der Regel durch einen dreidimensionalen Farbraum (RGB, YUV, ...) dargestellt werden, kann man sich die Tabelle als dreidimensionalen Würfel vorstellen. Dabei beschreibt jede Dimension eine der drei Farbkomponenten. In der Tabelle ist für jede mögliche Kombination eine Farbklasse angegeben, so dass jeder Pixel aufgrund seiner Farbwerte genau einer Klasse zugeordnet werden kann. Das Erstellen der Farbtabelle ist aufwändig und muss manuell erfolgen. Hierbei hilft der *ColortableDialog*, der zur *RoboCup06GUI* gehört (vgl. 3.3.2). Für jeden neuen Einsatzort der Roboter oder bei sich verändernden Lichtverhältnissen muss eine neue Tabelle angelegt,

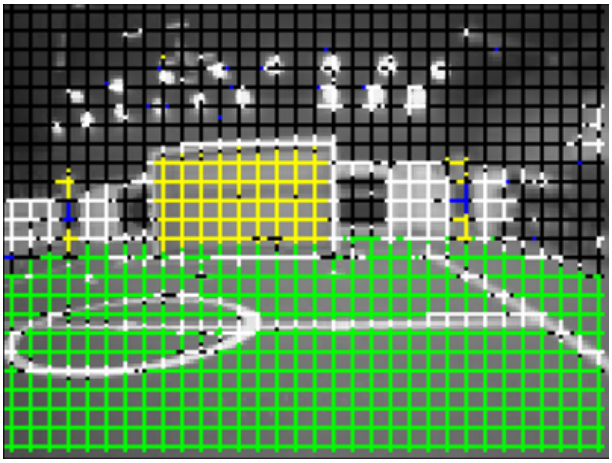

Abbildung 3.4: Bildsegmentierung mit *Scanlineraster*

bzw. eine vorhandene angepasst werden. Zwei zusätzliche Maßnahmen sollen die Ressourcen der Recheneinheit weiter entlasten. Zum einen werden pro Farbkanal immer vier benachbarte Farbtöne zusammengefasst, so dass die Farbtabelle wesentlich weniger Speicherplatz belegt und zum anderen wird nicht jeder Pixel eines Kamerabildes klassifiziert. Stattdessen durchlaufen nur diejenigen Pixel die Segmentierung, die sich auf einem Raster aus so genannten *Scanlines* befinden (siehe Abbildung 3.4). Somit verkleinert sich die Menge der weiter zu betrachtenden Bildpunkte und die Anzahl der Zugriffe auf die Farbtabelle.

Merkmalsextraktion Damit die Merkmalsextraktion performant abläuft, findet sie nach der Segmentierung statt. Dadurch arbeitet sie außerdem unabhängig vom ursprünglichen Farbraum der Kamera (YUV) und ist somit generisch einsetzbar.

Verschiedene parallel arbeitende Softwareteile versuchen anhand der segmentierten *Scanlines* Rückschlüsse auf Elemente des Spielfeldes zu ziehen. Diese als *Perceptoren* bezeichneten Einheiten sind jeweils für einen Elementtyp zuständig. So gibt es beispielsweise einen solchen Wahrnehmungsspezialisten zur Erkennung des Balls, der Tore und der Feldlinien.

Die Ballerkennung wird durch den *BallPerceptor* durchgeführt. Anhand des *BallPerceptors* soll im Folgenden beispielhaft die generelle Funktionsweise eines *Perceptors* verdeutlicht werden. Dieser arbeitet mit den segmentierten Pixeln, die durch die *Scanlines* erfasst wurden. Von diesen Pixeln werden die orangefarbenen herausgefiltert und auf Anhäufungen innerhalb des Koordinatensystems untersucht.

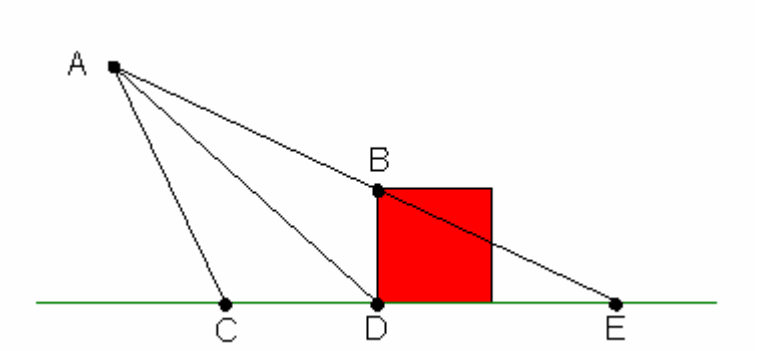

Abbildung 3.5: Projektion der Bildpunkte

Nach der Identifizierung der größten Anhäufung, wird diese anhand einer möglichst exakt platzierten Kreisbahn beschrieben, die durch den Mittelpunkt und der Radius bestimmt ist. Diese auf ein Minimum reduzierte Information wird in diesem Fall als *BallPercept* zusammengefasst und anderen Softwareteilen zur Verfügung gestellt. Die anderen Erkennereinheiten arbeiten analog und erzeugen entsprechend ähnliche *Percepte*.

Umrechnung der Bildkoordinaten Ein besonderes Problem stellt die Umrechnung von Bildkoordinaten in Roboterkoordinaten dar. Da das Bild nur eine zweidimensionale Abbildung der Umwelt darstellt, geht die Information der dritten Dimension verloren. Es ist dem Roboter nicht möglich Tiefen- und Höheninformationen der Umwelt aus dem Bild zu unterscheiden. Durch einen Sichtstrahl wird jeder Bildpunkt auf einen Punkt in Roboterkoordinaten projiziert. Nur wenn es sich um einen Fußpunkt eines Objektes im Bild handelt, kann die Position bezüglich des Roboters bestimmt werden. Abbildung 3.5 soll den Sachverhalt verdeutlichen.

Das rote Quadrat stellt ein Hindernis auf dem grünen Boden dar. Punkt A bezeichnet den Augpunkt des Roboters, von dem die schwarzen Sichtstrahlen ausgehen. Der linke und rechte Sichtstrahl entsprechen dabei dem unteren bzw. oberen Rand des Kamerabildes. Vereinfacht kann davon ausgegangen werden, dass eine vertikale Bildzeile betrachtet wird, in der von unten zunächst grün zu sehen ist (Strecke CD). Danach folgt rot (Strecke DB) bis zum oberen Bildrand (Punkt B). Bei der Berechnung der Punkte relativ zum Roboter projiziert die Bildverarbeitung die Bildzeile auf den Boden. Da es sich bei den Punkten C und D im Bild um Bodenpunkte handelt, werden diese wie erwünscht umgerechnet - quasi auf sich selbst projiziert. Die Abbildung von Punkt B hingegen ergibt Punkt E, der

viel zu weit vom Roboter entfernt liegt. Denn bei Punkt B handelt es sich nicht um einen Bodenpunkt. Durch das zweidimensionale Kamerabild fällt seine Höhenangabe mit der Entfernung zusammen. Diese Information ist verloren gegangen. Das Lot von B auf den Boden kann ohne die Kenntnis der Höhe des Hindernisses nicht gefällt werden. Noch problematischer sind Hindernisse, die nicht orthogonal auf dem Boden stehen. Punkt D kann nicht berechnet werden. Möchte man die Position eines Objektes auf dem Spielfeld bestimmen, so muss man mindestens einen Fußpunkt identifizieren (Punkt D in der Bildzeile finden). Diese Fußpunkte lassen sich dann richtig in das Roboterkoordinatensystem umrechnen, wie anhand Punkt C und D in der Grafik gezeigt.

3.3.1.2 Odometrie

Odometrie oder auch Hodometrie (von griech. hodós, „Weg" und métron, „Maß" -also: Wegmessung) ist die Wissenschaft von der Positionsbestimmung eines Fahrzeuges durch die Beobachtung seiner Räder. Die Odometrie ist ein grundlegendes Navigationsverfahren für bodengebundene Fahrzeuge aller Art (Kfz, Roboter), allerdings wird es aufgrund seiner Fehlereigenschaften selten als alleiniges Verfahren eingesetzt[10].

Der Roboter hat also nur Kenntnis darüber, welche Strecke er bei einem seiner Basisschritte zurücklegt und welchen Winkel er dabei beschreibt. Zu beachten ist hierbei, dass der Weg nur relativ zum Startpunkt angegeben werden kann. Es kann keine Aussage darüber getroffen werden, wo sich der Roboter auf dem Spielfeld befindet. Die Kenntnis der Odometriedaten eines Basisschritts beziehen sich auf den Optimalfall und Abweichungen durch Reibung und Schlupf kann dabei nicht Rechnung getragen werden. Stößt der Roboter beispielsweise während des Laufens gegen ein Hindernis, so wird der zurückgelegte Weg aus den Gehbewegungen des Roboters ermittelt, auch wenn er sich dabei gar nicht von der Stelle bewegt. Die Odometrie hat keine Möglichkeit auf Einflüsse der Umwelt zu reagieren. Für eine zurückgelegte Strecke zählt der Roboter die Informationen der Basisschritte zusammen. Es wird schnell klar, dass durch (kleine) Fehler bei einzelnen Schritten die Abweichung bei vielen Schritten erheblich sein kann.

3.3.1.3 Selbstlokalisation

Die Selbstlokalisation soll möglichst genau die Position und Ausrichtung (POSE) des Roboters bestimmen. Die Kenntnis über die Position und Ausrichtung des Roboters ist dazu notwendig, relativ zum Roboter wahrgenommene Elemente in

[10] Odometrie, Quelle: Wikipedia `www.wikipedia.de`

Spielfeldkoordinaten umzurechnen. Die gewonnenen Erkenntnisse können so zum Beispiel darauf verwendet werden, wahrgenommene Hindernisse auf einer Karte des Spielfeldes anzeigen zu lassen oder einem Mitspieler Positionen von erkannten Elementen mitzuteilen.

Zur Bestimmung der eigenen POSE wird das Vorwissen über die Umgebungsbedingungen genutzt. Dazu werden die aus der Wahrnehmung gewonnenen Informationen mit den tatsächlichen Spielfeldmaßen abgeglichen. Neben den Landmarken und Toren lassen sich besonders die Feldlinien zur Lokalisierung heranziehen.

3.3.1.4 Weltmodell

Das Weltmodell bildet die Schnittstelle zur Verhaltensebene. Es ist dafür zuständig, die aus der Wahrnehmung gewonnenen Daten aufzubereiten und kontinuierlich zur Verfügung zu stellen. Es speichert unter anderem die Position, an der der Ball zuletzt gesehen wurde. Würde man im Verhalten direkt mit den *Percepten* der Wahrnehmung arbeiten, so könnten immer nur die Informationen aus dem aktuellen Bild verwendet werden. Zusätzlich wird eine Glättung der ermittelten Positionen vorgenommen, da die *Percepte* in ihren Angaben oftmals springen. Dies ist durch ein Schwanken des Roboters bei Laufbewegungen zu erklären, die bisher nicht zu kompensieren sind.

3.3.2 RoboCup06GUI

Bei diesem dialogbasierten Programm handelt es sich um die Zusammenfassung verschiedenster Werkzeuge, die bei der Weiterentwicklung der Steuerungssoftware nützlich sind. Das Programm kommt auf einem Windows PC zum Einsatz und arbeitet in der Regel über eine TCP/IP Verbindung mit einer Instanz der Steuerungssoftware *RoboCup06App* zusammen. Die meisten der Tools empfangen lediglich Daten der Steuerungssoftware und dienen deren Anzeige. Sie werden im folgenden Abschnitt näher erläutert (3.3.2.1). Andere Dialoge dienen dazu Einstellungen an der laufenden Steuerungssoftware vorzunehmen oder Bewegungen und Verhalten ablaufen zu lassen. Diese sendenden Komponenten werden im Anschluss erläutert (3.3.2.2). Eine Sonderstellung nimmt der *LogRecorder* ein, da er Daten senden und empfangen kann. Seine Funktion wird weiter unten beschrieben (3.3.2.3).

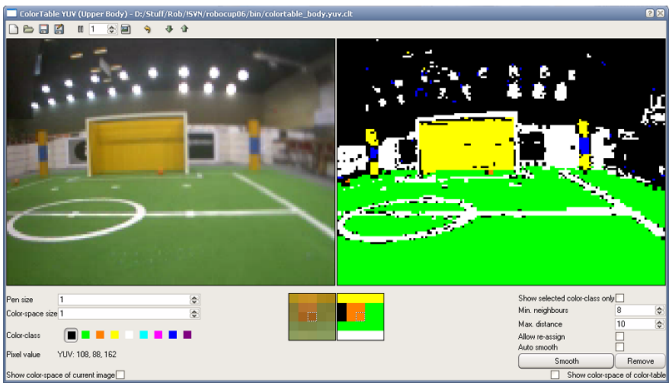

Abbildung 3.6: Dialog zum Erstellen einer Farbtabelle

3.3.2.1 Anzeigedialoge

Die Anzeigedialoge haben die Aufgabe, vom Roboter zur Verfügung gestellte Daten zu visualisieren. Hierzu gehört der *ImageViewer*, mit dem man die Kamerabilder des Roboters betrachten kann. Zusätzlich lassen sich die vom Roboter gelieferten *Percepte* im Bild einzeichnen. So besteht die einfache Möglichkeit die Wahrnehmungsobjekte auf Genauigkeit zu überprüfen.

Weiterhin stehen eine egozentrische und eine spielfeldorientierte Modelldatenanzeige zur Verfügung, in die zum Beispiel die Ballposition relativ zum Roboter bzw. absolut auf einem Abbild des Spielfeldes eingezeichnet werden kann. Es gibt noch eine Reihe weitere Dialoge, welche Informationen über den aktuellen Roboterstatus numerisch darstellen.

Der Dialog zum Erstellen der Farbtabelle (Abb. 3.6) soll hier noch gesondert erwähnt werden. Neben der reinen Anzeigefunktion vereinfacht dieser Dialog das Erstellen einer Farbtabelle von Hand; eine automatische Generierung ist bisher nicht möglich. Auf der linken Seite ist ein vom Roboter aufgenommenes Bild der Bauchkamera zu sehen. Durch einen Mausklick auf einen Pixel im linken Bild, wird an der entsprechenden Stelle in der Farbtabelle, die unten links im Dialog ausgewählte Farbklasse eingetragen. Das rechte Bild zeigt immer die Segmentierung mit der aktuellen Farbtabelle. Im Gegensatz zur Robotersoftware, die nur die Pixel der *Scanlines* betrachtet, nimmt der Dialog eine vollständige Segmentierung vor. *RoboCup06GUI* läuft auf einem normalen PC und daher stehen ausreichend Ressourcen zur Verfügung.

Das Erstellen der Tabelle ist sehr zeitaufwändig, da möglichst viele Bilder, die das Spielfeld samt Ball aus verschiedenen Perspektiven zeigen, in die Betrachtung mit einbezogen werden müssen. Hinzu kommt, dass für jeden neuen Ort, an dem

die Farberkennung eingesetzt werden soll, eine neue Farbtabelle erstellt werden muss.

3.3.2.2 Dialoge mit Sendefunktion

Die Werkzeuge dieser Kategorie haben verschiedene Aufgaben. Mit ihnen können Konfigurationsdaten zur Robotersoftware übermittelt, Bewegungen gestartet oder ein Verhalten ausgewählt werden.

3.3.2.3 LogRecorder

Der *LogRecorder* ist in der Lage, entweder Daten der Steuerungssoftware zu empfangen oder an diese zu senden. Im ersten Fall kann eine Menge sämtlicher von der Steuerungssoftware zur Verfügung gestellten Datentypen bestimmt werden, die dann aufgezeichnet und auch in einer Logdatei gespeichert werden kann. Im zweiten Fall kann man die aufgezeichneten Daten an die Robotersoftware schicken. Beispielsweise ist es möglich Kamerabilder einer Logdatei wie einen Film abzuspielen und dabei an die Steuerungssoftware auf einen Windows PC zu schicken. Speziell während der Entwicklung der Erkenner lassen sich so Veränderungen der *Percepte* anhand der gleichen Eingabedaten beobachten. Abgesehen davon ist man nicht unbedingt auf einen realen Roboter mit Kameras angewiesen.

3.3.3 Simulator

Da die Anzahl der Roboter am Fachgebiet begrenzt ist, sowie Tests an realen Robotern zeitintensiv sind, aber auch um das Material zu schonen, wurde ein Simulator entwickelt. Die Umgebung des Simulators besteht aus einer maßstabsgetreuen Abbildung des Spielfeldes samt Toren und Feldlinien (vgl. 3.2.2) und einem simulierten Roboter (Abb. 3.7).

Es gibt weiterhin einen statischen Hindernisroboter und einen Ball, welche von Hand positioniert werden können. Interaktionen, wie etwa das Treten des Balls, sind derzeit noch nicht möglich.

Der simulierte Roboter ist über die gleiche Schnittstelle mit der Robotersteuerungssoftware verbunden wie ein realer Roboter. Konsequenterweise empfängt der Roboter nicht nur Befehle für seine Bewegungen, sondern übermittelt auch Sensordaten an seine Steuerungssoftware (bidirektionale Kommunikation). Der Simulator beschränkt sich bei den Sensordaten auf die Generierung perspektivischer Kamerabilder, die die Bilder der realen Kameras ersetzen.

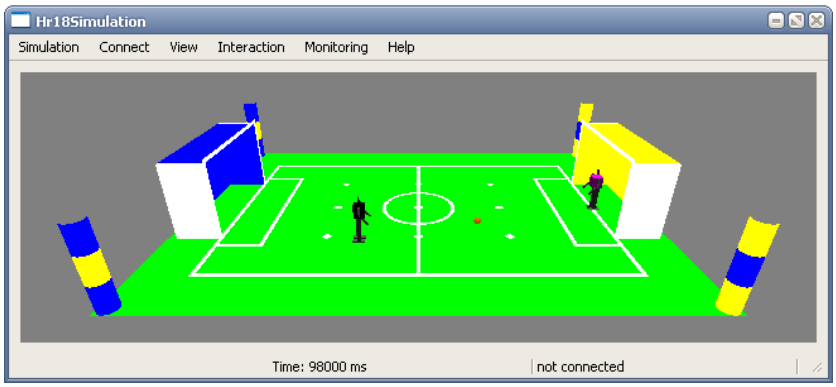

Abbildung 3.7: Vollständige Spielfeldansicht des Simulators

Der Simulator ist ein wirkungsvolles Werkzeug, wenn es darum geht, Entwicklungszeit einzusparen. Abschließende Tests am realen Roboter kann er aber nicht ersetzen. Dies ist durch sein stark vereinfachtes Abbild der Umwelt zu erklären. Zum einen bewegt sich der simulierte Roboter ideal, da z.B. keine Toleranzen in den Gelenken oder Schlupf an den Füßen existieren. Auf der anderen Seite beinhalten die generierten Kamerabilder nicht die Störeinflüsse einer realen Umwelt, wie Bildunschärfen, Verzerrungen oder Farbschwankungen durch verschiedene Lichteinflüsse. Vergleicht man die Kamerabilder des Simulators (Abbildung 3.8) mit denen des realen Roboters, so wird dies deutlich.

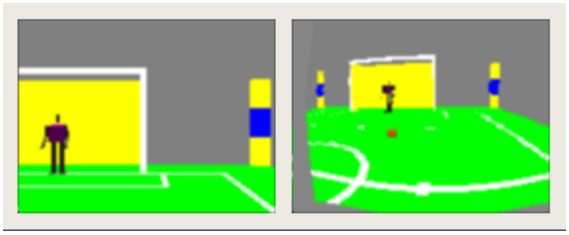

Abbildung 3.8: Kopf- und Bauchkamerabilder des Simulators

3.3.4 Entwicklungsumgebung

Durch die Verwendung von *RoboCup06App* ist die Programmiersprache auf C++ festgelegt. Als Entwicklungsumgebung kommt *Microsoft VisualStudio 2005* zum Einsatz. Für die Arbeit mit dem Pocket PC ist *Microsoft ActiveSync 4.0* und *Windows Mobile 5.0 SDK for Pocket PC 5.0* zu installieren. Um *RoboCup06GUI* kompilieren zu können, wird *Trolltech QT 4.0* für die plattformunabhängige, grafische Benutzeroberfläche (GUI) benötigt.

4 Konzept

Die vorhandene Steuerungssoftware *RoboCup06App*, die in den Grundlagen näher beschrieben ist, wird durch funktionale Komponenten erweitert, die eine Berücksichtigung von Hindernissen und Spielern ermöglichen. Diese Komponenten kapseln logisch zusammenhängende Datenverarbeitungsschritte und werden als Module bezeichnet. Ihre Erstellung orientiert sich an der bestehenden Struktur von *RoboCup06App*. Zwei Bereiche der Steuerungssoftware werden durch die Erweiterungen berührt.

Der erste Teil ist für die Wahrnehmung („*Sense*") der Umwelt zuständig, denn Hindernisse und Spieler müssen zunächst einmal erkannt werden. Da ein Kamerasystem die wichtigsten Sensordaten liefert, handelt es sich hierbei um den Bildverarbeitungsteil. Dort existieren bereits Module, die jeweils für die Erkennung bestimmter Objekte (Ball, Tore, Feldlinien, ...) zuständig sind. Neben diesen *Perceptoren* werden nun weitere zur Erkennung von Hindernissen und Spielern implementiert. Dies ist zum einen der *ObstaclesPerceptor*, ein Spezialist für die Wahrnehmung von Hindernissen. Bei der Spielererkennung unterscheidet man hingegen zwischen dem *GoalieInGoalPerceptor* für eine Torwarterkennung und dem *OpponentPerceptor* für eine allgemeine Spielererkennung. Die Konzepte der neuen Komponenten werden im Abschnitt Hindernis- und Spielererkennung (4.1) näher beleuchtet. Die drei genannten Wahrnehmungsspezialisten sind als Erweiterung der Bildverarbeitung zu sehen.

Der zweite Bereich, in den Neuerungen einfließen müssen, betrifft die Planungsebene der Steuerungssoftware („*Plan*"). Die Informationen über Hindernisse und Spieler können nicht direkt aus dem aktuellen Bild in die Verhaltenssteuerung einfließen. Sie müssen vielmehr über eine Anzahl von Frames zusammengetragen und aufbereitet werden. Beispielsweise ist es von Interesse zu wissen, wann und wo der Ball zuletzt gesehen wurde. Dazu ist eine Modellierung der Daten nötig, um auf der Verhaltensebene kontinuierlich auf solche aufgewerteten Daten zugreifen zu können. Das dafür zuständige Weltmodell übernimmt auch die Aufgabe der Modellierung für fast alle Wahrnehmungstypen, insbesondere auch die der Spieler. Aufwändigere Modellierungen, wie die des Balls, sind in einem eigenen Modul untergebracht. Auch die Modellierung der Hindernisse fällt komplexer aus. Sie

soll daher durch ein zusätzliches Modul realisiert werden, welches parallel zum Weltmodell agiert. Die Struktur der Steuerungssoftware sieht allerdings vor, dass das Weltmodell die einzige Schnittstelle zur Verhaltensebene bildet. Daher werden alle Anfragen des Verhaltens an die Hindernismodellierung über das Weltmodell weitergeleitet.

Die Verhaltensebene verwertet diese Informationen und gibt entsprechende Befehle an den Bewegungsapparat („*Act*") weiter.

Die Struktur und Interaktionen der neuen Komponenten sind in Abbildung 4.1 graphisch dargestellt. Dabei sind neue Softwaremodule in rot gehalten, bestehende in weiß. Durch die Erweiterungen am Weltmodell ist dieses schraffiert dargestellt. Die hellblauen Elemente dienen dabei der Veranschaulichung und sind nicht mit Softwarekomponenten gleichzusetzen.

Die folgenden Abschnitte erläutern die Konzepte der neu angelegten Module, bzw. die Erweiterungen bestehender Komponenten. Dem Schaubild 4.1 entsprechend wird zunächst die Erkennung behandelt und dann auf den Modellierungsteil (4.2) eingegangen. Das anschließende Kapitel 5 beschreibt das Vorgehen und die Funktionsweisen im Detail.

4.1 Hindernis- und Spielererkennung

Da Hindernisse im Allgemeinen nicht durch eine spezielle Farbe gekennzeichnet sind, ist zu überlegen, was als solches erkannt werden soll. Prinzipiell gilt alles als Hindernis, womit der Roboter kollidieren kann. Bei Zwei-gegen-Zwei-Spielen kommen dafür Tore, Landmarken, eigene und fremde Roboter, sowie Personen (z.B. Schiedsrichter) in Frage. Der Ball soll hier außer Acht gelassen werden. Neben diesen Zwei-gegen-Zwei-Spielen ergeben sich noch weitere technische Aufgabenstellungen (vgl. 3.2.3). Gerade beim Dribbling im Slalom wäre es von großem Vorteil, ebenfalls auf eine Hinderniserkennung zurückgreifen zu können.

Obwohl technisch möglich, soll eine Hinderniserkennung aus Effizienzgesichtspunkten grundsätzlich nicht zwischen den verschiedenen Hindernistypen unterscheiden und so generisch wie möglich arbeiten. Eine Unterscheidung bietet sich jedoch in den Fällen an, in denen eine Interaktion mit dem Hindernis möglich ist. Dies gilt in besonderem Maße für Mitspieler, bei welchen sich beispielsweise ein Passspiel anbietet. Auch im Hinblick auf die Berechnung eines Schusswinkels beim Torschuss sind Informationen über die Position des gegnerischen Torwarts sinnvoll verwertbar. In diesen Fällen erfolgt eine Differenzierung anhand der Kenntnisse um unterschiedliche charakteristische Merkmale.

Eine Spielererkennung ist ein Spezialfall der Hinderniserkennung, denn sie muss

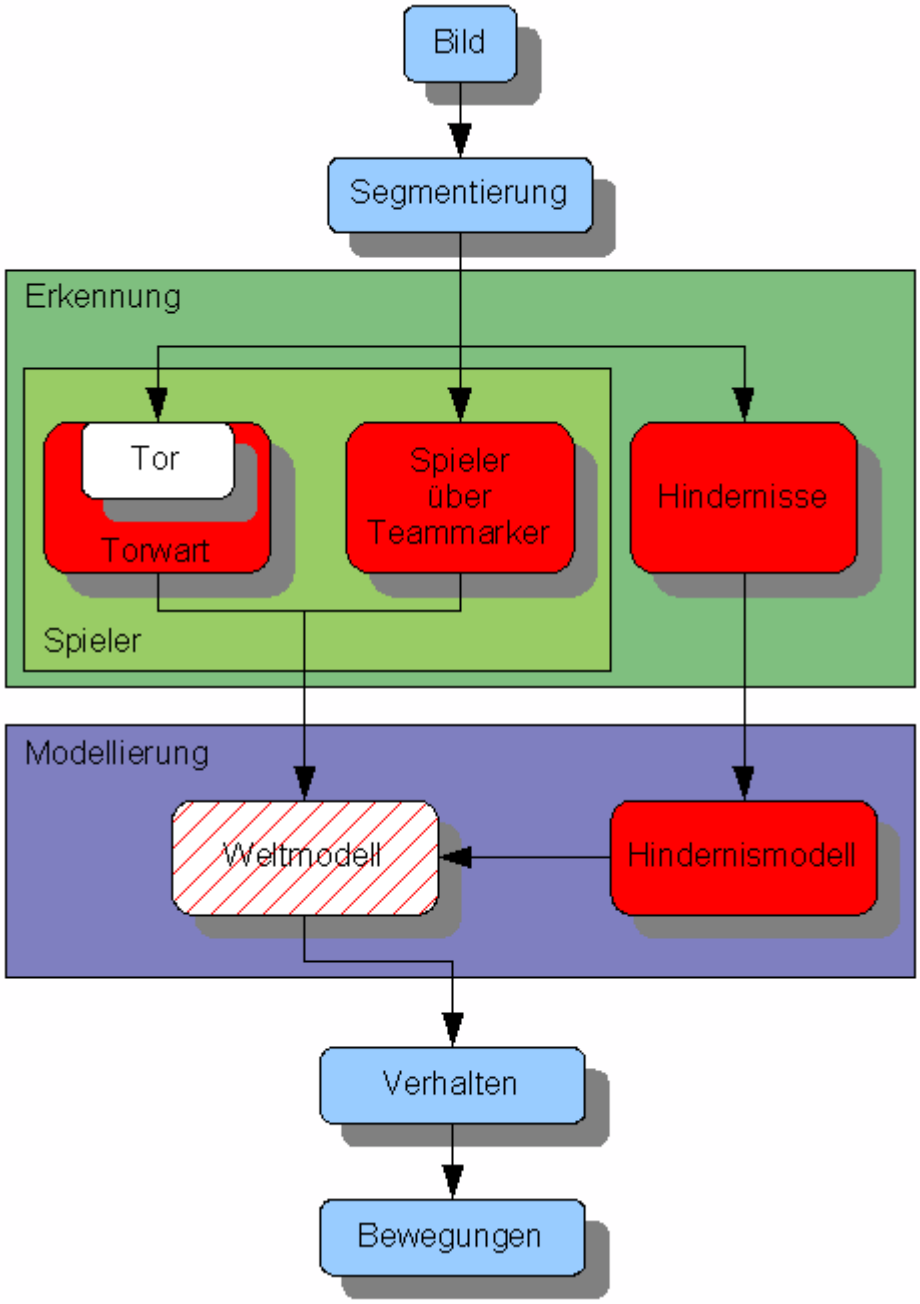

Abbildung 4.1: Integrationsstruktur der Softwaremodule

bestimmtes Vorwissen mit einbringen, um einen Spieler von einem allgemeinen Hindernis unterscheiden zu können. Ein Spieler ist nicht vollständig durch eine eigene Farbe definiert, die es ermöglicht den Spieler in seiner Gesamtheit zu erfassen, denn die Roboter sind in der Praxis sehr unterschiedlich. Selbst die vorgeschriebenen Schwarztöne unterscheiden sich erheblich, so dass bei der Segmentierung für Roboter keine eigene Farbklasse existiert. Eine schwarze Säule in Form einer Landmarke als Hindernis wäre zum Beispiel nicht ohne weiteres von einem Spieler zu unterscheiden. Erst die Kenntnis über einen Formunterschied oder ein spezielles Merkmal, wie zum Beispiel ein Teammarker, machen die Differenzierung praktisch möglich.

Wie aufgezeigt, unterliegt die Suche nach Hindernissen im Allgemeinen, gegenüber Spielern im Besonderen, verschiedenen Bedingungen und wird daher im Folgenden getrennt skizziert.

4.1.1 Hinderniserkennung

Wie bereits angesprochen, soll die Hinderniserkennung nicht zwischen verschiedenen Typen von Hindernissen unterscheiden. Nach Möglichkeit sollen alle Objekte erfasst werden mit denen der Roboter kollidieren kann. Da also keine Differenzierung stattfinden soll, kann auch nicht von einer Objekterkennung gesprochen werden. Das einzige Merkmal von Interesse, welches alle Hindernisse gleichermaßen besitzen, ist deren Abstand zum Roboter.

Zur Ermittlung dieser Abstandswerte sollen nun, strahlenförmig vom Roboter aus gesehen, Distanzwerte bis zum jeweils nächsten Hindernis bestimmt werden. Dazu wird jedes Bild an mehreren Stellen vom unteren Rand nach oben untersucht. Solange man beim Durchsuchen eine Bodenfarbe betrachtet, handelt es sich dabei um Freiraum. Stößt man jedoch auf eine andere Farbe, so handelt es sich aller Wahrscheinlichkeit nach um den Anfangspunkt eines Hindernisses. Für jede betrachtete Stelle soll maximal ein solcher Punkt gefunden werden, denn nur das jeweils nächste Hindernis ist von Interesse. Für jedes Bild können so mehrere Abstandswerte ermittelt werden. Durch eine Umrechnung von Bildkoordinaten kann man den Abstand zum Roboter in mehrere Richtungen bestimmen. Die Strecke vom Roboter bis zu jedem dieser Punkte wird jeweils als Freiraum gewertet. Die Bestimmung ist verständlicherweise nur im Bereich des Öffnungswinkels der Kamera möglich. Diese Vorgehensweise erlaubt keine Aussage bezüglich der beteiligten Hindernisobjekte aus den ermittelten Werten, noch nicht einmal deren Anzahl lässt sich bestimmen. Die Menge der gefundenen Distanzen mit zugehörigen Blickrichtungen eines Bildes werden der Modellierung für weitere Berechnungen übergeben.

Die Hinderniserkennung arbeitet mit den Bildern der Bauchkamera, die starr am Oberkörper des Roboters befestigt ist und nach vorne zeigt. Die Bauchkamera besitzt einen horizontalen Öffnungswinkel von 100°. Der Sichtbereich der Kopfkamera nimmt mit 45° einen geringeren Ausschnitt der Umwelt wahr. Durch die identische Auflösung können daher aber weiter entfernte Objekte besser wahrgenommen werden. Die bewegliche Montage ermöglicht der Kopfkamera die Erfassung eines insgesamt größeren Blickfelds. Sie ist daher für gezielte Detailbetrachtungen vorgesehen und dient vorwiegend der Ballerkennung. Wollte man zusätzlich Hindernisse erfassen, fiele die Beobachtungszeit des Balls durch die ausgelastete Steuerungssoftware geringer aus. Dies ist aktuell nicht zu vertreten, wäre mit einer höheren auswertbaren Framerate allerdings möglich. Da die potentiellen Hindernisse beim RoboCup deutlich größer ausfallen als der Ball, wäre eine ausschließliche Verwendung der Kopfkamera ohnehin fraglich, denn die Bauchkamera ist für die Erfassung großer Objekte besser geeignet. Eine zusätzlicher Gebrauch der Kopfkamera zur gezielten Betrachtung von Hindernisdetails, gerade mit zunehmender Entfernung, könnte hilfreich sein, ist aber aus Performancegründen gegenwärtig nicht realisierbar.

Das Softwaremodul *ObstaclesPerceptor* ist für die Detektion von Hindernissen zuständig und arbeitet parallel zu den weiteren Wahrnehmungsspezialisten (vgl. auch Abbildung 4.1). Wegen der vorgelagerten Segmentierung kann nur auf dem *Scanlineraster* operiert werden (vgl. 3.3.1.1) und durch die beschriebene Betrachtungsweise eines Bildes findet eine Beschränkung auf die vertikalen *Scanlines* statt.

Für die angesprochenen Bodenfarben kommen innerhalb des RoboCups nur die Farbklassen grün und weiß in Frage. Um den Ball nicht als Hindernis zu erkennen wird die Farbe orange hinzugenommen; alle anderen werden als Hindernisfarben interpretiert. Durch die Erkennung nach einer Bildsegmentierung müssen die Bodenfarben als Nicht-Hindernisse definiert werden. Objekte mit einer Bodenfarbe können aus diesem Grund auch nicht als Hindernisse erkannt werden.

4.1.2 Spielererkennung

Bei der Erkennung von anderen Robotern auf dem Spielfeld werden zwei Fälle unterschieden. Zum einen sollen Torwart-Roboter erfasst werden, um Torschüsse besser platzieren zu können. Dem steht die allgemeine Spielererkennung gegenüber. Während im zweiten Fall Vorwissen über den Roboter an sich, nämlich in Form des Teammarkers, herangezogen wird, finden bei der Torwarterkennung die Kenntnisse über den Hintergrund (Torfarbe) Verwendung. Diese unterschiedliche Herangehensweise legt eine separate Darstellung der beiden Module nahe. Für

beide Varianten wird jeweils ein eigener *Perceptor* angelegt.

4.1.2.1 Torwarterkennung

Gegenüber der allgemeinen Spielererkennung dient hier nicht der Teammarker, sondern der Kontrast gegenüber dem Hintergrund (das Tor) als Differenzierungskriterium. Die Farbtabelle sieht aufgrund der unterschiedlichen verwendeten Roboter keine eigene Klasse für die ganzheitliche Erfassung vor. Damit fallen die Bildpunkte von Robotern zum größten Teil in die Restklasse („unknown"). Durch die Torfarbe im Hintergrund hebt sich damit ein Torwart deutlich ab. Der *GoalieInGoalPerceptor* wird als Unterklasse des Torerkenners angelegt. So kann auf benötigte Informationen über ein erkanntes Tor zugegriffen werden, ohne diese in überflüssiger Weise ressourcenbindend erneut extrahieren zu müssen.

Die Erkennung des *Goalies* arbeitet nur mit den horizontalen *Scanlines*, da nur bei diesen ein Tor-Roboter-Tor-Übergang erwartet werden kann. Aus den Bildkoordinaten der detektierten Übergänge wird der Mittelpunkt der erkannten Roboterausschnitte als Zentrum des Torwarts berechnet.

Es ist derzeit nicht möglich, daraus die exakte Position des Torwarts bezüglich des eigenen Roboters zu bestimmen. Dies liegt an der beschriebenen Umrechnung von Bildpunkten in das Roboterkoordinatensystem (vgl. 3.3.1.1). Es kann aber ein recht genauer Blickwinkel ermittelt werden. Wird in einem Frame ein Torwart erkannt, so wird der Winkel zu diesem als *Percept* für das Weltmodell bereitgestellt. Durch die Modellierung des Weltmodells kann die Verhaltensebene jederzeit über diese Information verfügen und beispielsweise bei der Bestimmung eines Schusswinkels berücksichtigen.

4.1.2.2 Erkennung über Teammarker

Die Spielererkennung über Teammarker wird im Zwei-gegen-Zwei-Spiel nicht für eigene Roboter verwendet und ist somit streng genommen eine reine Gegnererkennung. Eigene Roboter sollten ihre Position kennen und können diese über *Teammessages* an die eigenen Mitspieler weitergeben. Somit kann die Rechenzeit, die zur Erkennung eines eigenen Spielers notwendig wäre, eingespart werden.

Es sind durchaus Situationen denkbar, in denen die Erkennung eigener Roboter sinnvoll ist. Zum Beispiel könnte die Erkennung von eigenen Mitspielern bei der Selbstlokalisierung helfen. Dazu müsste der Roboter die über die Spielererkennung ermittelten Positionen der Mitspieler mit den über *Teammessages* ausgetauschten Positionen kombinieren, um seine eigene Position bestimmen zu können. Eine andere Möglichkeit wäre präzise Pässe zum Mitspieler ausführen zu können, ohne da-

bei die Ungenauigkeiten der Selbstlokalisation beider Roboter in Kauf nehmen zu müssen. Derzeit werden diese Möglichkeiten aus Gründen der Ressourcenschonung nicht in Betracht gezogen, könnten aber in Zukunft durch ein größeres Spielfeld mit mehreren Mitspielern und höherer Rechenkapazität an Bedeutung gewinnen.

Die Funktionsweise der Erkennung von gegnerischen und eigenen Robotern unterscheidet sich nicht; sie hängt lediglich von der Farbe der Teammarker ab. Es ist problemlos möglich, die Erkennung in Zukunft auch für eigene Spieler zu nutzen. Aktuell ist die Spielererkennung über Teammarker, wie bereits erwähnt, als reine Gegnererkennung anzusehen.

Der *OpponentPerceptor* soll gezielt nach den Teammarkern der gegnerischen Roboter suchen. Die Farben der Marker sind eindeutig und haben keine mehrfache Verwendung auf dem Spielfeld. Die Erkennung arbeitet sowohl mit den horizontalen als auch vertikalen *Scanlines*.

Sind alle Fragmente einer Markerfarbe im Bild identifiziert, so muss entschieden werden, ob diese für eine Erkennung ausreichen. Wurden genügend Fragmente gefunden, so ist zu klären, ob diese zu einem oder mehreren Robotern gehören. Es sind Anhäufungen von Fragmenten, sogenannte *Cluster*, für jeden Roboter zu erwarten. Ein Clusteringverfahren erfasst diese Anhäufungen und bestimmt deren Anzahl, sowie das jeweilige Zentrum [1]. Die Clusterzentren werden jeweils als Markermittelpunkt interpretiert. Abschnitt 5.2.2 erläutert die Funktionsweise des verwendeten Algorithmus im Detail.

Es lässt sich nun, wie bei der Torwarterkennung auch, die Richtung zu jedem gefundenen Mittelpunkt vom eigenen Roboter aus gesehen, bestimmen. Ist die Identifikation mindestens eines Teammarkers erfolgt, so werden alle Informationen über erkannte Marker in einem *OpponentPercept* zusammengefasst und für die Modellierung bereitgestellt.

4.2 Modellierung

Jeder *Perceptor* erzeugt im Falle einer positiven Erkennung seiner Bildmerkmale ein *Percept*, welches die extrahierten Informationen in komprimierter Form enthält. Ein *Percept* repräsentiert aber immer nur die Daten des aktuellen Frames. Um mit Kenntnissen über kürzlich wahrgenommene Merkmale arbeiten zu können, bedarf es einer Modellierung. Diese muss eine zumindest vorübergehende Speicherung vornehmen, so dass diese Informationen nicht verloren gehen. Die Verlässlichkeit der Daten hängt stark von deren Alter ab, so dass dieses ebenfalls von der Modellierung berücksichtigt werden muss. Die aufbereiteten Daten werden so zur Verfügung gestellt, dass die Verhaltensebene kontinuierlich darauf zugreifen kann und anhand

des Alters der Daten selbst entscheidet, ob diese noch verlässlich genug sind, um verwendet zu werden.

Bei der Betrachtung eines *Percepttyps* bei mehreren aufeinander folgenden Bildern kann es sein, dass gegebenenfalls auch kein *Percept* erzeugt wird, obwohl sich das zu erkennende Objekt im Bild befindet. Dies kann beispielsweise an einem fehlerhaften oder unscharfem Bild liegen. Zudem springen die Positionsangaben von *Percept* zu *Percept* durch die Bewegungen des Roboters. Die Modellierung kann eine Glättung über mehrere *Percepte* vornehmen und die so aufgearbeiteten Daten der Verhaltensebene zur Verfügung stellen.

Modellierungen dieser einfacheren Art werden für die gegnerischen Roboter benutzt und im Abschnitt 4.2.2 beschrieben. Das Konzept der Hindernismodellierung fällt durch das Erstellen eine Karte der Umwelt wesentlich komplexer aus und wird in Abschnitt 4.2.1 genauer erklärt.

4.2.1 Modellierung der Hindernisse

Die Modellierung der Hindernisse erstellt mit Hilfe der *Hindernispercepte* ein Abbild der Umwelt, indem die Abstandsinformationen der *Percepte* in einer Freiraumkarte zusammengetragen werden. Da nicht nur die aktuell vor dem Roboter liegenden Hindernisse von Interesse sind, ist ein Modell wünschenswert, welches zu jedem Zeitpunkt möglichst genaue Informationen über Hindernisse rund um den Roboter zur Verfügung stellen kann.

Durch die Bewegungen des Roboters nimmt dieser unterschiedliche Ausschnitte seiner Umwelt wahr. Trägt man die Informationen der verschiedenen Umweltausschnitte in einer Karte zusammen, so umfasst das Abbild der Umwelt mehr Information als der wahrgenommene Ausschnitt des aktuellen Blickwickwinkels hergibt.

Der Roboter „merkt“ sich also kürzlich gesehene Hindernismerkmale, indem er eine Repräsentation seiner näheren Umwelt in einem „visuellen Sonar“ [10] speichert. Das egozentrische Modell erinnert stark an einen Radarschirm und entspricht der typischen Darstellungsform der Daten von 360°-Sensoren. Dabei fällt der Mittelpunkt mit dem Ursprung des kartesischen Roboterkoordinatensystems zusammen.

Das Modell ist in Sektoren aufgeteilt. Für jeden Sektor soll ein Abstandswert bis zum nächsten Hindernis gespeichert werden. Das Modell wird mit neuen *Percepten* der Hinderniserkennung fortwährend aktualisiert. Mit der Menge der Punkte des aktuellen *Percepts* lassen sich die Abstandswerte der Sektoren neu errechnen. Die Punkte werden dabei den einzelnen Sektoren zugeordnet und in jeweils einem Abstandswert zusammengefasst. Das Modell kann durch *Percepte* nur in dem Bereich aktualisiert werden, der dem momentanen Sichtfeld des Roboters entspricht. Die

anderen Bereiche der Karte werden gefüllt, wenn der Roboter sich bewegt.

Bevor die Aufnahme der erkannten Hindernispunkte einer neuen Position in das Modell erfolgen kann, muss das Modell aktualisiert werden, denn die relative Position zu einem bereits eingetragenen Hindernispunkt hat sich durch die Bewegung des Roboters verändert. Alle Punkte im Modell müssen so transformiert werden, dass die neu berechneten relativen Positionen möglichst gut mit der Realität übereinstimmen. Die Transformation der Punkte entspricht dem Weg, den der Roboter in der Zwischenzeit zurückgelegt hat. Der Weg lässt sich mathematisch durch eine Drehung und eine Verschiebung darstellen. Der Roboter versucht durch die interne Beobachtung seiner Laufbewegungen diese so genannten Odometriedaten zu liefern (siehe Abschnitt 3.3.1.2).

Die Abstandswerte, die aus älteren *Percepten* gewonnen wurden, verlieren zunehmend an verwertbarer Qualität. Um dem gerecht zu werden, wird der Abstandswert auf den Initialwert zurückgesetzt, sobald er ein gewisses Alter überschritten hat. Durch diese Maßnahme und bedingt durch die jüngsten Bewegungen des Roboters, kann es sein, dass die aktuelle Karte nicht vollständig ist.

4.2.2 Modellierung der Spieler

Die Position von gegnerischen Robotern soll möglichst genau bestimmt werden können, selbst wenn der eigene Roboter diese gerade gar nicht sehen kann.

Informationen über andere Roboter werden im Weltmodell gespeichert und modelliert. Hierzu gehört der relative Winkel zu jedem gegnerischen Roboter sowie jeweils eine Prozentangabe, die ein Maß für die Verlässlichkeit des Winkels darstellen soll. Je länger ein Roboter nicht mehr gesehen wurde, desto geringer fällt dieser Wert aus.

5 Realisierung

5.1 Hinderniserkennung und -modellierung

Die Hinderniserkennung versucht aus den Bilddaten der Roboter die maximal freie Fläche in Blickrichtung zu bestimmen. Hierfür ist der *ObstaclesPerceptor* zuständig, auf den im folgenden Abschnitt genauer eingegangen wird. Die Hindernismodellierung, welche anschließend näher beschrieben wird, beschäftigt sich mit der Aufbereitung und Repräsentation der gewonnenen Erkenntnisse. Mit Hilfe der Daten der Hinderniserkennung wird eine Freiraumkarte erstellt und aktuell gehalten. Abfragemethoden geben beispielsweise eine zusammenfassende Auskunft über den vor dem Roboter liegenden Freiraum. Diese Vorgehensweise orientiert sich an Arbeiten zur Hindernisvermeidung aus der *Sony Four-Legged Robot League* [6, 10].

5.1.1 Hinderniserkennung

Vorauswahl Wie bereits im Konzeptteil erwähnt, arbeitet die Hinderniserkennung nur mit den vertikalen *Scanlines* der Bauchkamera, nachdem diese bereits die Farbsegmentierung durchlaufen haben. Die *Scanlines* enthalten also nur Segmente, die aus den definierten Farben (Tabelle 3.1) bestehen. Davon werden alle Farben mit Ausnahme der Bodenfarben (grün, weiß) und der Ballfarbe (orange) als Hindernisfarbe gewertet.

Es wird nun über die Segmentgruppen gleicher Farbe jeder einzelnen vertikalen *Scanline* von unten nach oben iteriert, bis entweder eine Hindernisfarbe gefunden oder der obere Bildrand erreicht wird. Pro *Scanline* kann also kein oder genau ein solches Segment gefunden werden. Um sicher zu gehen, dass es sich auch tatsächlich um den Anfang eines Hindernisses handelt, muss vorher eine Bodenfarbe detektiert worden sein. Dies ist nicht möglich, wenn das Objekt sehr groß ist oder aber sehr nahe am Roboter steht, da dann die *Scanline* bereits unten mit dem Objekt und nicht mit einer Bodenfarbe beginnt. Daher werden solche Hindernissegmente mit einer Mindestlänge von 30 Pixeln ebenfalls akzeptiert.

Um die Anfälligkeit gegenüber Störpixeln einzuschränken, muss das Bodensegment mindestens drei und das Hindernissegment mindestens vier Bildpunkte um-

fassen. Zwei aufeinander folgende, durch Störpixel unterbrochene Segmente gleichen Typs werden dabei verbunden. Diese Werte wurden durch Tests im Simulator und am Roboter ermittelt.

Die den Einschränkungen genügenden Übergänge werden in einer Liste gespeichert. Die Bezeichnung als *CandidateRuns* verdeutlicht, dass es sich bei diesen um Kandidaten handelt, die im Fokus einer weiteren Betrachtung liegen. Sie stellen eine Sammlung von Punkten dar, an denen wahrscheinlich ein Hindernis beginnt.

Filterung Da die Vorauswahl noch etliche unerwünschte *CandidateRuns* enthalten kann, wird eine Filterung anhand unterschiedlicher Kriterien vorgenommen, um die Qualität der Erkennung zu erhöhen.

Beginnt das Hindernissegment eines *CandidateRuns* oberhalb des berechneten Horizonts, so wird dieser entfernt. Denn es handelt sich nicht um einen Bodenpunkt des potentiellen Hindernisses bzw. ist es in jedem Fall so weit entfernt, dass es nicht von besonderem Interesse ist. Außerdem wäre eine Bestimmung der Entfernung sehr ungenau, da in der Nähe des Horizonts ein Bildpunkt mehreren Dezimetern in Roboterkoordinaten entspricht.

Durch die begrenzte Auflösung der Kamera entstehen, besonders bei nicht orthogonal im Bild verlaufenden Spielfeldlinien, am Übergang zum Teppich verschwommene Bereiche. Dieser Effekt ist in der Literatur unter dem Begriff „Aliasing" bekannt [3]. Die angesprochenen Bildausschnitte fallen in die Klasse der nicht erkannten Farben und stellen somit potentiell Hindernisse dar. Es sind nur die Bereiche problematisch, welche nahezu parallel zu den vertikalen *Scanlines* verlaufen und sich im Nahbereich des Roboters befinden. Diese sind lang genug und können sich nach der Vorauswahl noch unter den Kandidaten befinden. Durch eine Erhöhung der Mindestlänge eines Hindernissegments würde die Erkennung von echten Hindernissen erschwert. Daher wird das Hindernissegment eines jeden *Runs* in Roboterkoordinaten abgebildet und seine Strecke vermessen. Unterschreitet die Länge den Schwellwert von 10 cm, so wird der entsprechende *Run* herausgefiltert. Die Vermessung des Hindernissegments auf diese Art ist kein exaktes Verfahren, da die Punkte nicht wie erwünscht in das Roboterkoordinatensystem umgerechnet werden können (vgl. 3.3.1.1). Es hat sich allerdings herausgestellt, dass die Inkaufnahme dieser Ungenauigkeit trotzdem zufrieden stellende Ergebnisse liefert.

Zeitweise bestand das Problem, dass sich eigene Roboterteile, wie Füße oder Arme, im Bild befanden. Damit diese nicht fälschlicherweise als Hindernisse anerkannt werden, wurde ein Mindestabstand zum Roboter für alle Anfangspunkte von Hindernissen eingeführt. Wegen der starren Montage der Bauchkamera und der aktuellen Bewegungsmuster, besonders nach der baulichen Verkürzung der

Arme, besteht das Problem derzeit nicht. Bei einer Veränderung der Bewegungen müssen eventuell Maßnahmen ergriffen werden um eine unerwünschte Detektion auszuschließen.

Auch wenn die Filterung hier der Übersichtlichkeit halber schrittweise aufgeführt ist, wird jeder *CandidateRun* nur einmal betrachtet und dabei gleichzeitig nach allen Kriterien bewertet.

Erzeugung des Percepts Sofern mindestens ein *Run* verblieben ist, wird ein *Percept* erzeugt. Für jeden *Run* wird der Anfangspunkt des Hindernissegments in Bild- und Roboterkoordinaten dem *ObstaclesPercept* hinzugefügt. Das *ObstaclesPercept* wird an das *ObstacleModel* versendet, welches die weitere Verarbeitung der gewonnenen Informationen übernimmt.

Abbildung 5.1 zeigt links ein Bauchkamerabild eines Spielfeldausschnitts und rechts die vollständige Segmentierung der Szene. Im linken Bild sind außerdem verschiedene Zeichnungen zu sehen. Das graue Raster stellt die aktuelle Kameramatrix dar, wobei die weiße Linie den Horizont bildet. Mit Hilfe der Kameramatrix wird bei der Umrechnung der Bildpunkte in Roboterkoordinaten die Ausrichtung der Kamera berücksichtigt.

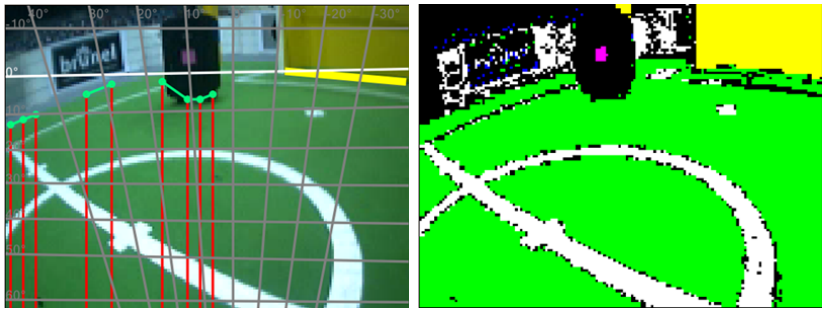

Abbildung 5.1: Hinderniserkennung im Bauchkamerabild mit Debugzeichnungen

Die hellgrünen Punkte sind Anfangspunkte von erkannten Hindernissen; sie stellen mit den Verbindungslinien das *Hindernispercept* dar. Die roten Strecken beschreiben die Abschnitte der vertikalen *Scanlines* bis zu einem identifizierten Bodenpunkt. Abbildung 5.2 zeigt das entsprechende *Hindernispercept* im Roboterkoordinatensystem. In beiden Bildern ist außerdem das gelbe *Torpercept* als weiterer Anhaltspunkt eingezeichnet.

Die roten Strecken in Bild- und Roboterkoordinaten entsprechen sich nur ungefähr und dienen der Veranschaulichung. Dies hängt vor allem mit der perspektivischen Verzerrung zusammen. Besonders deutlich wird dies bei der Betrachtung

von Abbildung 5.1 links, indem man die roten Linien mit dem Raster der Kameramatrix vergleicht. Durch die Parallelität der *Scanlines* in Bildkoordinaten kann es bei der Projektion in das Roboterkoordinatensystem keinen gemeinsamen Schnittpunkt geben.

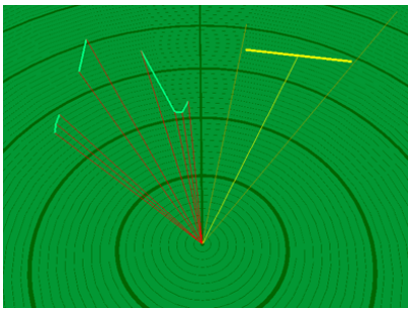

Abbildung 5.2: Detektierte Hindernisse im Roboterkoordinatensystem

Einschränkungen Es ist nicht möglich Hindernisse zu erkennen, die aus den Bodenfarben oder der Ballfarbe bestehen, da diese als Hindernisse ausgeschlossen wurden. Eine Erkennung von Hindernissen auf einem Untergrund, der nicht aus einer Bodenfarbe besteht, ist mit der gewählten Methode ebenfalls nicht möglich. Die Übergänge von Spielteppich zum Fußboden werden in der Regel als Beginn eines Hindernisses gewertet. Dies ist zum Beispiel in Abbildung 5.1 links zu sehen.

5.1.2 Hindernismodellierung

Die Aufgabe der Hindernismodellierung wird durch zwei Klassen realisiert. Das *ObstacleModel* beinhaltet das Abbild der Umwelt und bietet gleichzeitig eine Menge von Funktionen an, die zum einen zur Aktualisierung notwendig sind und zum anderen Abfragen auf dem Modell erlauben. Das *ObstaclesModelModule* enthält zur Laufzeit eine Instanz des *ObstacleModels*. Es steuert den Prozess der Aktualisierung und stellt das *ObstacleModel* für eine Abfrage durch die Verhaltensebene bereit. Eine zusätzliche lokale Kopie des *ObstacleModels* ermöglicht die weitere Verwendung bisher gespeicherter Hindernisse. Das *ObstaclesModelModule* führt in festen Zeitabständen (z.Z. alle 500 ms) eine Aktualisierung des *ObstacleModels* durch, welche aus drei Phasen besteht:

1. Alterung der Modells
2. Transformation entsprechend der Odometrie
3. Update durch das neue *Hindernispercept*

Die Alterung wird grundsätzlich immer durchgeführt, die beiden anderen Phasen nur, wenn neue Informationen vorliegen. In der Zeit bis zur nächsten Aktualisierung können mehrere Updates durch die Odometrie eintreffen, die alle Berücksichtung finden. Durch die gewählten Timings von Bauchkamera und Hindernismodellierung trifft aktuell maximal ein *Hindernispercept* pro Aufruf des *Obstacles-ModuleModules* ein. Durch eine Veränderung der Taktzeiten oder eine zusätzliche Verwendung der Kopfkamera könnten mehrer *Percepte* eintreffen. In diesem Fall müssten aus Gründen der Genauigkeit die drei Phasen mehrfach - für jedes *Percept* einmal - durchlaufen werden. Der Ort, an dem ein *Hindenrispercept* erzeugt wurde, muss über die Odometriedaten rückwirkend bestimmt werden, um die enthaltenen relativen Abstandsinformationen richtig in das Modell aufnehmen zu können.

Zunächst wird der Aufbau des Modells erklärt. Anschließend werden die Schritte einer Aktualisierung näher beleuchtet. Beim Start des Roboters ist das Modell noch leer. Weder der Alterungsschritt noch die Transformation wird auf einen leeren Sektor angewendet. Ein leerer Sektor kann allerdings in einem Transformationsschritt einen Distanzwert zugewiesen bekommen.

Das Sektormodell Das Sektormodell erinnert stark an einen Radarschirm, wie bereits im Abschnitt 4.2.1 erwähnt. Es wird intern durch ein *Array* repräsentiert. An jeder Stelle steht ein Objekt der Klasse *Sector*, welches jeweils drei Werte speichert: eine Distanz, einen Verlässlichkeitswert und einen Punkt.

Die Distanz gibt den Abstand in Millimetern bis zum nächsten Hindernis in der durch den Sektor bestimmten Richtung an. Da der Roboter sich nur auf dem Spielfeld bewegen sollte, dessen Länge ca. 4 m beträgt und Hindernisse in einer größeren Entfernung nicht zuverlässig erkannt werden, wird eine Maximaldistanz von 4 m eingeführt. Weiter entfernt liegende Hindernisse werden ignoriert. Dem Maximalwert kommt eine Sonderstellung zu, denn ein entsprechender Sektor wird als leer angesehen und so interpretiert, dass kein Hindernis vorhanden ist. Er wird bei der Initialisierung und beim Zurücksetzen eines Sektors verwendet und kann nicht überschritten werden. Falls sich die Voraussetzungen der Hinderniserkennung, zum Beispiel durch eine höhere Auflösung und Framerate der Kamera, verändern sollten, kann dieser Wert einfach angepasst werden, denn er ist an einer zentralen

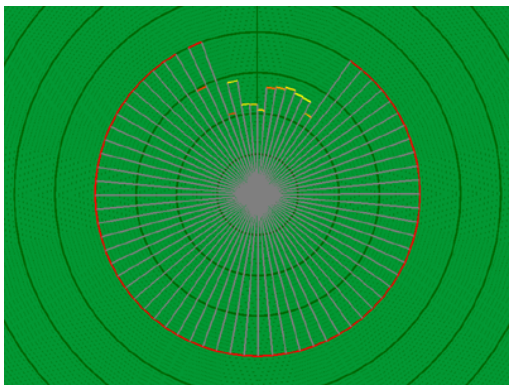

Abbildung 5.3: Hindernismodellierung: Das Sektormodell

Stelle redundanzfrei definiert.

Der Verlässlichkeitswert ist eine prozentuale Angabe, der über die Qualität der gespeicherten Distanz Auskunft gibt. Der Verlässlichkeitswert steht in Abhängigkeit zum Alter der Punkte, die in die Berechnung des Distanzwertes eingeflossen sind. Wird die eingetragene Distanz eines Sektors bei einer Aktualisierung durch ein neues *Percept* bestätigt, so erhöht sich der Verlässlichkeitswert.

Für die Transformationen des Modells ist es notwendig, zu jedem Sektor einen Punkt zu speichern, der repräsentativ für diesen Sektor steht (siehe weiter unten). Aktuell wird mit 72 Sektoren gearbeitet, so dass jeder Sektor einem Bereich von 5° entspricht. Die Anzahl der Sektoren kann ebenfalls an zentraler Stelle festgelegt werden.

Abbildung 5.3 zeigt das Modell im Roboterkoordinatensystem. Die Distanz der meisten Sektoren ist gleich dem Initialwert von 4 m, was durch den fast vollständigen Kreisbogen zu erkennen ist. Im vorderen Bereich wurden Abstandswerte aus der Hinderniserkennung eingetragen. Das Kreissegment eines Sektors ist seinem Verlässlichkeitswert entsprechend zwischen rot (0 %) und hell-gelb (100 %) fließend eingefärbt.

Das Sektormodell dient der Einschränkung der Anzahl der gespeicherten Hindernisdaten und sorgt außerdem für eine gleichmäßige örtliche Verteilung der Informationseinheiten. Würde man alle Hindernispunkte der *Percepte* mit einem Zeitstempel speichern und nur in Abhängigkeit zur Zeit wieder löschen, so ließe sich keine Aussage über die zu speichernde Datenmenge treffen. In Abhängigkeit dazu steht auch die Rechenzeit des Moduls, besonders während der Transformation mit Hilfe der Odometrie. Durch die Einschränkung der Datenmenge kann die Hindernismodellierung effizient arbeiten und der maximale Speicherbedarf sowie

die Rechenzeit sehr gut abgeschätzt werden. Dies erleichtert die Verteilung der ohnehin knappen Ressourcen der Steuerungseinheit auf die vielen verschiedenen Softwarekomponenten.

5.1.2.1 Alterung des Modells

Bei der Alterung des Modells senken sich die Verlässlichkeitswerte aller Sektoren um den gleichen Betrag. Erreicht der Wert 0 %, so wird der entsprechende Sektor auf die Maximaldistanz zurückgesetzt. Sektoren im Sichtbereich des Roboters verringern sich allerdings um den fünffachen Betrag. Der Grund dafür ist, dass keine Negativinformationen der Hindeniserkennung bei der Aktualisierung durch die Bildverarbeitung in das Modell einfließen. D.h. Sektoren im Sichtbereich, die während der Aktualisierung durch ein *Hindernispercept* keinen neuen Punkt zugewiesen bekommen, werden nicht zurückgesetzt. In einer Reihe von *Percepten* kann es durch Bildrauschen und Roboterbewegungen vorkommen, dass einige *Percepte* gewisse Hindernispunkte nicht enthalten. Das Hindernismodell soll den Effekt des Springens von Distanzwerten in diesen Fällen vermeiden. Daher rührt die Nichtbeachtung der Negativinformationen. Es handelt sich dabei um eine Art Glättung, denn einzelne *Percepte* fallen so weniger ins Gewicht. Die Aktualisierung durch die Bildverarbeitung erhält dadurch einen höheren Stellenwert, da mehrfach nicht bestätigte Sektoren im Sichtbereich schneller zurückgesetzt werden. Beispielsweise lässt sich so ein als Hindernis erkannter Schiedsrichter, der sich auf dem Spielfeld bewegt, schneller wieder aus dem Sektormodell entfernen.

Der Betrag, um den der Verlässlichkeitswert heruntergesetzt wird, besteht zunächst aus einem definierten Grundwert. Er entspricht dem Maximalwert, um den ein Sektor in einem Schritt gealtert werden kann. Der Grundwert verändert sich nun in Abhängigkeit zur Zeit, indem der aktuelle Zeitstempel mit dem der letzten Aktualisierung verglichen wird. Bei einer Differenz zwischen 0 und 1000 ms wird der Grundwert linear mit einem entsprechenden prozentualen Faktor multipliziert; ab 1000 ms bleibt der Faktor konstant bei 1. Dies ermöglicht ein *Debugging* mit *Breakpoints*, ohne dass dabei eine vollständige Zurücksetzung erfolgt. Im regulären Betrieb überschreitet die Differenz der Zeitstempel diese Grenze nicht.

5.1.2.2 Aktualisierung durch ein Hindernispercept

Erhält das *ObstaclesModelModule* neue Informationen von der Bildverarbeitung in Form eines *ObstaclesPercepts*, so werden diese in das Sektormodell eingebracht. Wie bereits erwähnt, umfasst die Aktualisierung durch die Bildverarbeitung immer genau den Ausschnitt des Modells, der dem momentanen Sichtbereich des Roboters

entspricht.

Bevor die Punkte eines *Hindernispercepts* nacheinander in das Modell eingetragen werden, findet noch eine minimale Glättung durch das *ObstaclesModelModule* statt. Ziel ist es, dabei einzelne Ausreißerpunkte im Bild zu eliminieren. Jeder Punkt der Menge wird mit seinen beiden Nachbarn bezüglich seiner y-Bildkoordinate verglichen. Die beiden Randpunkte dienen dabei nur zum Vergleich und werden selbst nicht in das Modell eingetragen. Gleiches gilt dann, wenn die Differenz zu mindestens einem der Nachbarn mehr als 30 Pixel beträgt und die beiden Nachbarn dabei nicht mehr als die gleiche Toleranz auseinander liegen.

Für jeden verbleibenden Punkt wird über trigonometrische Funktionen jeweils ein normalisierter Winkel im Bogenmaß und die Distanz zum Roboter bestimmt. Der Winkel entscheidet über den Sektor, in den der Punkt fällt. Der Repräsentant und die Distanz des entsprechenden Sektors bekommen die neuen Werte zugewiesen. Die Verlässlichkeit erhält einen Initialwert von 40 %. Falls die neue Distanz im Bereich des bereits eingetragenen Wertes liegt, gilt dies als Bestätigung. Die Toleranz liegt bei 30 cm in beide Richtungen. Für diesen Fall wird die Verlässlichkeit aus dem Maximum des alten Wertes und des Initialwertes gebildet und schließlich um 20 % angehoben, sofern dies möglich ist. Dadurch verbleiben diese Punkte länger im Hindernismodell.

5.1.2.3 Transformation (Aktualisierung durch Odometriewerte)

Das Modell muss, wie im Konzept angesprochen, vor jeder Aktualisierung durch ein *Hindernispercept* transformiert werden, um die relativen Abstände der Hindernisse aus dem *Percept* richtig in das Modell eintragen zu können. Die nötige Information liefert dabei die Odometrie über das Weltmodell.

Die Odometriedaten liegen nun in Form einer Rotation und einer Translation vor. Die Rotation wird durch einen Winkel im Bogenmaß angegeben, die Translation besteht aus zwei Komponenten (x, y). Sie stellen eine Verschiebung entlang der beiden orthogonalen Achsen des Roboterkoordinatensystems in Millimetern dar. Da es sich um ein egozentrisches Modell handelt, fällt die Applikation der Rotation leicht und wird daher auch als erstes beschrieben.

Rotation Damit bei wiederholten Drehungen des Roboters nicht jedes mal das ganze Modell rotiert werden muss, wird ein *Offset* mitgeführt. Dieser *Offset* fasst die einzelnen Drehbewegungen zu einer einzigen zusammen. Sobald die Odometrie eine Translation beinhaltet, wird die Drehung des *Offsets* in die Verschiebung der Punkte miteinbezogen.

Die von der Odometrie gelieferten Winkel für die Rotation fallen relativ klein aus, denn diese werden mehrmals pro Sekunde bereitgestellt. Um Rundungsfehler zu vermeiden wird der *Offset* als Fließkommazahl geführt, aus dem die Verschiebung um ganze Sektoren oder Vielfache für den Zugriff auf den *Array*, berechnet werden kann. Die *Array*-Positionen stimmen also nicht notwendigerweise mit den Sektornummern überein. Beim Start des Roboters ist dies der Fall; durch dessen Drehungen kommt es zu Verschiebungen.

Das Modell muss entgegengesetzt zur Drehung des Roboters rotiert werden, d.h. der Winkel der Odometrie wird immer vom *Offset* subtrahiert. Bei einem Zugriff auf einen Sektor wird zunächst der ursprüngliche *Array*-Index bestimmt und um die mögliche Verschiebung um ganze Sektoren gemäß dem *Offset* korrigiert. Danach erfolgt der eigentliche *Array*-Zugriff. Bei den Berechnungen eines Index ist zu beachten, dass dieser im vorgegebenen Wertebereich liegt. Dies wird durch eine einfache Modulo-Rechnung gewährleistet.

Translation Die zweidimensionale Verschiebung des Modells fällt etwas komplexer aus. Es ist nicht möglich die Distanzwerte der Sektoren direkt zu translieren, Punkte jedoch lassen sich leicht verschieben und ähnlich wie ein Update durch die Bildverarbeitung wieder in das Sektormodell überführen. Der erste Ansatz bestand darin, für jeden Sektor einen solchen Punkt zu bestimmen. Ausgenommen sind davon Sektoren mit der Maximaldistanz, da diese als leer gelten und sonst Hindernisse entstehen, wo keine sind. Die Winkelhalbierende beschreibt dabei jeweils die Ausrichtung des gesuchten Vektors, dessen Länge durch den gespeicherten Distanzwert gegeben ist. Der Vorteil lag darin, dass für jeden Sektor mit Hilfe trigonometrischer Funktionen normalisierte Vektoren vorberechnet werden konnten. Diese musste man dann nur noch mit dem jeweiligen Distanzwert multiplizieren. Bei Tests am realen Roboter stellte sich jedoch heraus, dass die Translationen betragsmäßig sehr klein ausfallen. Dabei kommt es bei den Verschiebungen nur in den wenigsten Fällen zur Überschreitung einer Sektorgrenze. Da im nächsten Schritt wieder ein zum Teil vorberechneter Punkt als Ausgangsbasis für eine Translation dient, gehen kleine Verschiebungen praktisch verloren. Im Prinzip handelt es sich um einen Rundungsfehler. Die Hindernismodellierung erzeugt dadurch ein falsches Abbild der Umwelt. Sie funktioniert so nicht und daher wurde das Konzept aufgegriffen, für jeden Sektor einen Repräsentantenpunkt mitzuführen. Daraus ergibt sich folgende Vorgehensweise:

Sobald die Odometrie eine Translation liefert, beginnt die Aktualisierung des Modells. Die Repräsentanten von nicht leeren Sektoren werden mit den zugehörigen Verlässlichkeitswerten gespeichert. Die lokale Kopie des Sektormodells wird

an dieser Stelle zurückgesetzt; die Punktmenge enthält alle nötigen Informationen. Jeder Punkt wird nun transformiert. Dabei fließen sowohl der externe *Rotationsoffset,* als auch die Verschiebung mit ein. Die neu berechneten Punkte werden mit entsprechenden Verlässlichkeiten der Reihe nach in Abhängigkeit ihres Winkels den Sektoren zugeordnet. Die Distanzwerte der betroffenen Sektoren berechnen sich dabei aus dem zugewiesenen Punkt. Ergeben sich nach der Transformation Punkte, die weiter entfernt liegen als die maximale Distanz, so werden diese verworfen. Der Vorgang ähnelt insgesamt der Aktualisierung durch ein *Hindernispercept.*

5.2 Gegnererkennung und -modellierung

Es soll gezielt nach gegnerischen Robotern gesucht werden, um deren Position auf dem Spielfeld möglichst genau bestimmen zu können.

5.2.1 Torwarterkennung

Bei diesem Ansatz sollen die Teammarker bei der Erkennung nicht gesondert beachtet werden. Ein Roboter ist schwer zu erkennen, denn die in der Humanoidliga eingesetzten Modelle unterscheiden sich in der Farbgebung. Daher gibt es in der Farbtabelle keine eigene Klasse für eine Roboterfarbe. Alle Pixel, die bei der Segmentierung nicht einer Klasse einer definierten Farbe zugewiesen wurden, gehören zur Klasse „unknown“. Roboter vor einem unbekannten Hintergrund sind nicht zu erkennen, da sie mit diesem bei der Segmentierung verschmelzen können. Die nötige Information geht bei diesem Schritt also verloren.

Es ist davon auszugehen, dass sich ein Roboter aufrecht vor dem generischen Tor befindet. Somit kann die Torfarbe im Hintergrund zur Erkennung des sich davor befindlichen Roboters ausgenutzt werden. Der Roboter besteht aus einer Menge von Pixeln der Klasse „unknown“ und hebt sich von der Torfarbe ab. Abbildung 5.4 zeigt die Segmentierung eines Simulatorbildes, bei der sich der *Goalie* gut vom Tor abhebt.

Innerhalb der horizontalen *Scanlines* sollen Tor-Roboter-Tor-Übergänge (*Runs*) identifiziert werden. Als Torfarbe kommen gelb und blau in Frage. Der Torerkenner arbeitet in der Regel nur für das gegnerische Tor und wird entsprechend initialisiert. Stellvertretend steht im Folgenden gelb als Torfarbe. Pixel der Klassen „unknown”, sowie Teammarkerfarben (cyan, magenta) kommen als Roboterfarben in Frage (vgl. Tabelle 3.1).

Ein einfacher endlicher Automat erledigt die Detektion der gesuchten Übergänge (Abbildung 5.5). Jede *Scanline* durchläuft den Automaten, jeweils im Startzustand

Abbildung 5.4: Segmentiertes Simulatorbild eines Torwarts

beginnend. Dabei werden die *Scanlinesegmente* der Reihe nach betrachtet und entsprechend ihre Farbe ein Zustandswechsel des Automaten ausgelöst. Das aktuell betrachtete Segment wird durch ein Übergangssymbol (T: Torfarbe, R: Roboterfarbe, X: andere Farbe) charakterisiert.

Der Endzustand (3) wird bei der Folge „Tor-Roboter-Tor" über die Zustände 1 und 2 erreicht, die jeweils einen Zwischenschritt darstellen. Bei einer Unterbrechung der gesuchten Folge findet ein Wechsel in den Startzustand statt. Im Falle einer Überlappung (2. Torsegment gleich dem ersten des folgenden *Runs*) wechselt der Automat von Zustand 3 nach 2.

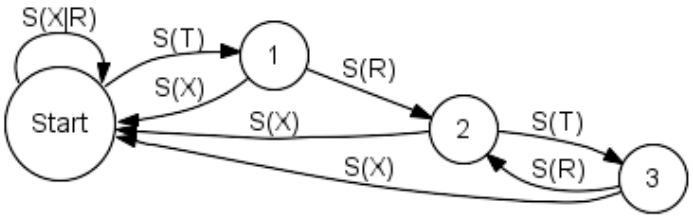

Abbildung 5.5: Torwarterkennung mit Hilfe eines endlichen Automaten

Für jeden identifizierten Übergang (Zustand 3 wurde erreicht) wird ein entsprechendes *Run*-Objekt angelegt und in einem Vektor gespeichert. Dieses Objekt enthält die Anfangs- und Endpunkte der drei aufeinander folgenden Segmente eines Übergangs in Bildkoordinaten.

Ob diese *Runs* tatsächlich über dem Torwart und nicht an anderen Stellen im Bild liegen, können wir als Mensch recht schnell erkennen. Liegt ein *Run* wie erwünscht über dem *Goalie*, so spricht man von einem positiven *Run*. Anderenfalls wird von einem negativen *Run* gesprochen. Diese Bewertung bezieht sich auf die Bewertung durch einen Menschen. Der Roboter versucht anhand der Vorauswahl

und der Filterung möglichst viele positive *Runs* aus dem Bild zu extrahieren.

Vorauswahl Die Vorauswahl versucht aus jedem Bild möglichst viele *Runs* zu bestimmen die den Torwart erfassen. Gleichzeitig wird angestrebt, dabei die Anzahl der *Runs* zu minimieren, die nicht über dem Torwart liegen. Wichtiger ist hierbei der erste Punkt, denn weitere Einschränkungen lassen sich durch späteres Filtern erreichen. Die in der Vorauswahl gefundenen *Runs* sind als Kandidaten zu sehen, die weiter betrachtet werden müssen. Analog zur Hinderniserkennung werden diese ebenfalls als *CandidateRuns* bezeichnet.

Zunächst kann man folgende Beobachtungen machen: Sehr kurze *Runs* sind überall im Bild zu finden, können jedoch nur eingeschränkt als die erwünschten Tor-Roboter-Tor Übergänge identifiziert werden. Bei genauerer Betrachtung stellt man fest, dass einzelne, anders-farbige Pixel am Torwart eine Erkennung der unmittelbar angrenzenden Tor- und Torwartsegmente verhindern. Dies ist durch Störpixel zu erklären, welche vor allem bei Farbübergängen auftreten und daher ignoriert werden sollen. Gleichzeitig werden Mindestlängen für die Segmente eines *Runs* eingeführt, um die Erkennungsgüte zu erhöhen. Denn die Erkennung von kurzen, unerwünschten *Runs* nimmt durch das Tolerieren der Störpixel noch zu. Zur Bestimmung der Mindestlängen der einzelnen Segmente und der Toleranzen bei Störpixeln wurden empirische Untersuchungen angestellt. Bei den Torsegmenten hat sich gezeigt, dass man mit einer Mindestlänge von vier Pixeln gute Ergebnisse erzielt. Die untere Grenze des mittleren Segments ist auf drei Pixel festgelegt. Höhere Werte führen dazu, dass der Torwart nicht mehr erkannt wird, wenn der Roboter weiter als zwei Meter von Torwart entfernt ist.

Bei der Bestimmung der Anzahl der zu tolerierenden Störpixel hat sich gezeigt, dass bis zu sechs Pixel ignoriert werden sollten. Dieser Wert mag zunächst hoch erscheinen, lässt sich aber anhand eines Beispiels begründen. Folgt auf ein erkanntes gelbes Segment ein Störpixel einer anderen Farbe, so würde ein anschließendes gelbes Segment bis zu einer Länge von drei Pixeln, wegen der zuvor festgelegten Mindestbreite für ein Torsegment (vier Bildpunkte), zu diesen gerechnet. Es wäre auch möglich, die Anzahl herunterzusetzen und Störpixel, die auf ein erkanntes Segment folgen, zu diesem hinzuzufügen. Das verwendete Verfahren funktioniert ausreichend gut und es wurden keine Probleme mit einer zu großen Anzahl von Störpixeln festgestellt. Im Falle von zwei als gleich erkannten, also auch ausreichend langen Segmenten, die durch tolerierte Störpixel getrennt sind, werden diese Segmente zusammengeführt.

Abbildung 5.6 zeigt die dem letzten Bild zugehörige Simulatorszene. Dabei beschreiben die eingezeichneten Linien die im Optimalfall gefundenen Übergänge

innerhalb der horizontalen *Scanlines* nach der Vorauswahl.

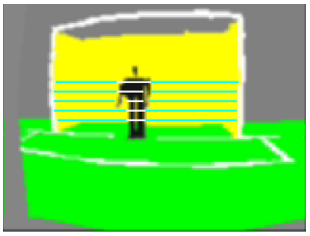

Abbildung 5.6: Simulatorbild mit Debugzeichnungen des Torerkenners

Filterung Die *CandidateRuns* werden gefiltert, um die Erkennungsgüte zu verbessern. Dies ist vor allem bei folgendem Problem von Bedeutung.

Das Gebiet zwischen Tor und Landmarke besteht oft aus Pixeln, die in die Klasse der nicht definierten Farben fällt. Die Landmarken bestehen aus Farben, die auch für die Tore verwendet werden. Es kann vorkommen, dass *Runs* erkannt werden, die von einer Landmarke bis zum Tor verlaufen.

Der schwarz segmentierte Teil dazwischen wird fälschlicherweise als Roboter interpretiert. Die Problematik soll durch Abbildung 5.7 veranschaulicht werden.

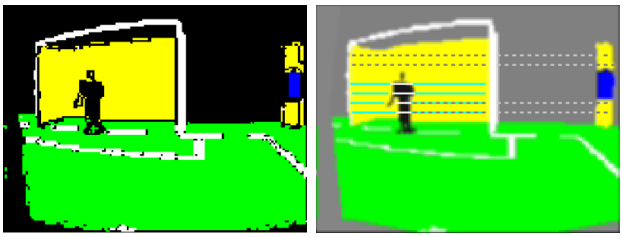

Abbildung 5.7: Problem mit Landmarken

Links ist die Segmentierung eines im Simulator aufgenommenen Bildes zu sehen, wodurch die Schwierigkeit der Unterscheidung von Roboter und der Fläche zwischen Tor und Landmarke deutlich wird. Die eingezeichneten Linien im zugehörigen Simulatorbild auf der rechten Seite zeigen detektierte Übergänge *(CandidateRuns),* die jeweils durch drei aufeinander folgende Lininenabschnitte dargestellt sind. Bei den unteren beiden betroffenen *Scanlines* ist eine Überlappung zu beobachten. Die durch gepunktete Linien repräsentierten *Runs* fließen nicht in die weiteren Berechnungen ein, weil sie außerhalb des Tors liegen und daher auch nicht den Torwart erfassen. Das maschinelle Verfahren zur Unterscheidung wird im Folgenden näher beschrieben.

Das schwarze Segment des *Runs* ist verglichen mit dem Segment der Landmarke relativ breit. Daher lag zunächst die Idee nahe, mit einer Maximalbreite für das Torwartsegment zu arbeiten. Dieser Ansatz stellte sich nach wenigen Tests als problematisch heraus. Zum einen ist bei Perspektiven schräg zum Tor das mittlere Segment nicht mehr auffallend breit und zum anderen erzeugt ein Torwart, aus geringer Distanz betrachtet, sehr ähnliche *Runs*. Es wurde noch über eine dynamische Bestimmung einer fixen Maximalbreite nachgedacht, die sich zum Beispiel aus dem Verhältnis der Längen der gelben und des schwarzen Segments berechnen ließe. Bei genauerer Betrachtung stellte sich heraus, dass dies nicht möglich war, denn die Längen der Segmente hängen zu stark von der Perspektive und der Entfernung zum Tor ab. Es wäre also nützlich, die Tordistanz in die Berechnungen einzubeziehen. Dabei ergibt sich aber ein neues Problem:

Durch die parallele Arbeitsweise aller *Perceptoren*, also aller Spezialisten für die Erkennung bestimmter Objekte, haben diese jeweils nur Kenntnis über ihre zu erkennenden Objekte und können nicht interagieren. Im konkret betrachteten Fall verfügt der Torerkenner allein über das nötige Wissen. Denkbar wäre es auch, die nötigen Informationen aus dem Weltmodell zu beziehen. Die Aufgaben der *Perceptoren* sind aus organisatorischen Gründen hinsichtlich der Objektorientierung so eingeschränkt, dass sie nicht mit diesem interagieren sollen. Der logische Schluss wäre also, die Torwarterkennung auch das Tor erkennen zu lassen. Den bereits vorhandenen Programmiercode des Torerkenners zu kopieren, spräche gegen das Paradigma der Redundanzvermeidung. Durch die Verwendung einer objektorientierten Programmiersprache ist es möglich, die Torwarterkennung als Erbe der Torerkennung zu implementieren. Zunächst wird die Torerkennung durch die Elternklasse vorgenommen und so gegebenenfalls ein *Torpercept* erzeugt, dessen Daten anschließend bei der Erkennung des Torwarts Verwendung finden.

Die ursprüngliche Idee war, Berechnungen mit der Distanz zum Tor durchzuführen, um die Falscherkennung eines Torwarts zwischen Landmarke und Tor auszuschließen. Die Entscheidung, die Daten des Torerkenners zu nutzen, brachte weitere Vorteile mit sich. Denn dieser stellt nicht nur die Tordistanz bereit, sondern auch eine recht genaue Position und Länge der Torgrundlinie. Dies bietet die Möglichkeit direkt zu überprüfen, ob sich das schwarze Segment eines *CandidateRuns* innerhalb der durch die Torlinie beschriebenen Ränder befindet. *Runs* mit schwarzen Segmenten, die teilweise oder ganz außerhalb liegen, werden verworfen. Aufgrund ihrer Größe können die Tore sehr präzise erkannt werden, so dass eine fälschliche Erfassung eines Torwarts außerhalb des Tors nahezu ausgeschlossen ist. Dieses Ergebnis wäre mit der ursprünglichen Idee nicht sicher gewesen. Die Möglichkeit bietet den weiteren Vorteil, zeitintensive Tests zur Bestimmung der Berechnungs-

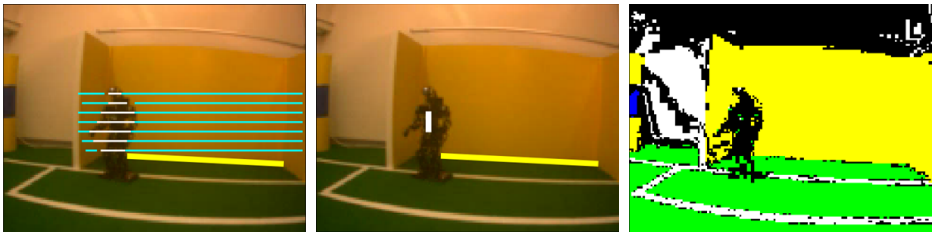

Abbildung 5.8: Torwarterkennung in Pfostennähe

vorschrift mit Hilfe der Tordistanz vermeiden zu können. Von Nachteil ist, dass ohne die Erkennung eines Tores kein Torwartroboter erkannt wird. Allerdings tritt dieser Fall äußerst selten auf und wiegt nicht so schwer wie ein falsch erkannter Roboter außerhalb seines Tors.

Die erkannte Torlinie ist im Kamerabild oft nicht ganz so breit wie die Fläche der Torfarbe. Daher wird ein Torwart, der sich dicht am Pfosten befindet, oft nicht erkannt. Um dies zu kompensieren, wird ein *Run* trotzdem akzeptiert, wenn er sich etwas neben dem erkannten Tor befindet (vgl. Abb. 5.8). Das erste Bild zeigt einen Roboter links im gelben Tor. Die gelbe Linie repräsentiert das erkannte *Torpercept*. Die in cyan und weiß gehaltenen Linien zeigen die *CandidateRuns* der Torwarterkennung, die den *Goalie* recht gut erfassen. Das mittlere Bild zeigt die aus den erfassten Übergängen berechnete Robotermitte - das *GoalieInGoalPercept* (weiße Markierung). Diese befindet sich links neben dem erkannten Tor, der Torwart wäre ohne die eingeführte Toleranz nicht erkannt worden. Zur Veranschaulichung zeigt das Bild rechts die vollständige Segmentierung der Szene. Die Torwarterkennung erfasst das Tor bei solchen Perspektiven besser als die allgemeine Torerkennung. Dies wird beim Vergleich der Breite des Torpercepts und der *CandidateRuns* in Abbildung 5.8 links deutlich. Der Grund dafür ist die ausschließliche Verwendung der vertikalen *Scanlines* durch die Torerkennung. Aus den *Runs* werden zusätzliche seitliche Begrenzungen des Tors ermittelt und das *GoalPercept* dadurch optimiert. Dies war nur durch die gewählte Struktur der Torwarterkennung als Unterklasse der Torerfassung möglich.

Ist die Distanz zum Tor bekannt, so kann auch die Erkennung des Torwarts aus geringeren Entfernungen verbessert werden. Durch die relativ gering gewählte Mindestbreite von drei Pixeln in der Vorauswahl gibt es aus größerer Entfernung immer noch *Runs*, die den Torwart erfassen. Befindet sich der Roboter jedoch selbst nahe am Tor, so erhöhte sich die Anzahl der negativen *Runs*. Arme und Beine des Roboters werden durch einzelne *Runs* abgedeckt und erfassen nicht, wie erwünscht, den Roboterkörper. Die *CandidateRuns* werden auf die Länge ihres

Mittelsegments überprüft. Bei einer geringeren Distanz als 80 cm zum Tor muss dieses mindestens acht Pixel aufweisen, sonst wird der *Run* verworfen.

Erzeugung des Percepts Nach der Filterung müssen mindestens drei *CandidateRuns* aus verschiedenen Zeilen des Bildes verbleiben, damit ein *Percept* des Torwartroboters erzeugt und verschickt wird. Ist die Anzahl der verbleibenden *Runs* geringer, so erhöht sich die Gefahr, dass ein noch verbleibender negativer *Run* zu einem falsch erkannten Torwart führt. Wählt man die Grenze indes höher, so kommt es vermehrt vor, dass der Torwart trotz genügend positiver *Runs* nicht erkannt wird.

Aus den verbleibenden *Runs* soll zunächst der Mittelpunkt des Torwartroboters im Bild möglichst genau bestimmt werden. Um dies zu erreichen, werden nur noch die Mittelpunkte der schwarzen Segmente eines jeden *Runs* betrachtet. Errechnet man aus diesen Punkten den Durchschnitt, so werden alle *Runs* gleich bewertet. Schon wenige negative *Runs* können das Ergebnis ungünstig beeinflussen. Es ist erwünscht, einzelne, stärker abweichende Punkte nach Möglichkeit nicht mit in die Berechnung des Mittelpunktes einfließen zu lassen. Für eine erforderliche Glättung bietet sich die Bestimmung des so genannten Median an. Dabei werden Elemente einer Dimension sortiert. Die mittlere Position des Vektors bestimmt sodann den Median. Gegenüber der Bildung eines Durchschnitts, fließen Werte mit einer hohen Abweichung vom Mittelwert nicht in die Berechnung mit ein.

Für jeden *Run* wird zunächst der Mittelpunkt des schwarzen Segments bestimmt. Da ein *Run* immer genau in einer Bildzeile liegt, kann die Zeilennummer als y-Wert direkt verwendet werden. Der x-Wert berechnet sich aus dem gespeicherten Anfangs- und Endpunkt des Mittelsegments. Es werden nun zwei Vektoren angelegt. Ein Vektor besteht aus allen Zeilennummern der *Runs*, der andere enthält die errechneten x-Werte. Zu beachten ist dabei, dass doppelt vorkommende Werte nicht eliminiert werden dürfen, damit die Bestimmung des Medians auch den erwünschten Effekt zeigt. Beide Vektoren werden sortiert und anschließend wird jeweils das Element an der mittleren Position bestimmt. Beide Werte bilden zusammengenommen den Mittelpunkt des Torwarts.

Der ermittelte Punkt wird in Roboterkoordinaten umgerechnet, indem er entlang eines Sichtstrahls auf die Spielfläche projiziert wird. Da es sich dabei im Bild aber nicht um einen Bodenpunkt handelt, liegt der projizierte Punkt hinter der tatsächlichen Torwartposition (vgl. 3.3.1.1). Dies ist nicht weiter überraschend und wurde bereits im Konzeptteil erwähnt (4.1.2.1). Es lässt sich allerdings ein Winkel finden, der die Richtung des Torwarts bestimmt.

5.2.2 Gegnererkennung über Teammarker

Die Roboter eines Teams sind mit Markierungen versehen, so dass die Zugehörigkeit zu einem Team deutlich wird. Die Teams sind dazu angehalten, 8x8 cm große farbliche Markierungen, welche je nach Möglichkeit von jeder Seite aus sichtbar sein sollen, an den Oberkörper ihrer Roboter anzubringen (vgl. 3.2.1).

Ziel ist es, die gegnerischen Roboter anhand ihrer Teammarker zu erkennen. Bei der Erkennung treten einige Komplikationen auf. Da die Teammarker nicht besonders groß sind, wird es mit zunehmender Entfernung immer schwieriger, einen Marker identifizieren zu können. Erschwerend kommt hinzu, dass ein Marker ganz oder teilweise verdeckt sein kann. Besonders häufig ist zu beobachten, dass ein Roboter seinen eigenen Marker zur den Seiten hin mit seinen Armen verdeckt. Je kleiner die sichtbare Fläche eines Markers im Kamerabild ist, desto schwieriger wird es, diesen zu erkennen. Dies hängt sowohl mit der Bildauflösung der Kamera, als auch mit der Verwendung des *Scanlinerasters* zusammen.

Ein weiteres Problem stellt die Anzahl der im Bild befindlichen Roboter dar. Bei einem wäre die Bestimmung des Markermittelpunktes einfach. Es würde analog zur Torwarterkennung eine Median-Bestimmung zum Einsatz kommen. So muss aber bestimmt werden, welche sichtbaren Markerteile zu welchem Roboter gehören. Wenn man also einmal die Pixel der gewünschten Markerfarbe im Bild identifiziert hat, ergibt sich daraus eine Menge von zweidimensionalen Punkten. Es ist zu erwarten, dass gewisse Anhäufungen von Punkten jeweils um die Zentren eines Markerteils im Bild entstehen. Aufgabe ist es, genau diese Zentren zu bestimmen. Dabei können dicht beieinander liegende Zentren verschmelzen. Probleme dieser Art sind hinreichend bekannt, oft werden Clustering-Algorithmen zur deren Lösung eingesetzt [1].

Das Hauptproblem besteht allerdings darin, aus einem erkannten Markerzentrum die Position des Roboters zu bestimmen. Dies hängt damit zusammen, dass nur Bodenpunkte richtig in das Roboterkoordinatensystem umgerechnet werden können (vgl. 3.3.1.1). Ein Ansatz bestand darin, vom erkannten Marker aus im Bild über die Beine des Roboters bis zu einem seiner Fußpunkte zu gelangen. Diese Vorgehensweise findet Anwendung bei der bereits im Kapitel „Motivation" angesprochenen Arbeit zur Spielererkennung in der *Sony Four-Legged Robot League* [2]. Von einem identifizierten Marker aus findet eine strahlenförmige Suche in Richtung unterer Bildrand statt. Dabei werden weitere Bildpunkte außerhalb des *Scanline*rasters betrachtet.

Eine Übertragung des Konzepts auf Humanoidroboter ist aus zwei Gründen nicht möglich. Eine Betrachtung zusätzlicher Pixel kommt aufgrund der knappen Ressourcen der Steuerungseinheit nicht in Frage. Zum anderen bereitet die unter-

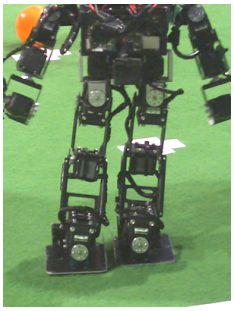

Abbildung 5.9: Rahmenteile der Beine eines Roboters

schiedliche Bauform in der Humanoidliga Probleme. Die zuverlässige Erfassung der Beine stellt die eigentliche Schwierigkeit dar. Durch die Lücke zwischen den Beinen und deren schlanke Form kommt es bei der verwendeten Kamera zu verschwommenen Farbübergängen. Manche der verwendeten Robotermodelle besitzen sehr viele Rahmenteile, durch die der Hintergrund scheint (Abb. 5.9).

Daher kommt es sehr oft zu einer schlechten Segmentierung, und eine zuverlässige Erfassung mit Hilfe der *Scanlines* ist nicht möglich. Es gibt Ausnahmen bei in unmittelbarer Nähe befindlichen Gegnern, die allerdings so weit entfernt sein müssen, dass sich unter ihren Füßen noch genügend Bodenfarbe im Bild befindet. Insgesamt betrachtet kann keine verlässliche Identifikation der Fußpunkte vorgenommen werden. Es ist also nicht direkt möglich, über einen Teammarker die Position eines Roboters zu bestimmen. Fraglich ist, welche Informationen man aus einem erkannten Marker ziehen kann. Es stellte sich heraus, dass der auf das Spielfeld projizierte Mittelpunkt des Markers weit hinter der eigentlichen Position des Roboters liegt, da es sich im Bild nicht um einen Bodenpunkt gehandelt hat (vgl. 3.3.1.1). Allerdings stimmt der relative Winkel zum projizierten Punkt recht genau mit dem relativen Winkel zum Roboter überein.

Erfassung Die Erfassung der Marker arbeitet sowohl mit horizontalen als auch vertikalen *Scanlines*. Innerhalb dieser werden alle Segmente einer Markerfarbe identifiziert. Diese müssen eine Mindestlänge von drei Pixeln aufweisen, um eine Detektion von Störpixeln zu vermeiden. Abbildung 5.10 zeigt im linken Bild einen liegenden Roboter mit magentafarbenem Teammarker. Die zwei kurzen Linienstücke in cyan stellen die innerhalb der *Scanlines* gefundenen Markersegmente dar. Zur Veranschaulichung ist im mittleren Bild die vollständige Segmentierung der Szene zu sehen. Das rechte Bild zeigt den erkannten Markermittelpunkt, der durch das im Folgenden beschriebene Clusteringverfahren aus den Mittelpunkten

der erkannten Markersegmenten berechnet wurde.

Abbildung 5.10: Erkennung über Teammarker

Clustering Es kommt das aus der Literatur bekannte *k-means*-Verfahren zum Einsatz [1]. Der Algorithmus geht dabei von einer festen Anzahl von Clusterzentren aus. Über eine Abstandsfunktion wird jeder Punkt dem Zentrum mit dem kleinsten Abstand zugeordnet. In einem weiteren Schritt findet eine Neuberechnung der Zentren aus den jeweils zugeordneten Punkten statt. Diese Schritte werden wiederholt ausgeführt. Der Algorithmus bricht dann ab, wenn keine Verbesserung der Clusterzugehörigkeit mehr möglich ist. Die Zentren werden in der Regel zu Beginn mit zufällig verteilten Punkten initialisiert.

Zurzeit kann davon ausgegangen werden, dass sich maximal zwei Roboter eines Teams gleichzeitig auf dem Spielfeld befinden. Der verwendete Algorithmus startet daher mit zwei Clusterzentren (k=2). Durch die Verwendung einer festen Anzahl ist die weitere Schonung von Ressourcen der Recheneinheit möglich. In Zukunft ist von einer Erhöhung der Spieleranzahl pro Team auszugehen; daher ist eine Erweiterung um wenige Spieler problemlos durchzuführen. Allerdings ist bei einer zunehmenden Anzahl die Qualität der Ergebnisse noch zu untersuchen.

Der Algorithmus terminiert nach vier Durchläufen, denn die ermittelten Zentren ändern sich nur noch marginal. Dies vermeidet den Verbrauch von unnötiger Rechenzeit. Die Initialisierung der Clusterzentren besteht aus festen, nicht zufällig gewählten Werten - den jeweiligen Mittelpunkten der rechten respektive linken Bildhälfte. Bekommt ein Zentrum während des ersten Schritts einer Iteration keinen Punkt zugewiesen, so erhält es den Wert (x, y) = (0, 0).

Erzeugung des Percepts Nach Ablauf des Algorithmus werden alle Zentren, die nicht mit dem Nullpunkt zusammenfallen, weiter betrachtet. Sehr nah beieinander gelegene Clusterzentren werden zusammengefasst. Die verbleibenden sollten nun jeweils einen Markermittelpunkt beschreiben, aus dem - analog zur Torwar-

terkennung - ein relativer Winkel zum eigenen Roboter bestimmt werden kann. Ein *OpponentPercept* fasst die ermittelten Informationen zusammen.

5.2.3 Gegnermodellierung

Die Modellierung der Gegner obliegt dem Weltmodell. Dazu werden der Anzahl der Spieler eines Teams entsprechend zwei Winkel gespeichert. Diese geben die Blickrichtungen aus Roboterperspektive an, in denen sich die beiden Gegner befinden. Jedem Winkel ist eine weitere Variable zugeordnet, die Auskunft über die Verlässlichkeit des zugehörigen Winkels gibt. Erhält das Weltmodell Informationen über Gegner aus einem Bild in Form eines *OpponentPercepts*, so findet eine Aktualisierung mindestens eines Winkels statt. Die zugehörige Verlässlichkeit bekommt den Wert eins zugewiesen. In jedem neuen Zyklus des Weltmodells ohne Aktualisierung werden die Verlässlichkeiten mit dem Faktor 0,95 multipliziert. Dies ist eine Alterungsfunktion, die auch bei anderen modellierten Daten, wie zum Beispiel dem Winkel zum gegnerischen Tor, Verwendung findet. Ein typisches Verhalten benutzt nur Werte mit einer zugeordneten Verlässlichkeit von über 0,5 und interpretiert die anderen auf Grund ihrer Alters als ungenau.

5.3 Bemerkung zu den gewählten Parametern

Die Werte für die Parameter der Hindernis- und Spielererkennung, wie z.B. Mindestlängen, stehen durch ihre einmalige Verwendung direkt an der entsprechenden Stelle im Programmcode. Durch den strukturierten und mit Kommentaren versehenen Code können diese leicht gefunden und in ihrem Zusammenhang erfasst werden. Eine Änderung der Parameter der Erkenner sollte nur in seltenen Fällen notwendig sein. Bei der Verwendung einer anderen Bildauflösung müsste beispielsweise eine Anpassung erfolgen.

Eine Ausnahme bilden einige Parameter der Hindernismodellierung (Tabelle 5.1). Durch deren (redundante) Verwendung lag eine Definition an zentraler Stelle[1] nahe. Hinzu kommt, dass eine Anpassung dieser Parameter wahrscheinlicher ist.

[1] Dateiname: obstacleModel.h

Name	**Beschreibung**
SECTORS	Die Anzahl der Sektoren des Hindernismodells (72).
DEGREE_PER_SECTOR	Der sich aus der Sektoranzahl ergebende Winkelbereich eines Sektors (5°).
MAX_SECTOR_DISTANCE	Die Maximaldistanz aller Sektoren, gleichzeitig der Initialwert, der einen leeren Sektor repräsentiert (4000 mm).
MAX_CONFIDENCE_AGING_STEP	Der Maximalwert für einen Alterungsschritt (0,02).
MAX_DISTANCE_TOLERANCE	Bestimmt den Toleranzbereich, der als Bestätigung eines Abstandswertes durch die Hinderniserkennung gewertet wird (300 mm).

Tabelle 5.1: Parameter des Hindernismodells

6 Ergebnisse

Wie bereits im Kapitel „Motivation" erläutert, soll ein Testverhalten zeigen, dass eine entsprechende Reaktion auf Hindernisse möglich ist. Durch die einfache Modellierung der Spieler ist eine Bewertung der Erkennung ausreichend. Daher kommt der Bewertung der Hinderniserkennung und -modellierung eine größere Bedeutung zu, welche zunächst detailliert beschrieben wird. Anschließend findet eine kürzere Bewertung der Spielererkennung statt, an die sich Performancetests anschließen.

6.1 Testverhalten für Hindernisse

Die neu gewonnenen Kenntnisse über Hindernisse sollen nach deren Modellierung für die Verhaltensebene nutzbar gemacht werden. Dazu werden Abfragemethoden auf dem Sektormodell angebracht. Zunächst soll für das Testverhalten nur bestimmt werden können, wie groß der Abstand bis zum jeweils nächsten Hindernis nach vorne und zu beiden Seiten des Roboters ist. Für jede Richtung wird über mehrere Sektoren des Modells die minimale Distanz dieser bestimmt. Nach vorne wird ein Bereich von 80° untersucht. Die Bereiche zu den Seiten schließen links respektive rechts jeweils mit einem Winkelbereich von 90° daran an.

Damit im Verhalten auf diese drei Distanzwerte zurückgegriffen werden kann, müssen entsprechende Symbole in der Beschreibungssprache XABSL [11, 12] hinzugefügt werden. Da anhand des Konzepts der Steuerungssoftware nur das Weltmodell als Schnittstelle für die Verhaltensebene vorgesehen ist, muss dieses alle Informationen der Hindernismodellierung bereitstellen. Dies wird durch entsprechend angebrachte Variablen und einfache Methoden zum Auslesen erreicht.

6.1.1 XABSL

Mit Hilfe von XABSL lassen sich Verhalten konzipieren, die durch einen endlichen Automaten beschrieben werden können. Das Verhalten besteht aus einzelnen Zuständen mit jeweils einer definierten Aktionsanweisung, von denen immer genau einer aktiv sein kann. Die Aktion wird solange ausgeführt, wie sich der Automat in

diesem Zustand befindet. Der Roboter kann auch nichts tun, indem die Anweisung *do_ nothing* als Aktion angegeben wird.

Für jeden Zustand werden Wechsel zu anderen Zuständen oder der Verbleib im selben beschrieben. Dies geschieht mittels der Implementierung konditionaler Abfragen über eine beliebige Auswahl von registrierten XABSL-Symbolen. Ein Symbol kann seinen Wert ändern und entspricht damit einer definierten Variablen. Es können Symbole verwendet werden, die direkt von XABSL zur Verfügung gestellt werden. Dazu gehört zum Beispiel die Angabe über die Zeitspanne, seit wann sich der Automat schon im aktuellen Zustand befindet. Außerdem können eigene Symbole definiert werden, z.B. für die neu hinzugekommenen Distanzen nach vorne und zu den Seiten.

6.1.2 Ein Testverhalten

Die Reaktion auf Hindernisse soll auf dem normalen Spielfeld getestet werden, wie es auch bei den Wettbewerben Verwendung findet (vgl. Abschnitt 3.2.2). Den auf dem Feld platzierten Hindernissen soll der Roboter ausweichen. Der Roboter soll dabei in der Hälfte des blauen Tors starten, denn das Ziel ist das kollisionsfreie Erreichen des gelben Tors.

Das Verhalten zum Erreichen des Zieles wurde wie folgt konzipiert und ist in Abbildung 6.1 graphisch dargestellt. Es besteht aus acht Zuständen (*init, finished, correct_ orientation_ to_ left, correct_ orientation_ to_ right, walk_ forward, avoid_ obstacle, walk_ left, walk_ right*), von denen *init* den Startzustand und *finished* den Endzustand beschreibt.

Der Roboter soll zunächst geradeaus laufen, was durch *walk_ forward* ausgedrückt wird. Solange die Freiraumdistanz nach vorne mindestens 30 cm beträgt, wird der Zustand beibehalten, ansonsten auf *avoid_ obstacles* geändert. Dieser Zustand entscheidet anhand der Distanzwerte zu Hindernissen auf den Seiten, ob nach rechts oder links ausgewichen werden soll. Sind zu beiden Seiten keine Hindernisse im Modell vorhanden, so weicht der Roboter immer zur linken Seite aus, in allen anderen Fällen zu derjenigen Seite, welche die größere Freiraumdistanz besitzt. Die entsprechenden Zustände sind *walk_ right* beziehungsweise *walk_ left*. Der Roboter führt als Aktion eine Gehbewegung zur entsprechenden Seite aus. Wird dabei die Distanz nach vorne wieder größer als 30 cm, so wird zurück zum Vorwärtslaufen gewechselt. Droht der Roboter bei der Seitwärtsbewegung gegen ein seitliche liegendes Hindernis zu stoßen, so wird der Zustand erneut zu *avoid_ obstacles* verändert, um einen erneuten Ausweichversuch zu starten.

Damit der Roboter das gelbe Tor erreichen kann, soll er immer in dessen Rich-

tung ausgerichtet sein. Dazu wird bei allen Zuständen, die eine Gehbewegung als Aktion haben (*walk_forward, walk_right, walk_left)*, ständig überprüft, ob der Winkel innerhalb eines Toleranzbereiches von 30° bewegt. Verlässt der Torwinkel den Toleranzbereich, so wird je nach Abweichungsrichtung in die Zustände *correct_orientation_to_left* oder *correct_orientation_to_right* gewechselt.

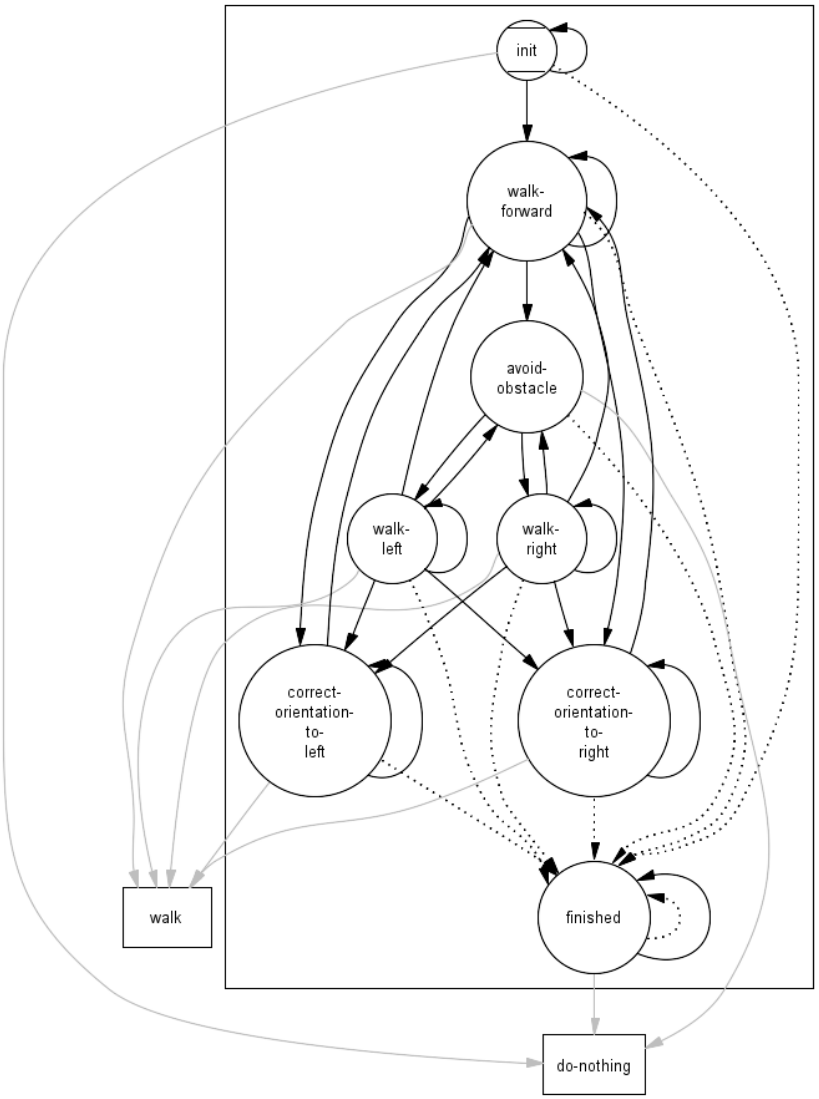

Abbildung 6.1: Verhalten zur Hindernisvermeidung

In diesen dreht der Roboter sich der Richtung entsprechend, bis der Torwinkel wieder im Toleranzbereich liegt. Der Zustand *walk_forward* wird wieder aktiv, sobald die Ausrichtung korrigiert ist.

Für alle Zustände gilt eine gemeinsame Übergangsentscheidung: Beträgt die Distanz zum Tor weniger als 1 m, so wird in den Endzustand *finished* gewechselt. Diese gemeinsamen Übergänge sind in der Abbildung mit gepunkteten Pfeillinien dargestellt.

6.1.3 Testläufe im Simulator

6.1.3.1 Setup

Für die Tests im Simulator wurde das Slalom Szenario einer *Technical Challange* verwendet. Dabei stehen drei Hindernisse in Form einer Landmarke in einer Reihe auf dem Spielfeld. Ein Hindernis befindet sich in der Mitte des Feldes, die beiden anderen jeweils zwischen Strafraum und Mittelpunkt. Der symmetrische Aufbau ist dabei fest, d.h. die Hindernisse können nicht an eine andere Position verschoben werden. Mit dem Hindernisroboter des Simulators (vgl. 3.7) kann ein viertes, frei positionierbares Hindernis dargestellt werden. Nur durch die Wahl dieses Szenarios hat man zusätzliche Hindernisse auf dem Feld, da im Simulator zurzeit nur ein verschiebbarer Hindernisroboter verfügbar ist. Dieser kann auch während der Testläufe bewegt werden.

Der eigene Roboter startet mit beliebiger Ausrichtung im Strafraum des blauen Tors.

6.1.3.2 Beobachtungen

Bei Testläufen ohne den Hindernisroboter wurde immer das Ziel ohne Kollision erreicht, da es sich um einen recht einfachen Test handelt. Oft war schon das Ausweichen vor dem ersten Hindernis ausreichend, um ohne weitere Blockaden auf geradem Weg zum gegenüberliegenden Strafraum gelangen zu können.

Bei den folgenden Tests wurde der Hindernisroboter so aufgestellt, dass er nach dem Überwinden des ersten Hindernisses den geraden Weg zum Ziel versperrt. Bei diesem Testfall konnte man eine Reaktion auf die seitlich liegenden Hindernisse beobachten. Diese lagen zunächst im Blickfelds des Roboters und sind im Modell entsprechend seiner Bewegungen verschoben worden, so dass er über die Kenntnis von Hindernissen verfügte, die sich nicht innerhalb seines aktuellen Blickfeld befanden.

6.1.4 Testläufe auf dem Spielfeld

6.1.4.1 Setup

Für die Tests auf dem Spielfeld wurden drei Hindernisse verwendet (vgl. Abbildung 6.2). Der Roboter startet auf der Grundlinie des blauen Tors mit Ausrichtung zum Mittelkreis. Zum Einsatz kommt das zuvor beschriebene Testverhalten, welches das kollisionsfreie Erreichen des gelbe Tors zum Ziel hat (vgl. 6.1.2). Die unterschiedlichen Farben der Säulen haben für den Roboter keine Bedeutung, da beide Farben (magenta, cyan) bei der verwendeten Farbtabelle nicht eigens definiert wurden (vgl. 5.1.1). Sie fallen somit in die Restklasse der nicht segmentierten Bildpunkte und gelten daher als Hindernisse.

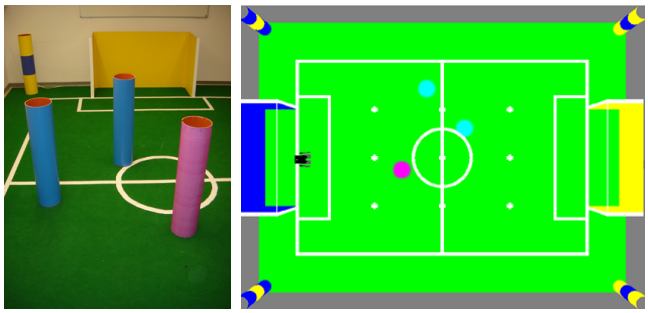

Abbildung 6.2: Spielfeldaufbau für Tests zur Hindernisvermeidung

Erwartungen Nach dem Start des Roboters soll dieser auf seinem Weg zum gelben Tor zunächst frontal auf das magentafarbene Hindernis stoßen. Zum Zeitpunkt der Entscheidung, zu welcher Seite auszuweichen ist, dürften sich keine Hindernisse zu den Seiten in Modell befinden. Daher wird mit einem Ausweichmanöver nach links gerechnet, welches das Standardverhalten für diesen Fall darstellt. Der Roboter sollte seinen Weg zum Tor fortsetzen und frontal auf das blaue Hindernis am Mittelkreis treffen. Beabsichtigt ist, ein Ausweichverhalten zur rechten Seite beobachten zu können, welches durch das Vorhandensein des anderen blauen Hindernisses im Modell ausgelöst wurde.

Der Testaufbau soll eine Reaktion auf Hindernisse zeigen, die sich nur im Modell befinden und nicht im Blickfeld des Roboters liegen.

6.1.4.2 Beobachtungen

Anhand einzelner Bilder aus einem bei den Testläufen aufgenommenen Video soll die Reaktion auf Hindernisse verdeutlicht werden (Abbildungen 6.3, 6.4, 6.6). Im

ersten Bild ist der Roboter nach dem Start aus dem Strafraum des blauen Tores zu sehen.

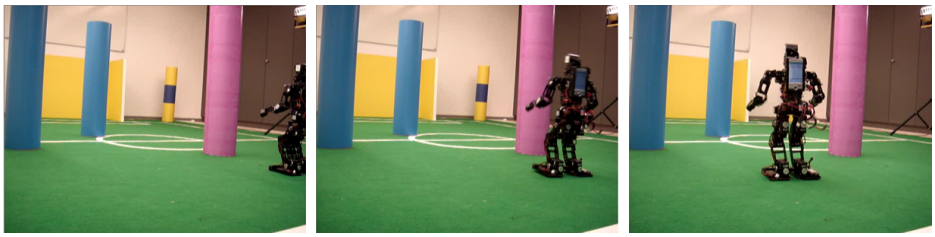

Abbildung 6.3: Testlauf der Hindernisvermeidung: Sequenz 1 (Bild 1-3)

Der HR 18 bewegt sich geradeaus auf die magentafarbene Röhre zu und stoppt kurz, als er dieser zu nahe kommt. Er beginnt sodann mit dem Ausweichen zur linken Seite (Bild 2), da in der Modellierung keine Hindernisse zu den Seiten verzeichnet sind. Sobald nach vorne wieder genügend Freiraum besteht, wird die Seitwärtsbewegung beendet (Bild 3) und mit dem Vorwärtslaufen begonnen, bis die mittlere Säule erreicht wird (Bild 4).

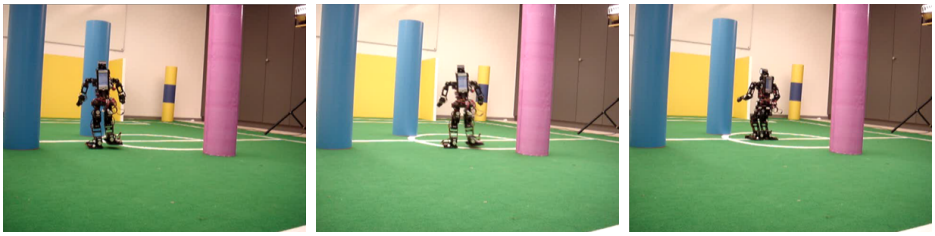

Abbildung 6.4: Testlauf der Hindernisvermeidung: Sequenz 2 (Bild 4-6)

Dabei wurde das linke Hindernis wahrgenommen und in das Modell aufgenommen. Durch die Aktualisierung des Modells mit Hilfe der Odometriedaten hat sich das Hindernis in der Radaransicht zur Seite des Roboters bewegt, so dass eine Übereinstimmung mit der Realität zu beobachten ist. Abbildung 6.5 zeigt das Hindernismodell in etwa zum Zeitpunkt der Aufnahme von Bild 4. Dort sind sowohl das Hindernis vor dem Roboter sowie das links gelegene abgebildet. Die restlichen Sektoren besitzen einen höheren Abstandswert und sind daher nicht zu sehen. Insbesondere sind keine Abstandswerte zur magentafarbenen Säule mehr im Modell vorhanden, da diese bereits herausgealtert sind. Die Distanzwerte der Sektoren sind je nach Verlässlichkeitswert (0-100 %) entsprechend von rot in einem Übergang zu gelb gehalten. Das zur Seite gelegene Hindernis befindet sich schon

etwas länger im Modell und ist daher rötlich markiert, das zur Front hingegen durch die Bestätigungen der Hinderniserkennung eher gelb. Die repräsentativen Punkte der Sektoren sind rot dargestellt. Anhand des Hindernismodells traf der Roboter die Entscheidung ein Ausweichmanöver einzuleiten. Dabei wurde anhand des Modells die rechte Seite bestimmt, da zur linken eine kleinere Freiraumdistanz bestand.

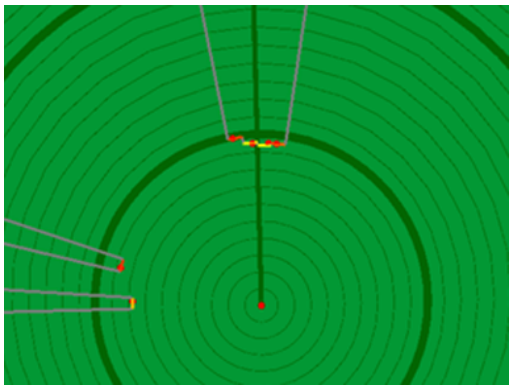

Abbildung 6.5: Hindernismodell während der Hindernisvermeidung

In Bild 5 ist der Weg nach vorne wieder frei, so dass der Roboter seinen Weg zum gelben Tor fortsetzen kann. Auf der Mittellinie angelangt, wird durch eine Drehung auf der Stelle die Ausrichtung zum Tor korrigiert (Bild 6), da es bei der Seitwärtsbewegung zu einer Kursabweichung kam. Die neue Ausrichtung machte ein erneutes Ausweichen erforderlich. Da sich das linke Hindernis noch in der Abstandskarte befand, wurde wieder nach rechts ausgewichen (Bild 7). Anschließend läuft der Roboter geradeaus zum Tor (Bild 8) und erreicht nach einer kleinen Neuausrichtung den Strafraum und somit sein Ziel (Bild 9).

Abbildung 6.6: Testlauf der Hindernisvermeidung: Sequenz 3 (Bild 7-9)

Um das erwartete Verhalten auf dem Spielfeld beobachten zu können, waren etliche Testläufe notwendig. Zunächst mussten die vielen, durch Simulatortests gefundenen Parameter, der Hinderniserkennung und -modellierung sowie die Schwellwerte des Testverhaltens angepasst werden. Der Simulator hat das Ermitteln von Grundwerten sehr beschleunigt, die Bestimmung der exakten Parameter für die reale Verwendung war allerdings nicht möglich, denn der Simulator stellt nur ein vereinfachtes Abbild des Roboterumfeldes dar. Die Technik der realen Roboter bereitet durch die Komplexität der Systeme Probleme, mit denen gegenüber dem Simulator ein deutlicher Mehraufwand einhergeht. Dazu gehören zum Beispiel Verbindungsschwierigkeiten über WLAN, Akkuwechsel, Probleme mit den Kameratreibern, Abstürze der Pocket PCs und Wackelkontakte zwischen PPC und Roboter. Hinzu kommen Kalibrierungen für Gelenke und Kameras der einzelnen Roboter, da diese nie exakt gleich gebaut werden können. Gerade deshalb und wegen der unzureichend simulierbaren Störeinflüsse der Umwelt (Wackeln des Roboters, Rauschen in den Kamerabildern, Lichtverhältnisse, ...) sind reale Tests unabdingbar.

Von der Anpassung der Parameter war hauptsächlich die Hindernismodellierung betroffen, denn die Erkennung wurde bereits mit realen Roboterbildern getestet. Dazu gehörten im einzelnen Parameter, die die Alterung und die Aktualisierung des Modells beeinflussen sowie der Toleranzbereich für die Bestätigung des Distanzwertes eines Sektors (vgl. 5.1.2.2). Außerdem mussten die Winkelbereiche angepasst werden, die die Distanzen mehrerer Sektoren für eine Abstandsbestimmung nach vorne und zu den Seiten zusammenfassen. Dazu wurden auch andere Testszenarien verwendet als der detailliert beschriebene Aufbau, mit dem nun die eigentlichen Testläufe stattfanden.

Bei manchen Anläufen gelangte der Roboter durch Abweichungen beim Geradeauslaufen ohne oder mit anderen Ausweichmanövern in den Strafraum des gelben Tores. Dabei wurde nur selten ein Hindernis berührt. Durch die Korrekturen seiner Laufrichtung kann es zu längeren Verweildauern kommen, die dazu führen, dass ein Hindernis zur Seite aus dem Modell entfernt wird. Trifft der Roboter in einer solchen Situation frontal auf ein Hindernis, so kann es beim Ausweichen zu den Seiten zu einer Kollision kommen. Es kam nie zu Zusammenstößen beim Vorwärtslaufen, höchstens zu leichten Berührungen mit den Armen des Roboters.

6.2 Bewertung der Spielererkennung

Die Module zur Spielererkennung erforderten schon während ihrer Entwicklung ausgiebige Tests. Durch die Einbeziehung möglichst vieler Testsituationen mit Hil-

fe des Simulators und vom Roboter aufgezeichneter Bilder konnte eine robuste Arbeitsweise der Erkennung erzielt werden. Denn die Erkenntnisse der Analysen führten unmittelbar zur Festlegung der Parameter. Dazu gehörten beispielsweise Mindestlängen für die zu erkennenden Segmente oder das Maß der Akzeptanz für Störpixel. Gerade bei der Torwarterkennung, aber auch bei der Erkennung über *Teammarker,* sind einige Problemfälle im Kapitel Realisierung aufgeführt (vgl. 5.2.1, 5.2.2).

Die Torwarterkennung kam bei der RoboCup Weltmeisterschaft 2006 in Bremen zum Einsatz. Das dort eingesetzte Verhalten für den Strafstoßschützen der *Penalty*-Disziplin verwendete den ermittelte Winkel zum Torwart bei der Berechnung eines optimalen Schusswinkels. Die Erkennung des Torwartes funktionierte dabei erstaunlich zuverlässig.

6.3 Performance

Wie bereits mehrfach erwähnt, muss wegen der verwendeten Recheneinheit bei der Programmierung stark auf die Effizienz bezüglich der Laufzeiten geachtet werden. Daher wurden alle neu entwickelten Module auf ihre Performance untersucht. Für die Zeitmessungen kam das vom *RoboFrame* zur Verfügung gestellte *Chronometer* zum Einsatz. Da die Module beim Framework registriert sind, um eine vollständige Anwendung zu bilden, genügen nur wenige Zeilen zusätzlichen Codes, um die Messungen an den richtigen Stellen starten und stoppen zu können. Die Tests wurden am realen Roboter auf dem Spielfeld durchgeführt, um die tatsächlichen Laufzeiten auf dem Pocket PC zu ermitteln. Über eine WLAN Verbindung konnten die Zeiten im Chronometer, welches Teil von *RoboCup06GUI* ist, abgelesen werden.

Zunächst wurden die Module zur Erkennung der Spieler (*GoalieInGoalPerceptor*, *OpponentPerceptor*) untersucht. Da der *GoalieInGoalPerceptor* eine Unterklasse der Torerkennung darstellt, wurde der Teil zur Erkennung der Tores von den Messungen ausgeschlossen. Beide Komponenten liegen mit ca. 20 Millisekunden durchschnittlicher Laufzeit und keinen Ausreißern über 41 ms im unteren Mittelfeld aller *Perceptoren.*

Die Hinderniserkennung (*ObstaclesPerceptor*) benötigt durchschnittlich 33 ms Prozessorzeit pro Aufruf. Es kann zu Spitzenwerten bis 106 ms kommen. Weitere Untersuchungen haben ergeben, dass die Projektionen von Bildpunkten in das Roboterkoordinatensystem 50 % der Zeit verbrauchen. Dies ist nicht weiter verwunderlich, denn in der Regel wird für jede vertikale *Scanline* mindestens eine Projektion benötigt. *Scanlines,* die vollständig aus einer Bodenfarbe bestehen, kommen ohne Projektionen aus, sind aber eher selten. In den anderen Fällen muss

zumindest der erste Punkt nach einer Bodenfarbe in Roboterkoordinaten umgerechnet werden. Nur so lässt sich entscheiden, ob es sich um ein zu beachtendes Hindernis handelt. Die Laufzeit des *ObstaclesPerceptors* liegt im Vergleich zu den anderen *Perceptoren* zwar im oberen Segment, fällt aber noch in den akzeptierten Normalbereich.

Die Hindernismodellierung (*ObstacleModelModule*) wurden ebenfalls auf ihre Effizienz hin überprüft. Die dabei gemessene mittlere Laufzeit von zwei Millisekunden fiel gegen alle Erwartungen sehr gering aus. Selbst in weiteren Tests mit besonders vielen Hindernissen im Modell wurde bei über 1000 Messungen ein Maximalwert von 9 ms nicht überschritten.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Das Fachgebiet Simulation und Systemoptimierung nimmt mit seinen humanoiden autonomen Robotern regelmäßig an den Fußball-Wettbewerben des RoboCups teil. Bei den verschiedenen Aufgabenstellungen hat sich gezeigt, dass eine Berücksichtigung von Hindernissen einen großen Vorteil bringt.

Im Rahmen dieser Diplomarbeit wurde eine allgemeine Hinderniserkennung, sowie eine Spielererkennung entwickelt, da eine Unterscheidung aufgrund unterschiedlicher Interaktionsmöglichkeiten sinnvoll sein kann (vgl. 4.1). Die verschiedenen neuen Softwarekomponenten, zu denen auch eine aufwändige Hindernismodellierung gehört, stellen eine Erweiterung der eingesetzten Steuerungssoftware *RoboCup06App* dar.

Es konnte gezeigt werden, dass die Torwarterkennung - ein Spezialfall der Spielererkennung - sehr robust arbeitet und hervorragend dazu geeignet ist, einen optimalen Schusswinkel zum Tor zu berechnen. Durch den Entwurf und die Implementierung eines Roboterverhaltens war es möglich eine Hindernisvermeidung zu demonstrieren. Mit Hilfe der Umweltkarte der Modellierung konnten bei Ausweichmanövern kürzlich gesehene Hindernisse beachtet werden, die sich nicht im aktuellen Blickfeld befanden.

7.2 Ausblick

Durch den zeitlich begrenzten Rahmen war eine Weiterverfolgung oder gar Umsetzung aller Ideen nicht möglich. Besonders davon betroffen ist der Bereich der Spielererkennung und -modellierung, welcher weiterhin ein großes Verbesserungspotential birgt. Eine Fortführung der Entwicklung der Torwarterkennung erscheint indes nicht sinnvoll, da diese bereits zuverlässig funktioniert. Es ist zudem fraglich, woran eine solche Verbesserung anknüpfen könnte. Ausgenommen sind dabei

selbstverständlich Möglichkeiten, die sich aus einer Erhöhung der zur Verfügung stehenden Ressourcen ergeben.

Von der Verwendung einer leistungsfähigeren Steuerungseinheit profitiert besonders die Bildverarbeitung. Die Verarbeitung einer größeren Anzahl von Bildern sowie eine größere Bildauflösung führen zu einer verbesserten Erkennung. Das *Scanlineraster* könnte optimiert werden, indem in Bildbereichen mit potentiell höherem Informationsgehalt eine engere Rasterung eingesetzt wird. Bekanntermaßen ist dies der Bereich um den Horizont, durch die besondere Bedeutung der Bodenpunkte speziell der Bereich unterhalb des Horizonts. Die erhöhte Auflösung der Kamerabilder und das optimierte *Scanlineraster* ermöglichen eine gesteigerte Genauigkeit bei der Umrechnung der Bildpunkte in das Roboterkoordinatensystem. Ebenfalls verbessert würde die Zuverlässigkeit bei der Erkennung von kleineren und weiter entfernten Objekten (z.B. Teammarker).

Die Spielererkennung könnte nach der Identifikation der Teammarker im Bild eine Suche nach Bodenpunkten durchführen, so dass die Möglichkeit einer Positionsbestimmung der Spieler besteht. Dies steht allerdings in Abhängigkeit zum Problem der Erfassung der Roboterbeine, welches möglicherweise durch eine höhere Auflösung zu kompensieren wäre (vgl. 5.2.2). Zuvor sollte allerdings untersucht werden, ob das Hindernismodell die gesuchten Bodenpunkte der Spieler liefern kann. Der zu einem Spieler ermittelte Winkel identifiziert den entsprechenden Sektor im Hindernismodell. Dessen Distanzwert sollte zusammen mit dem Winkel die Position des gesuchten Spielers ergeben.

Eine weitere Möglichkeit besteht darin, mit der Kenntnis der Größe der Teammarker eine Entfernung abzuschätzen. Dies war bisher durch die recht grobe Erfassung der Marker nicht realisierbar. Neben der Position ist auch die Ausrichtung der Spieler von Interesse. Diese aus den Bildern zu erkennen gestaltet sich bisher äußerst schwierig. Ursachen hierfür sind zum einen in der Verdeckung der Marker durch Roboterarme und zum anderen in der Größe der Marker im Bild zu finden. Mit der verbesserten Bildverarbeitung könnte die horizontale Ausdehnung eines Markers dazu verwendet werden zumindest eine grobe Ausrichtung zu ermitteln. Sinnvoller erscheint indes eine Bestimmung mit Hilfe einer erweiterten Modellierung. Wird ein Spieler an zwei Positionen erfasst, lässt dies möglicherweise Rückschlüsse zu. Voraussetzung ist ein Vorwärtslaufen der Roboter. Weiterhin könnte eine Kombination aus Erkennung und Modellierung zu verbesserten Ergebnissen, auch bei omnidirektionalen Bewegungen, führen.

Ein neues Feld für Weiterentwicklungen stellt die Sensorfusion dar. Hierfür bietet sich die Kombination von Kopf- und Bauchkamerabildern bei einem Roboter oder aber auch die Fusion von über *Teammessages* kommunizierten Daten an. Vor-

aussetzung für die Kombination der Daten mehrerer Roboter ist eine möglichst genaue Selbstlokalisation, denn die Spielfeldkoordinaten bilden das gemeinsame Bezugssystem. Beispielsweise würde sich eine Kreuzpeilung zur Positionsbestimmung der Gegner eignen. Wird ein Torwart im eigenen Team eingesetzt, ist dieser in der Lage, durch seine eingeschränkten Bewegungen das gesamte Spielfeld recht gut zu observieren. Er könnte den anderen Feldspielern so als Informationsquelle dienen. Die Roboter könnten Daten des Hindernismodells des Mitspielers mit berücksichtigen oder gar ein gemeinsames absolutes Hindernismodell verwenden. Dazu wäre ein Potentialfeld geeignet, welches das Spielfeld in diskrete Flächen einteilt (z.B. Quadrate). Für jedes Element bestimmt ein Wahrscheinlichkeitswert, ob es durch ein Hindernis okkupiert ist. Die Hinderniserkennung könnte dabei weiter verwendet werden. Die *Hindernispercepte* müssten nur in das neue Modell einfließen. Die Wahrscheinlichkeit eines Feldes erhöht sich, sobald ein Punkt aus einem *Percept* in dieses hineinfällt. Über eine Standardnormalverteilung werden auch die Nachbarfelder in ihrem Wert angehoben. Das Modell sollte ebenfalls einem Alterungsprozess unterliegen oder versuchen Negativinformationen über Hindernisse zu verwenden.

8 Literaturverzeichnis

[1] Bacher, Johann: *Clusteranalyse.* 2. Auflage. München : Oldenbourg, 1996

[2] Dassler, Marc: *Gegnererkennung und -modellierung für Passspiel in der 4-Legged RoboCup League*, TU Darmstadt, FB Informatik, Diplomarbeit, 2005

[3] Foley, James D. ; van Dam, Andries ; Feiner, Steven K. ; Hughes, John F.: *Computer Graphics: Principles and practice.* Adisson-Wesley Publishing Company , 2006

[4] Friedmann, M. ; Kiener, J. ; Kratz, R. ; Ludwig, T. ; Petters, S. ; Stelzer, M. ; Stryk, O. von ; Thomas, D.: Darmstadt Dribblers 2005: Humanoid Robot (Team Description Paper) / Technische Universität Darmstadt. 2005. – Forschungsbericht. – 10 pages

[5] H. Asada and J.J. Slotin: *Robot Analysis and Control.* New York : Wiley, 1986

[6] Hoffmann, Jan ; Jüngel, Matthias ; Lötzsch, Martin: A Vision Based System for Goal-Directed Obstacle Avoidance. In: *RoboCup 2004: Robot Soccer World Cup VIII.* Lisbon, Portugal : Springer, 2005 (Lecture Notes in Artificial Intelligence)

[7] Jähne, B.: *Digital image processing.* 3. Auflage. Berlin : Springer-Verlag, 1995

[8] Kondo Kagaku Co. Ltd. (Hrsg.): *KHR 1 - Hardware Manual.* Kondo Kagaku Co. Ltd., 2004. `http://www.kondo-robot.com/pdf/KHR-1HardwareManual_English.pdf`

[9] Kondo Kagaku Co. Ltd. (Hrsg.): *RCB 1 - Software Manual.* Kondo Kagaku Co. Ltd., 2004. `http://www.kondo-robot.com/pdf/SoftwareManualEnglish.pdf`

[10] Lenser, S. ; Veloso, M.: *Visual sonar: Fast Obstacle Avoidance using Monocular Vision.* citeseer.ist.psu.edu/lenser03visual.html. Version: 2003

[11] Lötzsch, Martin: *XABSL - A Behavior Engineering System for Autonomous Agents*, Humboldt-Universität zu Berlin, Diplomarbeit, 2004. www.martin-loetzsch.de/papers/diploma-thesis.pdf

[12] Lötzsch, Martin ; Bach, Joscha ; Burkhard, Hans-Dieter ; Jüngel, Matthias: Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. In: *RoboCup 2003: Robot Soccer World Cup VII* Bd. 3020. Padova, Italy : Springer, 2004 (Lecture Notes in Artificial Intelligence), S. 114–124

[13] Petters, Sebastian ; Thomas, Dirk: *RoboFrame - Softwareframework für mobile autonome Robotersysteme*, TU Darmstadt, FB Informatik, Diplomarbeit, 2005

[14] Stentz, Anthony J.: *The Navlab system for mobile robot navigation.* Pittsburgh, PA, USA, Diss., 1990