Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Simulation und Systemoptimierung

Ein globales Sichtsystem zur Unterstützung der Entwicklung autonomer Fußballroboter

A Global Vision System to Support Development in Autonomous Robotic Soccer

Diplomarbeit von Ronnie Brunn und Michael Kunz

Aufgabensteller:	Prof. Oskar von Stryk
Betreuer:	Dipl.-Tech. Math. Maximilian Stelzer
Abgabeterim:	25. Januar 2005

Erklärung zur Diplomarbeit

Hiermit versichern wir, dass die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt wurde. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ronnie Brunn
Darmstadt, den 25. Januar 2005

Michael Kunz
Darmstadt, den 25. Januar 2005

Abstract

The Simulation and Systems Optimization Group of the Technische Universität Darmstadt participates in *RoboCup* conferences and championships. The *Darmstadt Dribbling Dackels* compete in the 4-legged league, which is played with Sony-Aibo robots. By utilizing the soccer game as an enviroment to test the skills the *RoboCup* tries to accelerate the progress in autonomous robotics.

This thesis describes a system to acquire a global worldmodel utilizing a ceiling mounted camera. The global view of the enviroment, which is not allowed during the actual competition, shall be used for development and evaluation purposes. In detail algorithms to compensate distortions of the camera image and to detect and identify the objects on the field have been developed. As a proof of concept the system was used to evaluate some of the approaches on autonomous selflocalization of the robots.

Zusammenfassung

Das Fachgebiet Simulation und Systemoptimierung nimmt im Namen der Technischen Universität Darmstadt am *RoboCup* teil. Die *Darmstadt Dribbling Dackels* treten dabei in der Vierbeiner-Liga an, in der mit Aibo Robotern der Firma Sony gespielt wird. Am Anwendungsbeispiel des Fußballspielens sollen im *RoboCup* Fortschritte auf allen Gebieten der autonomen Robotik erzielt werden, von Bildverarbeitung und Weltmodellierung über die Verhaltensplanung bis hin zur Bewegungssteuerung.

Diese Diplomarbeit beschreibt ein Deckenkamerasystem zur Erfassung eines globalen Weltbildes, welches bei der Entwicklung und Bewertung der Softwaremodule für die autonomen Roboter eingesetzt werden soll. Im Einzelnen wurden hierfür Algorithmen zur Kompensation der Verzerrungen des Kamerabildes sowie zur Erkennung und Modellierung der Objekte auf dem Spielfeld entwickelt. Als Beweis der Einsetzbarkeit des Systems wurde eine Beispielanwendung zur Bewertung der Selbstlokalisierung der Roboter erstellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Vorwort

1.1 Ziel dieser Diplomarbeit

Ziel dieser Arbeit ist es ein Deckenkamerasystem zur Erfassung eines gobalen Weltbildes zu erstellen. Im Einzelnen umfasst dies die Erfassung der Position und Ausrichtung der Roboter auf dem Spielfeld, die Erkennung der Position und Geschwindigkeit des Balls sowie eine zeitliche Synchronisierung dieser Daten mit den Debugdaten der Roboter. Das System soll in Echtzeit unter möglichst spielnahen Bedingungen einsetzbar sein und mehrere Entwickler gleichzeitig mit den erfassten Daten versorgen. Als Anwendungsbeispiel dient eine einfache objektive Beurteilung verschiedener Ansätze zur Selbstlokalisierung, die an Hand der so gewonnenen Daten zur Demonstration des Konzepts durchgeführt wird.

1.2 Aufbau der Arbeit

Inhalte der einzelnen Kapitel im Überblick:

Kapitel 2
Beschreibung der Umgebungsbedingungen in der Sony-Vierbeiner-Liga des *RoboCup*, soweit sie für das in dieser Arbeit vorgestellte Deckenkamerasystem relevant sind.

Kapitel 3
Kurzer Überblick über bereits existierende Ansätze.

Kapitel 4
Methoden zur Kompensation der Verzerrungen des Kamerabildes durch optische Effekte und Perspektive.

Kapitel 5
Beschreibung der Bildverarbeitung und -auswertung. Algorithmen zur Farbklassifizierung sowie Suche, Identifikation und Modellierung der einzelnen Objekte,

d.h. Bestimmung ihrer Position, Ausrichtung und Geschwindigkeit. Messreihen zur Bestimmung der Stabilität und Güte der Objekterkennung.

Kapitel 6
Anbindung des Deckenkamerasystems an die Entwicklungsumgebung des *GermanTeam*, d.h. das Programm *RobotControl* zur PC-gestützten Auswertung der Debugdaten des Roboters. Verfahren zur Zeitsynchronisierung der Daten von Robotern und Deckenkamera.

Kapitel 7
Bestimmung der Güte einiger Selbstlokalisierungsalgorithmen und der Stabilität des Ballerkenners anhand des durch das Deckenkamerasystems gewonnenen globalen Weltbildes.

Kapitel 8
Zusammenfassung der Ergebnisse dieser Diplomarbeit sowie Ausblick auf zukünftige Einsatzmöglichkeiten des Systems sowie Ansätze zu dessen Erweiterung.

Anhang A
Arbeitsschritte zur Montage, Ausrichtung und Kalibrierung des Deckenkamerasystems.

Anhang B
Bedienung der Software zur Auswertung der Deckenkamerabilder, sowie kurzer Überblick über die Implementierung.

Anhang C
Bedienung der in *RobotControl* integrierten Client-Schnittstelle für zum Deckenkamerasystem und Arbeitsschritte zur Aufzeichung zeitsynchronisierter Logdateien.

1.3 Urheberschaft der einzelnen Kapitel

Da diese Diplomarbeit in einem kleinen Team bearbeitet wurde, gilt es aufzuzeigen, wer für welche Teile der Arbeit verantwortlich zeichnet. Ronnie Brunn hat die Kapitel 2, 4 bis 4.1, 5 bis 5.3, 6, 8 und Anhang A, Michael Kunz das Kapitel 3, die Abschnitte 4.2, 5.4, das Kapitel 7, die Anhänge B und C verfasst.

Kapitel 2

Rahmenbedingungen

Im Folgenden sollen die Bedingungen des Umfeldes für diese Diplomarbeit genauer erläutert werden.

2.1 RoboCup

Das in dieser Arbeit beschriebene Kamerasystem soll zu Entwicklungszwecken im Rahmen der Sony-Vierbeiner-Liga im *RoboCup* eingesetzt werden. Die Rahmenbedingungen des *RoboCup* werden im Folgenden kurz erläutert, insoweit sie in diesem Zusammenhang relevant sind.

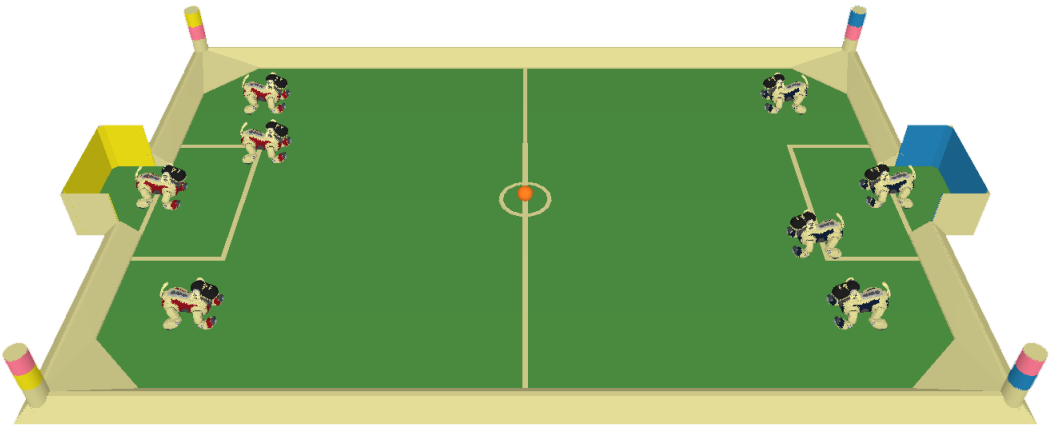

Abbildung 2.1: Spielfeld

2.1.1 Feld

Die Sony-Vierbeiner-Liga im *RoboCup* spielt nach den offiziellen Regeln für das Jahr 2004 auf einem Feld (Abb. 2.1) von 4,20×2,70 m. Beide Torräume ragen darüber noch einmal 35 cm hinaus, so dass die Gesamtfläche die von der Deckenkamera abgedeckt werden muss, um die Roboter und den Ball in allen Spielsituationen zu erfassen, ca. 5×2,80 m beträgt. Der Boden ist mit grünem Teppich ausgelegt, auf dem Mittellinie, Mittelkreis, Strafraumlinien und Torlinie mit weißem Klebeband abgeklebt werden. Darüber hinaus verfügt das Feld über eine weiße Bande. Die Tore sind gelb und blau eingefärbt, des Weiteren werden Landmarken in den Ecken mit Farbkombinationen aus gelb, blau und pink eingesetzt. Der Ball ist orange. Um Fehlerkennungen durch die Roboter zu vermeiden, dürfen jegliche zusätzlichen Marker für die Deckenkamera die genannten Farben möglichst nicht enthalten oder nur in einer nicht mit den Objekten auf dem Spielfeld verwechselbaren Kombination.

2.1.2 Roboter

Gespielt wird mit Sony Aibo Robotern der Produktreihen ERS-210, ERS-210A sowie ERS-7 (Abb. 2.2) in den Farben Anthrazit und Weiß. Diese tragen zur Unterscheidung der Mannschaften Marker in Dunkelblau und Rot. Aus dem Sichtwinkel einer Deckenkamera sind die Trikots der Roboter sowie für den Spielbetrieb angebrachte Nummern nicht erkennbar. Selbst eine Erkennung der Ausrichtung des Roboters direkt aus dem Kamerabild erscheint wenig aussichtsreich. Das Anbringen zusätzlicher Marker ist im regulären Spielbetrieb untersagt, für die Entwicklung aber insoweit unbedenklich wie die Roboter dadurch nicht behindert oder abgelenkt werden.

Abbildung 2.2: ERS7 und ERS210(A)

2.1.3 Regeln

Die Roboter spielen autonom ohne Unterstützung durch externe Computer. Lediglich die Kommandos des Schiedsrichters werden ihnen über WLAN mitgeteilt. Eine Kommunikation der Roboter eines Teams untereinander ist erlaubt. Die hier vorgestellte Software zur Erstellung eines globalen Weltbildes über eine Deckenkamera ist also im offiziellen Spielbetrieb der Liga weder vorgesehen noch erlaubt, sondern dient einzig und allein der Unterstützung der Entwicklung im Vorfeld. Dies bringt den Vorteil mit sich, dass das verwendete Kamerasystem nicht mobil sein muss, um es auf Wettbewerben einzusetzen, und Montage sowie initiale Kalibrierung nur einmalig im heimischen Entwicklungslabor erfolgen müssen.

2.2 GermanTeam - Architektur

Das *GermanTeam* ist ein Zusammenschluss aus derzeit vier Universitäten in Deutschland, die auf internationaler Ebene im *RoboCup* gemeinsam als *GermanTeam* antreten. Beteiligt sind die Humboldt Universität zu Berlin, die Universität Bremen, die Technische Universität Darmstadt sowie die Universität Dortmund. National und in anderen Wettbewerben wie z.B. den *RoboCup German Open* treten die Teams auch gegeneinander an. Um die standortübergreifende Entwicklung an verschiedene Lösungen der Teilprobleme zu unterstützen, entwickelte das *GermanTeam* eine modulare Softwarearchitektur. Hierbei können für eine spezielle Aufgabe, z.B. die Selbstlokalisation des Roboters auf dem Feld oder die Erkennung des Balls, verschiedene Lösungen entwickelt werden. Die Schnittstellen zum Rest der Software sind hierbei festgelegt, so dass zwischen verschiedenen Versionen eines Moduls jederzeit schnell gewechselt werden kann. Nach den *German Open* entscheidet sich das *GermanTeam* für die vielversprechendsten Ansätze und stellt aus diesen den Modulsatz zusammen, der bei der Weltmeisterschaft antritt.

Beim Vergleich verschiedener Lösungsansätze für ein Modul fällt ein objektiver Vergleich in vielen Fällen schwer. Insbesondere wäre eine Beurteilung der Module unter Spielbedingungen wünschenswert, nicht anhand künstlich generierter Testfälle, die die Dynamik der Umgebungssituation im laufenden Spiel nicht wiedergeben können. Ein Modul, das in einem künstlich erstellten Testparcours im Labor gut abschneidet, muss schließlich noch lange nicht in Spielsituationen die sinnvollste Lösung darstellen, in denen diverse Seiteneffekte und Wechselwirkungen auftreten können.

2.3 Globales Weltbild zu Debugzwecken

Um objektive Vergleiche verschiedener Algorithmen oder Parametersätze in Spielsituationen zu ermöglichen, benötigt man ein globales Weltbild, das als Vergleichsbasis herangezogen werden kann, um die Debugdaten vom Roboter zu beurteilen. Testet man z.B. verschiedene Ansätze zur Selbstlokalisierung und lässt sich zu diesem Zweck

von einem Roboter auf dem Spielfeld die Position schicken, an der er sich nach seiner Berechnung befindet, ist es, sobald er sich in Bewegung befindet, nicht mehr möglich, diese durch manuelles Nachmessen zu überprüfen. Nur durch möglichst viele Vergleichsdaten zwischen den Roboterdaten und der Realität kann der Entwickler beurteilen, welcher Algorithmus oder welcher Parametersatz unter Spielbedingungen am besten abschneidet.

Steht ein solches System zur Verfügung, das die realen Positionen von Robotern und Ball erfasst und zur Unterstützung der Entwicklung bereitstellt, kann damit aber noch weit mehr erreicht werden als eine bloße Beurteilung bestehender Lösungen. Eine automatisierte Optimierung z.B. verschiedener Parametersätze für die Laufbewegung des Roboters ist ebenso denkbar wie das Kalibrieren von Algorithmen, die mit den Unsicherheiten in der Sensorik des Roboters rechnen und deren genaues Ausmaß bisher nur grob von Hand abgeschätzt wurde. Die Weltmodellierung auf dem Roboter sowie die Verhaltenssteuerung könnten erheblich davon profitieren, wenn der Roboter in der Lage wäre, zumindest die Unsicherheiten seiner Berechnungen zu erkennen. Beispielsweise würde das Erfassen der realen Ballposition und Ballgeschwindigkeit die Kalibrierung der Kovarianzmatrizen für den Kalmanfilter in der Ballmodellierung erlauben.

2.4 Deckenkamerasystem zur Erfassung des globalen Weltbildes

Abbildung 2.3: Sony DFW-SX900 Firewirekamera

Im Rahmen dieser Diplomarbeit wurde ein neues System mit einer hochauflösenden Kamera entwickelt, die dank weitwinkligem Objektiv das komplette Spielfeld inklusive der Torräume überblicken kann. Dabei kommt eine SONY DFW-SX900 (Abb. 2.3) zum Einsatz, eine 1/2 Zoll CCD Kamera mit FireWire-Anschluß, die mit

einem hochwertigen 3,6 mm C-Mount Objektiv ausgestattet wurde. Die Kamera wurde dem Fachgebiet Simulation und Systemoptimierung durch die Firma Allied Vision Technologies [1] zur Verfügung gestellt. Sie ist an der Decke des Labors montiert und blickt geradlinig nach unten auf das Spielfeld. Die Auswertung des Kamerabildes erfolgt bei einer Auflösung von 1280×960 Bildpunkten und einer Framerate von 7,5 fps in Echtzeit durch einen PC, der das berechnete globale Weltbild dann per Netzwerk an die Entwicklungssoftware auf den Client-PCs weiterverteilt. Um einen möglichst genauen Vergleich mit den Debugdaten von den Robotern innerhalb der Entwicklungssoftware auf den Client-Rechnern zu ermöglichen, wurde ein Verfahren zur Erfassung des zeitlichen Offsets zwischen den Uhren auf den Robotern sowie der Systemzeit des Deckenkamera-Rechners entwickelt.

Kapitel 3

Existierende Ansätze

In den unterschiedlichen Ligen des *RoboCups* wurden bereits verschiedene Systeme entwickelt, die mittles einer unter der Decke montierten Kamera den Ball und die Roboter auf dem Spielfeld erkennen. Allerdings ist deren Verwendung während des Wettbewerbs nur in der Smallsize-Liga erlaubt. Aus diesem Grund existieren dort die fortschrittlichsten Systeme. Aus der Sony-Liga selbst sind allerdings keine laufenden Deckenkamerasysteme bekannt.

3.1 Ballerkenner

Im *GermanTeam* wurde im Jahr 2004 eine sehr robuste Ballerkennung für den Roboter entwickelt [9]. Sie basiert darauf, vom ImageProcessor mit einem orangefarbenen Punkt im Bild aufgerufen zu werden und von diesem aus mit einem spinnennetzartigen Suchmuster nach den Grenzen des Balls zu suchen. Dabei wird unter anderem ein Verfahren eingesetzt, das für eine Farbe die Ähnlichkeit zur orangenen Farbe des Balls berechnet. Da dieser Ballerkenner mit wenigen Änderungen an die Erfordernisse des Deckenkamerasystems angepasst werden konnte, musste hierfür nicht auf die Erfahrungen anderer *RoboCup*-Teams zurückgegriffen werden.

3.2 Robotererkenner

An der HU-Berlin wurde zu Beginn der Aktivitäten in der Sony-Liga ein System zur Erfassung der Roboterpositionen über eine an Decke bzw. Wand befestigte Webcam programmiert. Allerdings war dieses System recht ungenau und auf Grund der gewählten Farbmarker auf den Robotern recht unzuverlässig in der Erkennung. Es wurde im Rahmen der Entwicklungen im *GermanTeam* nicht mehr praktisch eingesetzt und mit dem Wechsel auf ein neues Framework endgültig verworfen.

In der Smallsize-Liga des *RoboCup* werden schon seit längerem viele verschiedene Methoden zur Erkennung von Robotern auf dem Spielfeld mittels einer darüber

befestigten Kamera verwendet. Viele davon beruhen auf der Kombination mehrerer verschiedener Farben, die in dieser Liga erlaubt sind. Hierzu ein Zitat aus „LAW 4“ der Regeln der Smallsize-Liga [7]:

> Colours and Markers
>
> Before a game, each of the two teams has a colour assigned, namely yellow or blue. Each team must be able to use either yellow or blue markers. Circular markers of the assigned colour must be mounted on top of the robots. The centre of the marker must be located in the visual centre of the robot when viewed from above. The markers must have a diameter of 50 mm.
>
> Robots may use black and white colouring without restriction. Robots may also use light green, light pink and cyan markers where the colours are defined by cardboard sheets distributed in advance by the local organizing committee.

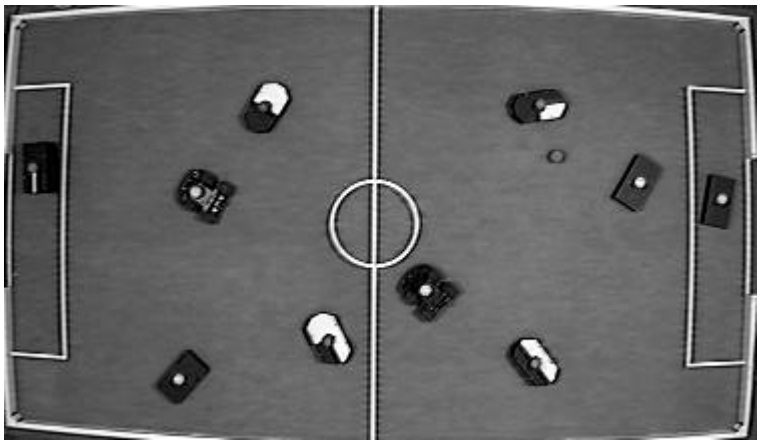

Abbildung 3.1: Schwarzweiße Markierungen der Roboter der RoboRoos [11]

Die RoboRoos von der University of Queensland aus Brisbane, Australien [11] verwendet ausschließlich schwarzweiße Markierungen zum Lokalisierung der Roboter (Abb. 3.1). Die Farbmarkierung wird nur benutzt, um die eigenen Roboter von denen der Gegenspieler zu unterscheiden. Diese Methode wäre grundsätzlich auf das hier beschriebene System übertragbar. Da aber auch die Roboterkörper selbst schwarzweiß auf dem Kamerabild erscheinen, ist es schwierig, darauf die richtige Position der Marker zu erkennen und die binären Codes fehlerfrei auszulesen.

Ein Beispiel für die Robotererkennung mittels mehrfarbiger Codes liefern die FU-Fighters von der Freien Universität Berlin [10] (Abb. 3.2). Diese Methoden sind für die Sony-four-legged-league nicht einsetzbar, da die meisten Farben exklusiv bestimmten Objekten auf dem Spielfeld zugewiesen sind (siehe Abschnitt 5.4.2.1).

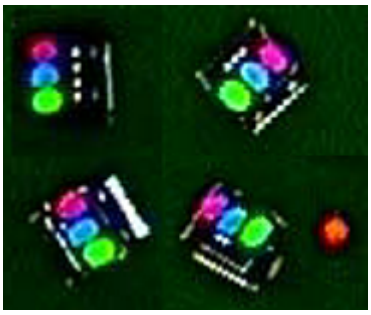

Abbildung 3.2: Farbige Spielermarkierungen der FU-Fighters [10]

Abbildung 3.3: Kombination aus farbigen Markern und Schwarzweiß-Codes des Teams 5dpo [4]

Das Team 5dpo von der Faculdade de Engenharia da Universidade do Porto aus Porto, Portugal [4] verwendet eine Kombination aus beiden Ansätzen. In der Mitte des kreisförmigen Markers befindet sich eine einfarbige Kreisfläche. Darum sind sektorweise schwarze und weiße Flächen angebracht. Der farbige Marker in der Mitte dient zur Lokalisierung des Markers. Die umgebenden schwarzweißen Flächen dienen zur Unterscheidung der verschiedenen Spieler und zur Bestimmung der Ausrichtung der Roboter (Abb. 3.3). Dieses Prinzip erscheint am besten auf die Gegebenheiten in der Sony-Liga anpassbar und diente deshalb als Vorlage für den zweiten Entwurf von Markern.

Kapitel 4

Vorverarbeitung

Um eine Objekterkennung auf Basis der Deckenkamerabilder zu ermöglichen, ist es zuerst nötig, jedem Pixel im Bild eine Position auf dem Feld zuordnen zu können. Hierbei sind zwei wesentliche Verzerrungen des Bildes zu beachten. Dies sind zum einen die intrinsischen, d.h. durch die Optik der Kamera verursachten und zum anderen die extrinsischen, d.h. perspektivischen Verzerrungen (Abb. 4.1). In den folgenden Abschnitten wird die Modellierung dieser Fehler und deren Kompensation (Abb. 5.1) erläutert.

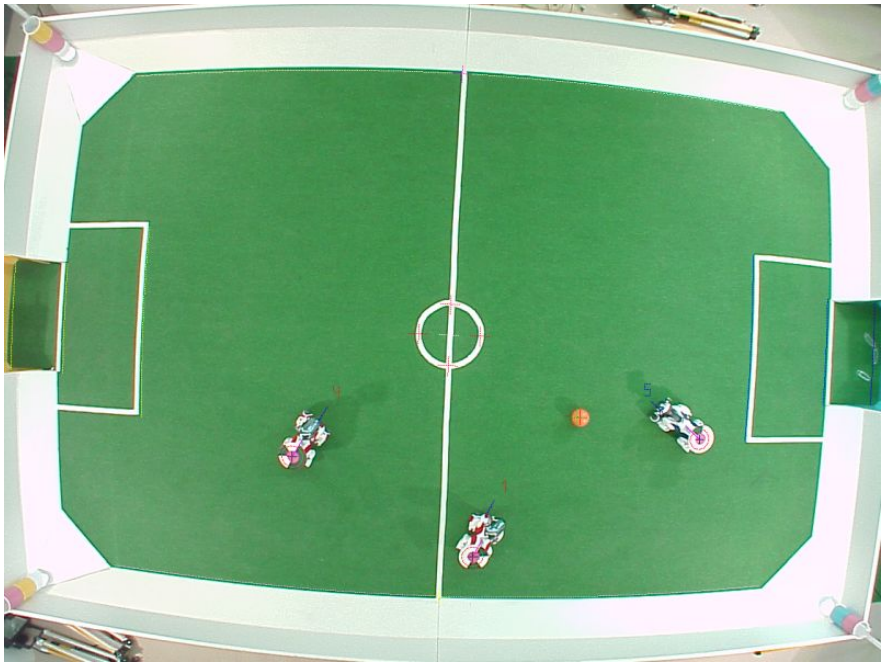

Abbildung 4.1: Verzerrtes Kamerabild

4.1 Kompensation der intrinsischen Verzerrung

Lichtstrahlen, die durch das Objektiv einer Kamera auf deren Sensor fallen, müssen je nach Einfallswinkel unterschiedlich lange Wege durch die Linse zurücklegen. Dadurch werden sie auch unterschiedlich stark gebrochen. Der Effekt ist umso größer, je weiter ein Bildpunkt von der optischen Mitte, also dem Zentrum der Linse, entfernt ist und je weitwinkliger ein Objektiv ausgelegt wurde. Um das gesamte Spielfeld mit der Deckenkamera erfassen zu können, ist diese mit einem Weitwinkelobjektiv ausgestattet. In den äußeren Bildbereichen kommt es dadurch zu erheblichen Verzerrungen. Gerade Linien erscheinen beispielsweise als nach innen hin gebogene Kurven. Dieser „Fischaugeneffekt“ muss bei der Bildauswertung berücksichtigt und kompensiert werden. Neben diesen Verzerrungen, die den Hauptteil der intrinsischen Fehler ausmachen, gibt es noch weitere kleinere Effekte. So ist die optische Mitte des Objektivs beispielsweise selten identisch mit der Mitte des Sensors und die Pixel nicht vollständig quadratisch, sondern leicht rechteckig.

In der Entwicklungsphase des Projektes wurde an Hand der Bilder der Deckenkamera recht schnell ersichtlich, dass die Verzerrung durch das Objektiv auf jeden Fall bei der Objekterkennung berücksichtigt werden muss. Messreihen (Abb. 4.2) zeigten deutliche Nichtlinearitäten hinsichtlich der Pixelentfernung zum Bildmittelpunkt und der tatsächlichen Entfernung von Messpunkten auf dem Boden.

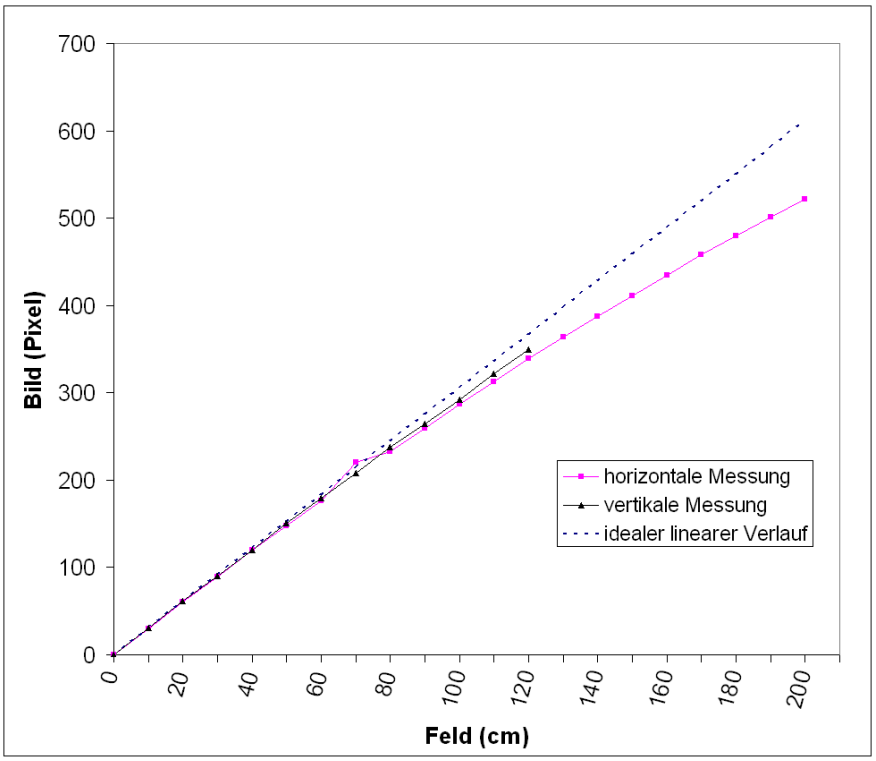

Abbildung 4.2: Verzerrungen durch das Objektiv

4.1.1 Näherung der radialen Verzerrung mit Polynominterpolation

In einem ersten Ansatz zur Kompensation des Problems entschloss man sich zur einer Modellierung des Fehlers durch eine Polynominterpolation. Begonnen wurde hierbei mit einem kubischen Polynom und demzufolge drei Messpunkten zur Kalibrierung. Die intrinsische Verzerrung hat im Wesentlichen nur eine radiale Komponente, d.h. Pixel im Bild erscheinen näher am Bildmittelpunkt, als sie idealisiert betrachtet liegen dürften. Daher wurden die Pixelpositionen in Polarkoordinaten, also in Winkel und Entfernung zum Bildmittelpunkt, umgerechnet und anschließend über das kubische Polynom die korrigierte Entfernung bestimmt. Die damit erzielten Ergebnisse wirkten auf den ersten Blick recht ermutigend. Die gröbsten Verzerrungen wurden kompensiert, aber es waren noch immer Abweichungen erkennbar. Zudem zeigte sich, dass die Modellierung über ein kubisches Polynom ungeeignet war. Je nach Wahl der Messpunkte bei der Kalibrierung traten entartete Polynome auf, die zwar die verwendeten Messpunkte enthielten, aber für die dazwischen liegenden Bereiche vollkommen unsinnige Werte lieferten. Erste Ansätze, anstatt dreier diskreter Messpunkte eine Optimierung der Parameter bezüglich einer größeren Menge an Kalibrierungsdaten durchzuführen, wurden allerdings recht schnell verworfen.

4.1.2 Kameramodell für intrinsische Verzerrungen

Als aussichtsreichere und fundiertere Lösung des Problems wurde schließlich eine Modellierung von Jean-Yves Bouguet gewählt [2]. Sein internes Kameramodell basiert im Wesentlichen auf dem von Heikkilä und Silven [5]. Das komplexe Modell schließt sowohl radiale Verzerrungen bis zur sechsten Ordnung als auch tangentielle Verzerrungen mit ein. Im Rahmen dieser Diplomarbeit wird allerdings nur ein Teil des Modells verwendet. Für die gewünschte Genauigkeit ist ein reduziertes Modell ausreichend, welches die tangentiellen Verzerrungen vernachlässigt und die radiale Verzerrung lediglich bis zur vierten Ordnung modelliert.

Die verwendeten Parameter des Modells sind damit der Vektor $\vec{fc}$, der die Brennweite (Focallength) in X- und Y-Richtung enthält, der Vektor $\vec{cc}$ mit den Koordinaten des optischen Ursprungs der Kamera (Principal Point / Optical Center) sowie für die radiale Verzerrung die Faktoren zweiter (kc_1) und vierter (kc_2) Ordnung.

Ein Punkt P mit den Koordinaten (X_c, Y_c, Z_c) im Kamerakoordinatensystem kann nun unter Beachtung dieser Parameter auf die Bildebene projiziert werden. Die einfache perspektivische Abbildung liefert:

$$\vec{n} = \begin{pmatrix} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Nun werden die radialen Verzerrungen durch das Objektiv hinzugerechnet und es ergibt sich:

$$r = \sqrt{x^2 + y^2}$$

$$\vec{d} = \left(1 + kc_1 * r^2 + kc_2 * r^4\right) * \vec{n}$$

Damit ergeben sich die Pixelkoordinaten des Punktes schlussendlich zu:

$$\vec{p} = \begin{pmatrix} \left(\vec{fc}\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) * \left(\vec{d}\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) \\ \left(\vec{fc}\begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) * \left(\vec{d}\begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) \end{pmatrix} + \vec{cc}$$

Die benötigten Parameter ermittelt die Matlab Toolbox von Bouguet durch ein Optimierungsverfahren auf Basis einer Reihe von Bildern eines in unterschiedlichen Abständen und Winkeln vor der Kamera platzierten Schachbrettmusters. Ein erster Testlauf mit einem behelfsmäßig gebastelten Schachbrett und einer Parameteroptimierung mit der Toolbox führte zu wesentlich besseren Ergebnissen als alle bisherigen Versuche. Die intrinsischen Verzerrungen der Kamera werden damit mehr als zufriedenstellend kompensiert und der Kalibrierungsprozess ist relativ benutzerfreundlich. Zudem liefert die Toolbox auch genauere Daten zur Kamera, z.B. wird bei der Brennweite zwischen horizontaler und vertikaler Richtung unterschieden und somit der Tatsache Rechnung getragen, dass die einzelnen Pixel auf dem CCD-Sensor der Kamera nicht quadratisch, sondern rechteckig angeordnet sind.

Da die intrinsischen Verzerrung nur von Kamera und verwendetem Objektiv abhängig sind, braucht diese Kalibrierung nur einmalig erfolgen. Die Tatsache, dass die Optimierung der Parameter also extern über die Matlab Toolbox geschieht und nicht in die Software zur Auswertung der Deckenkamerabilder integriert wurde, stellt nach Ansicht der Autoren somit keinen Nachteil dar. Eine Beschreibung zur Verwendung der Toolbox findet sich im Detail auf den Webseiten des Autors Jean-Yves Bouguet [2] sowie ein Überblick dazu im Anhang A.2.

4.2 Extrinsische Entzerrung

Nachdem die Verzerrungen des Linsensystems entfernt wurden, ergibt sich ein Bild wie in Abb. 4.3. Im nächsten Schritt müssen nun Rotationen sowie sich daraus ergebende perspektivische Verzerrungen kompensiert werden. Außerdem müssen Positionen im Kamerabild in Koordinaten im Feldkoordinatensystem umgesetzt werden. Dazu wurden jeweils parallel zu den verschiedenen Methoden der intrinsischen Entzerrungen unterschiedliche Ansätze verwendet.

4.2.1 Näherung mittels Bilinearform

Im ersten Ansatz wurde versucht, die intrinsischen Verzerrungen durch Annäherung mit einem kubischen Polynom zu modellieren. Hierzu wurde der Mittelpunkt des Kamerabildes auf den Feldmittelpunkt ausgerichtet. Da hierbei ohnehin in Polarkoordinaten umgerechnet wurde, konnte im gleichen Schritt die Rotation der Kamera um die

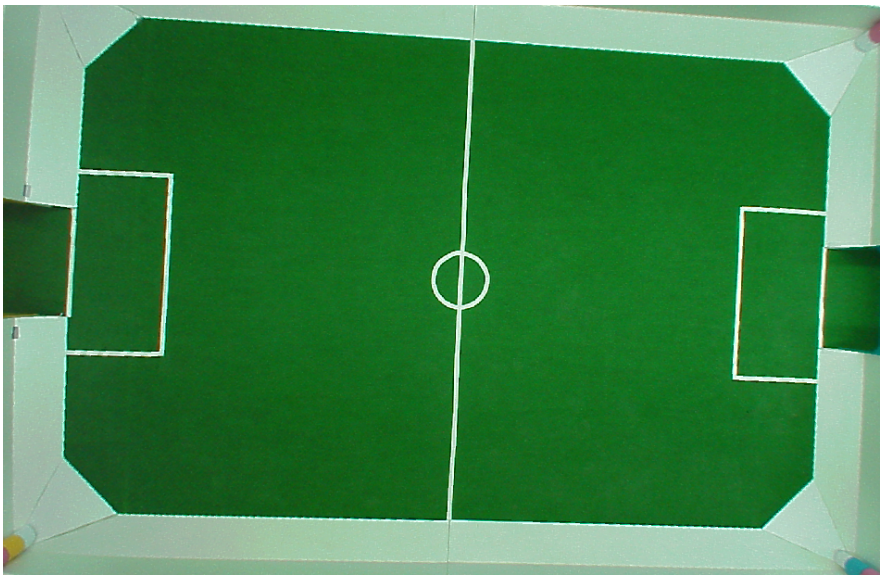

Abbildung 4.3: Das Kamerabild nach der Entzerrung der intrinsischen Fehler

Blickrichtung kompensiert werden. Außerdem sorgte die kubische Funktion, die auf die Entfernung zum Mittelpunkt angewendet wurde, dafür, dass die Einheit von Pixel in Millimeter umgerechnet wurde. Zur vollständigen Entzerrung des Bildes musste also nur noch die perspektivische Verzerrung beseitigt werden, die dadurch entsteht, dass die Kamera nicht exakt über dem Feldmittelpunkt aufgehängt ist und so leicht schräg in Richtung Feldmittelpunkt blickt.

Hierzu wurde ein Verfahren angewandt, das die Verzerrungen mittels einer Bilinearform folgender Gestalt annähert:

$$\begin{aligned} \tilde{x} &= a_1 * x + a_2 * xy + a_3 * y + a_0 \\ \tilde{y} &= b_1 * x + b_2 * xy + b_3 * y + b_0 \end{aligned}$$

Hierbei sind (x, y) die Koordinaten im perspektivisch verzerrten Bild und $(\tilde{x}, \tilde{y})$ das Ergebnis der Entzerrung. Zur Anpassung an die gegebenen Verhältnisse dienen hierbei die acht Parameter a_{0-3} und b_{0-3}. Zur Bestimmung dieser Parameter wird für jede der obigen Gleichungen ein Gleichungssystem bestehend aus vier Gleichungen aufgestellt, in die für $\tilde{x}_i$ und $\tilde{y}_i$ für vier verschiedene Punkte auf dem Spielfeld die jeweiligen tatsächlichen Feldkoordinaten eingesetzt werden. Für x_i und y_i hingegen werden die Werte eingesetzt, die die intrinsische Entzerrung für eben diese Punkte im Kamerabild liefert.

$$\begin{aligned} \tilde{x}_1 &= a_1 * x_1 + a_2 * x_1 y_1 + a_3 * y_1 + a_0 \\ \tilde{x}_2 &= a_1 * x_2 + a_2 * x_2 y_2 + a_3 * y_2 + a_0 \\ \tilde{x}_3 &= a_1 * x_3 + a_2 * x_3 y_3 + a_3 * y_3 + a_0 \\ \tilde{x}_4 &= a_1 * x_4 + a_2 * x_4 y_4 + a_3 * y_4 + a_0 \end{aligned}$$

$$\begin{aligned}
\tilde{y}_1 &= b_1 * x_1 + b_2 * x_1 y_1 + b_3 * y_1 + b_0 \\
\tilde{y}_2 &= b_1 * x_2 + b_2 * x_2 y_2 + b_3 * y_2 + b_0 \\
\tilde{y}_3 &= b_1 * x_3 + b_2 * x_3 y_3 + b_3 * y_3 + b_0 \\
\tilde{y}_4 &= b_1 * x_4 + b_2 * x_4 y_4 + b_3 * y_4 + b_0
\end{aligned}$$

Bei der Auswahl dieser vier Punkte wurde darauf geachtet, möglichst weit auseinander liegende Punkte zu verwenden, also z. B. die Feldeckpunkte oder die Ecken der Strafräume. Außerdem wurde der Feldmittelpunkt hinzugenommen, der jeweils die Koordinaten $(0, 0)$ besitzt. Dies führt dazu, das der Mittelpunkt auch immer als Mittelpunkt erkannt wird und die Parameter a_0 und b_0 immer den Wert 0 erhalten. Das Lösen des Gleichungssystems konnte vorberechnet werden, so dass direkt aus der Eingabe der Sollkoordinaten der drei freien Punkte und deren Position im intrinsisch entzerrten Bild die sechs übrigen Parameter a_{1-3} und b_{1-3} ermittelt werden konnten.

In der ersten Testumgebung führte dies zu guten Ergebnissen. In der Umgebung des neuen Robotiklabors führte die Wahl der drei freien Punkte verteilt auf drei der vier Quadranten des Spielfeldes allerdings dazu, dass die Entzerrung im vierten Quadranten nur sehr ungenaue Ergebnisse lieferte. Um dies zu vermeiden, wurden für alle vier Koordinaten Soll- und Istposition vermessen und dann aus jeweils drei davon die Parameter berechnet. Dies wurde für alle Kombinationen durchgeführt und die Ergebnisse für die sechs Parameter danach gemittelt. Dies führte zu einer Gleichverteilung des Fehlers über das gesamte Spielfeld. Dieser Fehler war allerdings immer noch zu groß, so dass über eine Optimierung über noch mehr Messpunkte nachgedacht wurde.

Zwei Faktoren sprachen aber gegen eine Weiterverfolgung dieses Ansatzes. Zum einen liefert das neue Verfahren zur Eliminierung der intrinsischen Verzerrung nun Richtungsvektoren aus Sicht des Kamerakoordinatensystems, so dass auch die Kompensierung der Rotation um die Sichtachse nun bei der Beseitigung der extrinsischen Fehler stattfinden muss. Auf der anderen Seite wurde festgestellt, dass die Umrechnung von Pixelkoordinaten im Bild zu Feldkoordinaten (und entgegengesetzt) nicht nur für die zweidimensionale Ebene des Spielfeldes zur Verfügung stehen muss, sondern auch in Höhe des Balles und der Robotermarker. Da die Entzerrung mittels obiger Bilinearform nur im Zweidimensionalen arbeitet, ist sie für diese Anforderungen also nicht geeignet. Deswegen wurde darauf verzichtet und ein neuer Ansatz gewählt.

4.2.2 Direkte Berechnung mittels Transformationsmatrix

Wie bereits erwähnt, kann man das Ergebnis der intrinsischen Entzerrung als Richtungsvektor $\vec{c}$ im Koordinatensystem der Deckenkamera interpretieren. Um daraus Koordinaten auf dem Spielfeld zu ermitteln, wird eine Gerade mit Aufhängungspunkt an der Kamera und Richtung $\vec{c}$ mit der Spielfeldebene geschnitten. Zur Bestimmung von Feldkoordinaten von Objekten oberhalb der Spielfeldebene wird die zu schneidende Ebene um die Höhe des Objekts h parallel zum Spielfeld nach oben verschoben.

Der Ursprung des Feldkoordinatensystems befindet sich im Mittelpunkt des Spielfeldes. Die x-Achse zeigt in Richtung des blauen Tores. Die y-Achse verläuft ent-

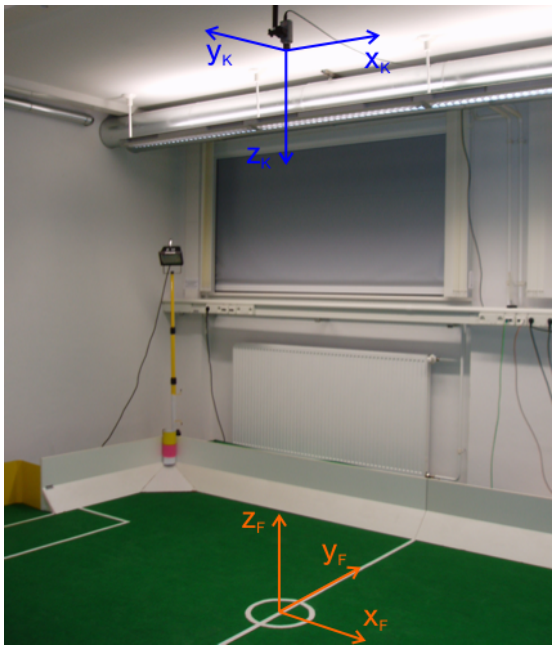

Abbildung 4.4: Koordinatensysteme von Kamera und Feld

lang der Mittellinie nach links (bei Blickrichtung entlang der x-Achse). Die z-Achse schließlich zeigt vom Feldmittelpunkt senkrecht nach oben. Das Kamerakoordinatensystem hat seinen Nullpunkt in der Linse der Kamera. X- und y-Achse verlaufen gemäß den Pixelkoordinaten nach rechts bzw. unten im Kamerabild. Die z-Achse verläuft entlang der Blickrichtung der Kamera. Zur Verdeutlichung dient Abb. 4.4, die die Situation im Robotiklabor widerspiegelt.

4.2.2.1 Aufstellen der Transformationsmatrix

Zum Aufstellen der Transformationsmatrix, die die Koordinaten vom Kamerakoordinatensystem in das Koordinatensystem des Spielfeldes überführt, dienen die Ortsvektoren im Feldkoordinatensystem, die wie in Anhang A.3 beschrieben ermittelt werden. Der Vektor zur Kameralinse $\vec{t}$ dient dabei direkt als Translationsteil der Transformationsmatrix. Zum Aufstellen der Rotationsmatrix wird zusätzlich der Vektor zum Feldpunkt, der im optischen Zentrum des Kamerabildes liegt, $\vec{o}$ benötigt, außerdem der Vektor $\vec{u}$, der von dort zu einem Feldpunkt zeigt, der im Kamerabild direkt unterhalb des optischen Zentrums liegt.

Die dritte Spalte $\vec{r_3}$ der Rotationsmatrix entspricht der Blickrichtung der Kamera, also dem Vektor von Kamera zum optischen Zentrum, in Feldkoordinaten.

$$\vec{r_3} = \vec{o} - \vec{t}$$

In die zweite Spalte $\vec{r_2}$ wird die Richtung der y-Achse des Kamerakoordinatensystems in Feldkoordinaten eingetragen. Diese erhält man, indem man die x- und y-

Kompenenten des Vektors $\vec{u}$ verwendet und die z-Koordinate so wählt, dass $\vec{r_2}$ senkrecht auf $\vec{r_3}$ steht.

$$\vec{r_2} = \begin{pmatrix} u_x \\ u_y \\ \frac{r_{3,x}u_x + r_{3,y}u_y}{-r_{3,z}} \end{pmatrix}$$

Da das Kamerakoordinatensystem ein orthogonales Rechtssystem ist, kann die erste Spalte $\vec{r_1}$, die der Richtung dessen x-Achse im Feldkoordinatensystem entspricht, als Vektorprodukt aus $\vec{r_2}$ und $\vec{r_3}$ berechnet werden.

$$\vec{r_1} = \vec{r_2} \times \vec{r_3}$$

Die komplette Rotationsmatrix R setzt sich schließlich aus den normalisierten Spalten $\vec{r_1}$, $\vec{r_2}$ und $\vec{r_3}$ zusammen.

$$R = \begin{pmatrix} \frac{\vec{r_1}}{|\vec{r_1}|} & \frac{\vec{r_2}}{|\vec{r_2}|} & \frac{\vec{r_3}}{|\vec{r_3}|} \end{pmatrix}$$

Mithilfe dieser Rotationsmatrix und des Translationsvektors lassen sich nun Kamera- und Feldkoordinaten ineinander umrechnen.

4.2.2.2 Von Kamera- zu Feldkoordinaten

Um einem im Kamerabild erkannten Objekt der Höhe h eine Position in Feldkoordinaten zuweisen zu können, werden die Pixelkoordinaten zuerst durch die intrinsische Entzerrung korrigiert. Das Ergebnis ist, wie bereits erwähnt, der Richtungsvektor $\vec{c}$ in Kamerakoordinaten. Daraus ergibt sich der Richtungsvektor von der Kamera zu dem erkannten Objekt im Feldkoordinatensystem $\vec{v}$ durch Multiplikation mit R.

$$\vec{v} = R\vec{c}$$

Für den Ortsvektor $\vec{f}$ vom Feldmittelpunkt zu dem Objekt lässt sich somit folgende Gleichung aufstellen:

$$\vec{f} = \vec{t} + n\vec{v}$$

Da bekannt ist, dass $f_z = h$, lässt sich daraus der Parameter n folgendermaßen ermitteln:

$$n = \frac{h - t_z}{v_z}$$

Die Position des Objektes im Feldkoordinatensystem ergibt sich sodann aus obiger Gleichung. Somit ist es möglich, die x- und y-Koordinaten für jedes Objekt zu ermitteln, dessen Höhe über dem Spielfeld bekannt ist.

4.2.2.3 Von Feld- zu Kamerakoordinaten

Zur Darstellung eines entzerrten Kamerabildes muss für jede Position auf dem Spielfeld das korrespondierende Pixel im Kamerabild ermittelt werden können. Für den extrinsischen Teil der Entzerrung bedeutet dies, aus dem Ortsvektor $\vec{f}$ in Feldkoordinaten den Richtungsvektor $\vec{c}$ im Kamerakoordinaten zu ermitteln. Dazu wird zuerst der Vektor $\vec{v}$ von dem Kamera zum Punkt auf dem Spielfeld bestimmt:

$$\vec{v} = \vec{f} - \vec{t}$$

Dabei repräsentiert $\vec{t}$ wieder den Ortsvektor der Kamera im Feldkoordinatensystem. Zur Bestimmung von $\vec{c}$ muss dann nur noch die transponierte Rotationsmatrix R mit diesem Vektor $\vec{v}$ multipliziert werden.

$$\vec{c} = R^T \vec{v}$$

Aus diesem Richtungsvektor wird dann schließlich mithilfe der intrinsischen Parametern die Pixelkoordinaten im Kamerabild bestimmt.

Kapitel 5

Objekterkennung

Durch die im vorherigen Abschnitt beschriebenen Algorithmen ist es möglich, jedem Pixel im Bild eine Position auf dem Feld, bzw. in jeder beliebigen Höhe über dem Feld zuzuordnen. Sowohl die intrinsischen als auch die extrinsischen Verzerrungen (Abb. 4.1) werden damit kompensiert (Abb. 5.1). Da die nötigen Berechnungen allerdings relativ viel Rechenzeit benötigen, wird zur Objekterkennung nicht jedes Bild komplett entzerrt, sondern nur die zur Auswertung benötigten Bereiche. Mit dieser Voraussetzung ist die Erkennung von Objekten im Bild und die Detektion ihrer Position auf dem Feld möglich. Im folgenden Abschnitt werden die Routinen zur Objekterkennung und Modellierung näher erläutert.

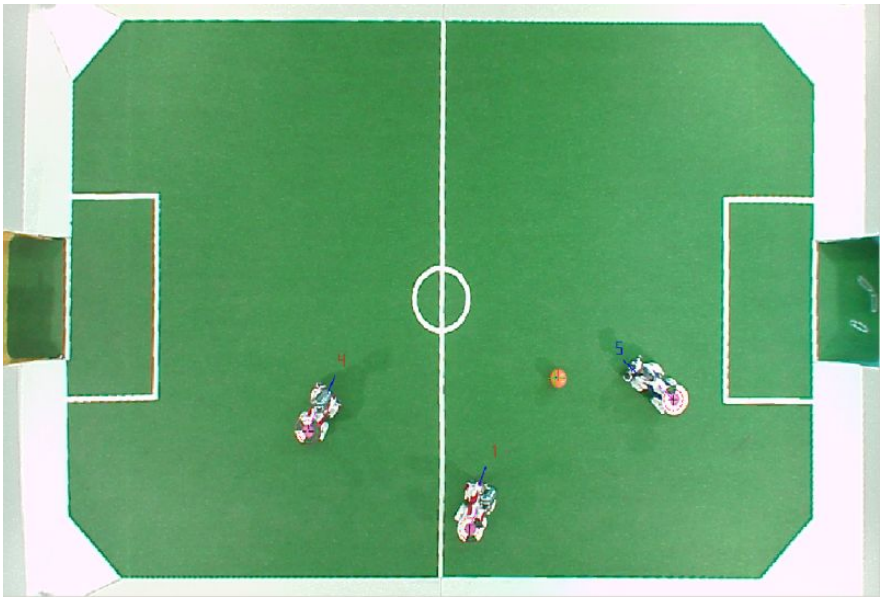

Abbildung 5.1: Zurückgerechnetes Bild nach der Entzerrung

5.1 Farbklassifizierung

Die verwendete Deckenkamera liefert Farbbilder in hervorragender Qualität. Sehr früh in der Entwicklungsphase des System wurde daher bereits ersichtlich, dass eine Objekterkennung auf Basis einer Farbklassifizierung des Bildes die naheliegendste und vor allem auch aussichtsreichste Lösung darstellt. Das YUV-Farbtabellenmodell, welches vom *GermanTeam* bei der Programmierung der Roboter genutzt wird, wurde daher auf den RGB-Farbraum übertragen, in dem die Bilder der Deckenkamera vorliegen. Mittels eines einfachen Farbtabellentools lassen sich Blöcke im RGB-Farbraum einzelnen Farbklassen zuordnen. Motiviert durch die gute Farbwidergabe der Kamera wurden Filter programmiert, die das Kamerabild in verschiedenen Farbkanälen darstellen. Dabei wird ein Graustufenbild erzeugt, dass die Abweichung der Farbe eines Pixels auf RGB-Bild bezüglich der jeweils idealisierten Farbe darstellt. Je höher der Helligkeitswert auf dem Graustufenbild, desto ähnlicher die Farbe zur Referenzfarbe (Abb. 5.2):

$$\text{similarityToOrange} = \begin{cases} r - \frac{3}{2}b - 2 * (g - \frac{r}{2}), g >= \frac{r}{2} \\ r - \frac{3}{2}b - \frac{3}{2} * (\frac{r}{2} - g), g < \frac{r}{2} \end{cases}$$

$$\text{similarityToPink} = r - 2 * \left| g - \frac{r}{2} \right| - 2 * \left| b - \frac{r * 5}{7} \right|$$

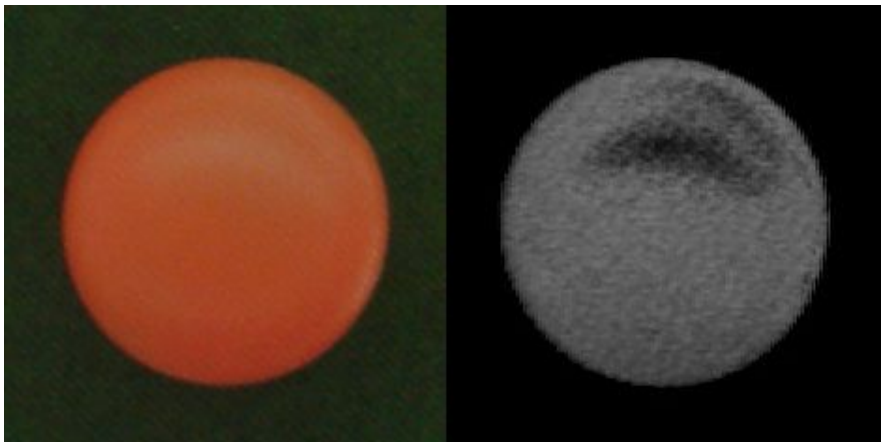

Abbildung 5.2: Ball aus Kamerasicht und Orangeähnlichkeit

Alle Farben, die einen einstellbaren Ähnlichkeitswert erreichen oder überschreiten, lassen sich der Farbklasse in der Farbtabelle zuschreiben. Dadurch ist es möglich, diese sehr schnell und sozusagen „ohne Löcher“ zu erstellen. Des Weiteren ist ähnlich zu der im *GermanTeam*-Code gebräuchlichen Farbtabellenerstellung auch ein manuelles „zusammenklicken“ der Farbtabelle möglich. Dabei werden durch Markieren von Pixeln im Bild Blöcke von Farben um die betreffende Position des Pixels im Farbraum einer Farbklasse zugeordnet.

Aufgrund der guten Farbwidergabe der Kamera kann das System auch leicht variierende Beleuchtungsverhältnisse im Labor nach einer einmaligen Kalibrierungsphase mit nur einer einzigen Farbtabelle handhaben. Das überraschend gute Ergebnis der mit einfachen Mitteln automatisch generierten Farbtabelle, die bereits ohne manuelles Eingreifen passable Farbklassifizierung erlaubt, lässt darauf hoffen, dass in Zukunft die Erstellung vollständig überflüssig werden könnte und eine direkte Erkennung der Farben denkbar wird. Entsprechende Ansätze zur automatischen Farbkalibrierung für die Roboter waren dort bisher auf Grund der relativ schlechten Farbwiedergabe der Roboterkamera gescheitert. Durch ihre im Vergleich zu den Robotern wesentlich bessere Brillianz könnten solche Ansätze im Rahmen zukünftiger Weiterentwicklungen allerdings für die Deckenkamera durchaus zum Erfolg führen.

5.2 Objektsuche

Den ersten Schritt zur Objekterkennung bildet das prinzipielle Auffinden der Teilbereiche des Bildes, in denen ein Objekt vermutet wird. Die Klasse *ImageProcessor* legt zu diesem Zweck ein Suchraster über das komplette Bild und untersucht so nur einen kleinen Bruchteil der verfügbaren Pixel. Eine komplette Farbklassifizierung des gesamten Kamerabildes ist auf Grund der Echtzeitanforderungen an der System weder sinnvoll noch nötig, um eine sichere Erkennung der Objekte zu gewährleisten. In diesem Arbeitsschritt genügt es, Objekte nicht zu *übersehen*, eine exakte Bestimmung ihrer Position und Identität erfolgt in weiteren Bearbeitungsschritten. Im Einzelnen wird so bei den Schlüsselfarben Orange und Pink die Ball bzw. Markererkennung angestoßen. Die Erkennung von Ball und Robotern erfolgt dann durch spezialisierte Algorithmen, die in den folgenden beiden Abschnitten beschrieben werden.

5.3 Ball

Die Erkennung des Balles wird beim Auffinden orangefarbiger Pixel während der Objektsuche angestoßen. Die verwendeten Algorithmen wurden in Anlehnung an den für die Roboter genutzten Varianten für das Deckenkamerasystem angepasst und erweitert. Während dem Abscannen eines 5 Pixel breiten Rasters über das unverarbeitete Kamerabild werden die Pixel anhand der Farbtabelle klassifiziert. Wenn dabei ein orangener Pixel gefunden wird, wird getestet, ob er auf den Boden projiziert innerhalb der Spielfeldgrenzen liegt. Hierbei wird für die Rückrechnung der Kameraverzerrungen angenommen, dass sich orangene Punkte auf halber Ballhöhe über dem Feldboden befinden. Falls sich der Pixel demnach innerhalb des Spielfeldes befindet, wird überprüft, ob er zu einem der im aktuellen Bild evtl. bereits erkannten Bälle gehört. Ist dies nicht der Fall, wird der für das Deckenkamerasystem adaptierte *GT2004BallSpecialist* aufgerufen.

5.3.1 Erkennung des Balls

Ausgehend von dem orangenen Pixel, der innerhalb eines Balles vermutet wird, sucht der *BallSpecialist* in verschiedene Richtungen nach Randpunkten des vermuteten Balls. Dazu werden sowohl horizontale wie auch vertikale und diagonale Scanlinien gestartet, die von diesem Startpunkt ausgehend nach außen den Rad des Balles finden sollen. Dabei werden die Farben der getesteten Pixel klassifiziert und dahingehend bewertet, ob sie noch innerhalb oder außerhalb des Balls liegen. Zur Entscheidung, ob der aktuell untersuchte Farbwert noch als eine Form von Orange angesehen werden kann, wird hierbei allerdings nicht länger die statische Farbtabelle eingesetzt, die beim Abscannen des Rasters benutzt wurde, sondern direkt die im Abschnitt *Farbklassifizierung* vorgestellte *Orangeähnlichkeit*. Dies hat den Vorteil, dass in der Farbtabelle nur eindeutig zum Ball gehörige Farben auch als Orange klassifiziert werden müssen und nur auf diesen eine Triggerung während der Objektsuche durchgeführt wird. Während der Suche nach den Randpunkten des Balles hingegen ist es sinnvoll, weniger enge Bedingungen an die Farben der Pixel zu stellen. Glanzlichter oder Schatten auf den Bällen sollen die Suche nach den Randpunkten möglichst nicht unterbrechen, sondern dabei als zum Ball gehörig angesehen werden. Gleichzeitig muss aber auch bedacht werden, dass die hellen und teilweise stark ihre Umgebung widerspiegelnden Roboter auch Bälle verdecken und so aus Sicht der Deckenkamera anschneiden können. Ein solcher angeschnittener Ball präsentiert sich gegenüber der Deckenkamera als eine Reihe von Randpunkten, von denen einige am Grün des Teppichs also am echten Rand des Balles liegen und andere im eigentlichen Ball als Grenzpunkte zum Roboterkörper auftreten. Die gefundenen Randpunkte werden daher klassifiziert und solche mit einem sehr starken Kontrast bezogen auf die Orangeähnlichkeit bevorzugt zur weiteren Berechnung herangezogen. Anschließend wird geprüft, ob die verbleibenden Punkte im Rahmen einer gewissen Toleranz innerhalb des so berechneten Balles liegen, wie es bei einem angeschnittenen Ball der Fall sein müsste. Nur falls dies nicht der Fall ist werden auch die Punkte mit geringem Kontrast für die Berechnung verwendet. Da die Grenzpunkte zum grünen Hintergrund des Teppichs somit bevorzugt zur Berechnung des Kreises herangezogen werden, der dem Ball im Kamerabild entspricht, werden die meisten im Bild angeschnittenen Bälle immer noch korrekt erkannt.

Nach der *Levenberg-Marquardt-Methode* [3] wird durch die so ausgewählten Randpunkte ein Ausgleichskreis gelegt und über diesen der Mittelpunkt des Balles bestimmt. Der Radius des so gefundenen Kreises muss im Rahmen gewisser Abweichungen dem für einen regulären Ball zu erwartenden Radius entsprechen und der gefundene Kreis muss genügend orangene Pixel enthalten, ansonsten wird kein Ball erkannt. Dies verhindert Fehlerkennungen von winzigen oder riesigen Bällen, sowie Geisterbälle auf Grund von Farbschatten an Spielfeldlinien, Banden oder Robotern. Erfüllt der erkannte Ball aber alle Bedingungen, so wird der so gefundene Mittelpunkt als im Ballradius über dem Feld schwebende Position des Ballmittelpunktes betrachtet und darüber die Position des Balles auf dem Feld berechnet. Erkennt das System mehr als einen Ball auf dem Bild, endet hiermit die Modellierung des Balles, wird nur ein

einziger Ball erkannt, erfolgt eine Berechnung der Geschwindigkeit und Richtung seiner Bewegung auf Basis seiner Position innerhalb der letzten 500 Millisekunden. Eine Ermittlung der Geschwindigkeit für mehr als einen Ball würde eine unterscheidung, also ein Tracking der verschiedenen Bälle erfordern. Da diese Situation im normalen Spielverlauf nicht vorkommen kann, wurde auf die Implementierung entsprechender Algorithmen allerdings verzichtet.

5.3.2 Ergebnisse des Ballerkenners

Zur Überprüfung der Güte des Ballerkenners wurde eine Reihe von Messungen durchgeführt. Zur Validierung der Daten des Deckenkamerasystems wurde der Ball an einigen leicht von Hand vermessbaren Feldpositionen abgelegt und anschließend mit *RobotControl* Logdateien mit den Weltbildern der Deckenkamera aufgezeichnet. Der Ball lag dabei an den Eckpunkten der Strafräume, dem Feldmittelpunkt sowie zwei Punkten auf der Mittellinie platziert, die jeweils 1 m vom Mittelpunkt entfernt liegen (Abb. 5.3). An jeder dieser sieben Positionen wurden ca. 100 Messungen des Deckenkamerasystems für die Bewertung herangezogen.

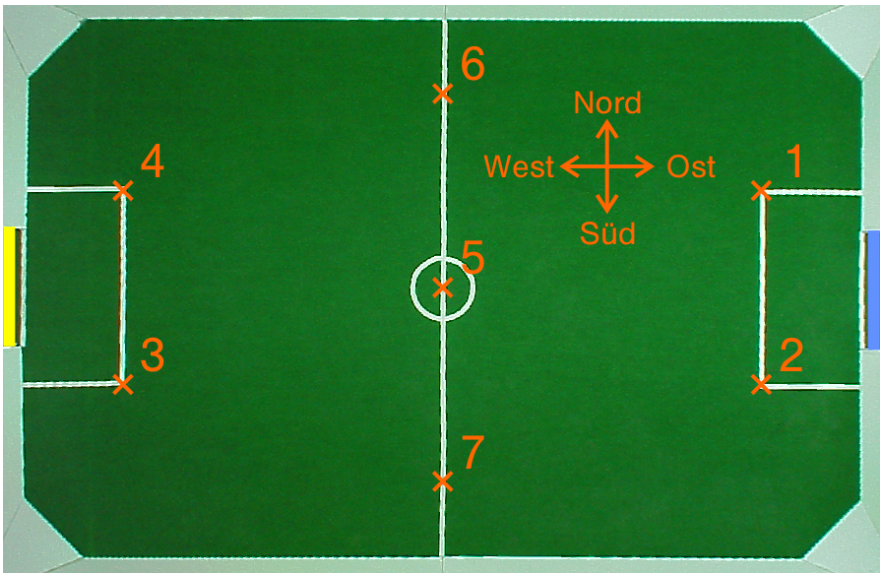

Abbildung 5.3: Feldpositionen für Messungen von Ball- und Robotererkenner

Die Daten der Deckenkamera über die Position des Balles wurden mit *RobotControl* aus den sieben Logdateien in kommaseparierte Dateien exportiert und anschließend mit Hilfe eines Tabellenkalkulationsprogramms analysiert. Für jede der von der Deckenkamera gelieferten Positionen wurde im Vergleich zur realen Position des Messpunktes auf dem Feld die Abweichungen in X- und Y-Achse sowie der absolute Positionsfehler bestimmt. An jedem der sieben Messpunkte ergeben sich somit die in Tabelle 5.1 aufgeführten mittleren Fehler. Ebenfalls in der Tabelle angegeben sind

als Maß für die Streuung der Messwerte die Standardabweichungen. Diese wurden mit den statistischen Funktionen des Tabellenkalkulationsprogramms jeweils über alle Messungen an dem jeweiligen Punkt ermittelt. Die letzte Zeile der Tabelle gibt die Mittelwerte über alle sieben Messreihen an.

		X-Richtung		*Y-Richtung*		*Position*
Pos	*Messungen*	*Ø-Fehler*	*Stdaw.*	*Ø-Fehler*	*Stdaw.*	*Ø-Fehler*
1	99	6,3 mm	2,7 mm	-2,8 mm	1,7 mm	6,9 mm
2	99	2,2 mm	3,3 mm	0,4 mm	0,7 mm	2,2 mm
3	94	-26,3 mm	0,9 mm	10,5 mm	0,0 mm	28,3 mm
4	101	-24,4 mm	2,5 mm	23,7 mm	2,1 mm	34,0 mm
5	102	0,0 mm	1,2 mm	-2,1 mm	1,4 mm	2,1 mm
6	102	4,0 mm	0,3 mm	16,0 mm	2,0 mm	16,5 mm
7	102	4,5 mm	1,1 mm	-11,8 mm	1,4 mm	12,6 mm
Mittel	699	-4,8 mm	1,7 mm	4,8 mm	1,3 mm	14,7 mm

Tabelle 5.1: Mittlere Fehler und Standardabweichungen des Ballerkenners

Bereits an diesem Punkt ist ersichtlich, dass der Hauptanteil der ermittelten Fehler an den Positionen drei und vier an den Strafraumecken des gelben Tores auftritt. Die gleichen Messpositionen ergaben auch bei den Messungen zur Güte des Roboterer-kenners, die im Abschnitt 5.4.3 detailliert beschrieben werden, die stärksten Beiträge zu den gemessenen Fehlern. Da die dazu symmetrischen Positionen eins und zwei am Strafraum des blauen Tores keine auffälligen Fehler aufzeigen, liegt die Vermutung nahe, dass es evtl. zu einem Fehler beim Ausmessen der Spielfeldlinien gekommen ist. Bei einem erneuten Nachmessen der Linien vor dem gelben Tor konnten Abweichungen von maixmal 5 mm festegestellt werden. Die verbleibenden Diffenrenzen sind wahrscheinlich auf Fehler bei der Bestimmung der Kameraparametern zurückzuführen, die durch die Messungenauigkeit entstehen. Insgesamt bleibt aber positiv anzumerken, dass der durchschnittliche Fehler auch an den beiden problematischen Positionen mit ca 3 cm relativ gering ausfällt und im Mittel sogar weniger als die Hälfte davon beträgt.

Die Fehler der einzelnen Messungen wurden mit Hilfe des Tabellenkalkulationsprogramms in Klassen von jeweils 5 mm Intervallbreite einsortiert. Die Beiträge der beiden angesprochenen auffälligen Messreihen drei und vier wurden dabei zur Illustration schraffiert dargestellt. Bei der Betrachtung der Verteilung des komponentenweisen Fehlers hinsichtlich der X-Achse (Abb. 5.4) fallen sofort die deutlichen Ausreißer aus diesen beiden Messreihen ins Auge. Während sich die anderen Messreihen in den Klassen geringer Fehler nahe der 0 mm Marke konzentrieren, häufen sich die Fehler aus diesen beiden Messreihen eher bei -20 mm bis -30 mm. Die Streuung fällt dabei mit einer Standardabweichung von im Mittel über alle sieben Messreihen 1,7 mm vergleichsweise sehr gering aus. Auch die maximale Standardabweichung hinsichtlich einer einzelnen Messreihe beträgt lediglich 3,3 mm und liegt damit quasi bei einem

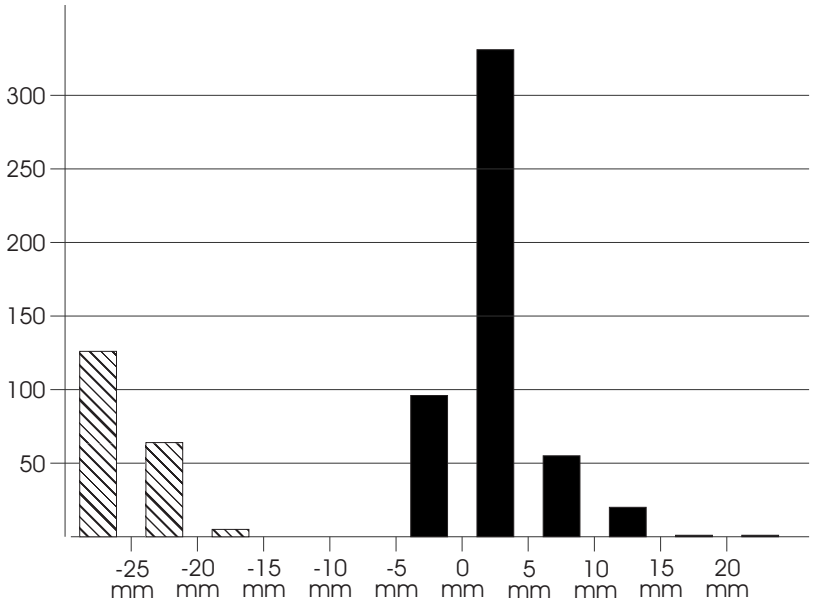

Abbildung 5.4: Verteilung des Fehlers auf der X-Achse

einzelnen Kamerapixel, denn die 1280 Pixel Breite des Kamerabildes verteilen sich auf ca. 4,20 m Feldlänge, was pro Pixel unter Beachtung der Verzerrungen des Bildes folglich in etwa 3 mm bis 4 mm entspricht. Dies lässt also die Aussage zu, dass der Ballerkenner hinsichtlich der Position des Balles im Bild äußerst präzise arbeitet, aber bei der Rückrechnung auf Feldkoordinaten offenbar ein prinzipbedingter Fehler auftritt. Dieser Fehler könnte sowohl in einer durch die Messungenauigkeiten nicht optimal kalibrierten Kamera als auch in Fehlern beim Ausmessen der erwarteten realen Werte für die sieben Messreihen begründet sein.

Betrachtet man hingegen die Verteilung der Fehler hinsichtlich der Y-Komponente (Abb. 5.5), zeigen die Messreihen drei und vier zwar auch stärkere Fehler als die übrigen fünf Messungen, aber sie stechen nicht so stark ins Auge wie bei den Fehlern im X-Wert. Die Standardabweichungen liegen unterhalb der Pixelauflösung der Kamera, was wieder die Stabilität der Bildverarbeitung bestätigt. Vergleichsweise etwas größere mittlere Fehler treten nun außer bei den Positionen drei und vier auch bei den Positionen sechs und sieben auf, also den beiden Positionen auf der Mittellinie in 1 m Abstand vom Feldmittelpunkt. Das geringe Auftreten von Fehlern in den Klassen zwischen -10 mm und -5 mm bzw. 5 mm und 10 mm erscheint zunächst merkwürdig, liegt aber bei genauerer Betrachtung der Messergebnisse darin begründet, dass die Streuungsbreite äußerst gering ausfällt und die auftretenden Fehler in den verschiedenen Messreihen so für deutlich voneinander getrennte Anhäufungen in der Gesamtverteilung sorgen.

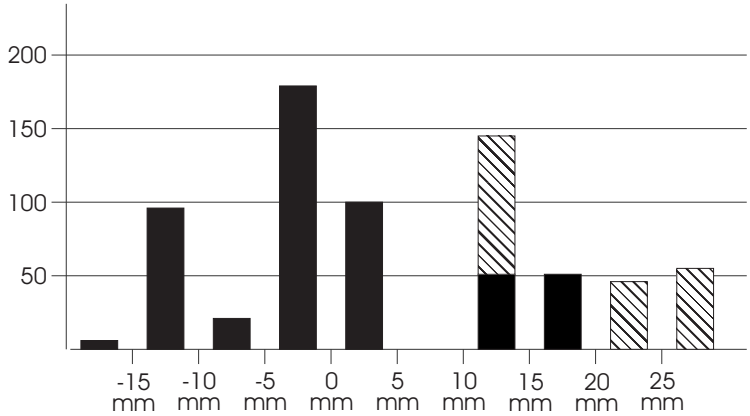

Abbildung 5.5: Verteilung des Fehlers auf der Y-Achse

Die Verteilung des absoluten Fehlers (Abb. 5.6) zeigt dementsprechend keine überraschend neuen Ergebnisse. Wieder tritt in der Klasse zwischen 5 mm und 10 mm eine vergleichsweise geringe Fehlermenge zu Tage, da keiner der Messpunkte entsprechende Abweichungen produzierte. Die Messreihen drei und vier liefern erwartungsgemäß auch im absoluten Fehler die deutlich stärksten Anteile. Der mittlere absolute Fehler von 1,5 cm sollte evtl. noch Anlass für weitere Überprüfungen der Güte des Gesamtsystems sein. Dabei könnten allerdings bereits auch Abweichungen durch die manuelle Positionierung des Balles an den Messpunkten die Erklärung liefern. Die hier verwendeten Messreihen wurden jeweils dadurch erzeugt, dass der Ball einmalig an der Position abgelegt und daraufhin eine Reihe von Messungen durchgeführt wurde. Das Experiment sollte daher evtl. in der Form erweitert werden, dass der Ball mehrfach von Hand neu an den betreffenden Punkt gelegt wird, um die menschliche Komponente dieses Fehlers herauszumitteln. Sollten dann immer noch ähnliche Ergebnisse auftreten, ist ggf. eine neue Kalibrierung der Kamera bzw. ein exakteres Ausmessen der Kameramatrix durch den Einsatz von genaueren Messwerkzeugen notwendig. Die dazu nötigen Verfahren werden im Anhang A ausführlich beschrieben.

5.4 Roboter

Weitaus komplexer als die Erkennung des einzelnen Balls ist das Auffinden und Unterscheiden der verschiedenen Roboter auf dem Spielfeld. Der Ball erscheint weitgehend als einfacher einfarbiger Kreis. Der Roboter dagegen hat eine komplexere Form, die sich außerdem durch die Bewegungen der einzelnen Gelenke des Roboters ständig ändern kann. Zudem besitzt der Roboter keine einheitliche Farbe, sondern besteht hauptsächlich aus weißen und spiegelnden Flächen und den Farben rot und blau des Trikots. Dazu kommt, dass bei der Robotererkennung eine Bestimmung der Positi-

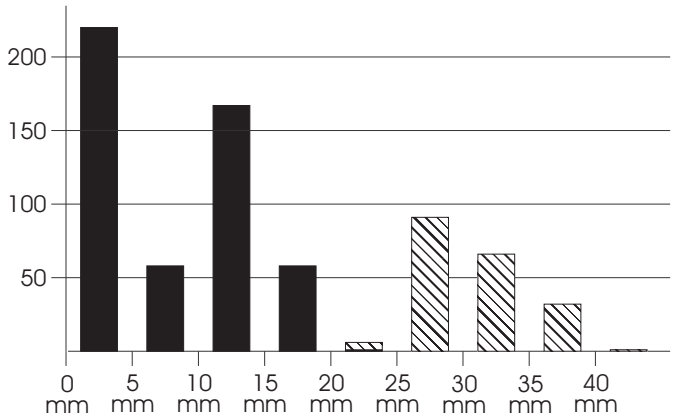

Abbildung 5.6: Verteilung des absoluten Fehlers

on nicht ausreicht, sondern auch die Ausrichtung ermittelt werden muss. Weiterhin müssen die Roboter auch unterschieden werden können. Es muss also zusätzlich die Spielernummer ermittelt werden.

5.4.1 Direktes Erkennen der Roboter

Um möglichst wenig in das *RoboCup*-Umfeld einzugreifen, wäre es am besten, die Roboter zu erkennen ohne diese zu verändern. Die Position der Roboter könnte man ermitteln, indem man auf dem Spielfeld nach Stellen sucht, die nicht in der grünen Farbe des Teppichs gefärbt sind. Dabei muss darauf geachtet werden, dass die Spielfeldlinien und der Ball auch in diese Kategorie fallen. Von den Spielfeldlinien ist die Position im Vornherein bekannt und können somit ausgeschlossen werden. Der Ball kann durch seine orangene Farbe ausgefiltert werden. Außerdem erscheinen Roboter immer ungefähr in der gleichen Größe auf dem Spielfeld. Dies kann als weiteres Kriterium zum Filtern der verschiedenen Löcher im grünen Bereich herangezogen werden.

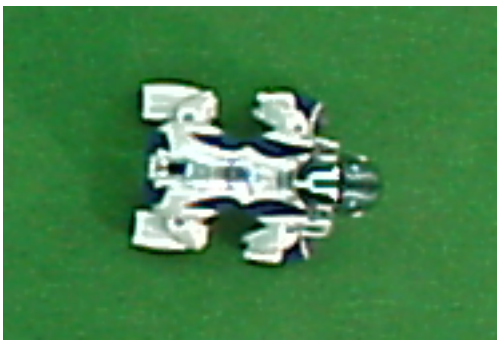

Abbildung 5.7: Deckenkameraaufnahme von Roboter ohne Marker

Es gibt allerdings einige Probleme, die gegen ein direktes Erkennen der Roboter sprechen. Zu einem genauen Modell der Objekte auf dem Spielfeld gehört auch eine möglichst genaue Angabe der Ausrichtung aller Roboter auf dem Spielfeld. Der Körper selbst lässt eine Unterscheidung von vorne und hinten kaum zu, da die von der Spiegelsymmetrie abweichenden Elemente für das Kameraauge zu klein sind. Es bliebe also nur der Kopf für diese Erkennung. Dieser ist aufgrund seiner spiegelnden Oberfläche sehr schwer zu erkennen (Abb. 5.7). Außerdem kann er in Positionen bewegt werden, in denen er im Kamerabild kaum zu erkennen ist. Eine Berechnung der Roboterausrichtung kann ohne zusätzliche Hilfsmittel also nur sehr ungenau und fehleranfällig realisiert werden.

Ein weiterer Punkt, der gegen das direkte Erkennen der Roboter spricht, ist die Notwendigkeit, die Roboter zu unterscheiden. Erst dadurch wird es möglich, die Daten der Deckenkamera später einem bestimmten Roboter zuzuordnen und mit dem von diesem Roboter gelieferten Daten zu vergleichen. Für den menschlichen Spielbeobachter dienen dazu die rot und blau gefärbten Trikots zum Erkennen der Teamzugehörigkeit sowie aufgeklebte Ziffern an drei Seiten des Kopfes zur Identifizierung der Spielernummer. Leider sind diese beiden Unterscheidungshilfen fast vollständig nur an den Seiten des Roboters angebracht und aus der Perspektive der Kamera unter der Decke nicht einzusehen. Deshalb können diese nicht zur Roboterunterscheidung im Kamerabild herangezogen werden. Ein Ausweg wäre, den verschiedenen erkannten Roboter beim Start manuell eine Nummer zuzuweisen und sie während des Spielverlaufs zu verfolgen. Dieser manuelle Zusatzaufwand sollte allerdings möglichst verhindert werden. Außerdem führt dieses Tracking fast zwangsläufig zu Verwechslungen, sobald sich die Roboter sehr nahe kommen, z. B. bei einem längeren Kampf um den Ball.

5.4.2 Erkennen mittels Marker

Um die Ausrichtung der Roboter genau erfassen und die Roboter eindeutig identifizieren zu können, sind also Erkennungshilfen am Roboter notwendig. In der Smallsize-Liga des *RoboCups* (in der auch während des Wettbewerbs eine Kamera über dem Spielfeld eingesetzt wird) werden dazu verschiedenfarbige Markierungen auf der Oberseite der Roboter angebracht. Dieses Prinzip hat sich dort bewährt und soll deshalb für unser Umfeld adaptiert werden.

5.4.2.1 Einschränkungen bei der Wahl der Marker

Bei der Wahl der Marker gelten in der Sony-four-legged-league allerdings einige Einschränkungen. Da alle wichtigen Objekte (Ball, Tor, Landmarken) farbkodiert sind, stehen nur sehr wenige Farben zur Verfügung, die genutzt werden können, ohne das Spielgeschehen zu beeinflussen. Die Roboter tragen die Teamfarben rot und blau, der Ball ist orange, die Tore hellblau und gelb und der Teppich grün. Zusätzlich wird von den Landmarken pink in Kombination mit gelb und hellblau verwendet (Abb. 5.8). Somit sind fast alle klar unterscheidbaren Farben bereits belegt und eine Markierung

mittels Kombination mehrerer Farben nicht möglich, ohne dadurch die Bildverarbeitung des Roboters zu beeinflussen.

Abbildung 5.8: Die verschiedenen Farben auf dem *RoboCup*-Spielfeld

Ein weiteres Problem stellt die Befestigung der Marker dar. Diese muss auf der einen Seite so stabil sein, dass der Marker während des Spielverlaufs an seiner Stelle bleibt. Auf der anderen Seite darf der Marker den Roboter nicht behindern und muss bei Bedarf möglichst rückstandsfrei wieder entfernt werden können. Auch der Ort der Befestigung ist problematisch. Da der Kopf sich sehr stark bewegen kann und es keinen Teil des Kopfes gibt, der ständig von oben sichtbar ist, scheidet dieser Bereich als Befestigungspunkt aus. Auch ein großer Bereich hinter dem Kopf ist nicht geeignet, da dieser durch den schrägen Blickwinkel zu den Randbereichen des Spielfeldes leicht vom Kopf verdeckt werden kann. Es steht also nur der hintere Bereich des Roboterkörpers zur Verfügung. Da der Marker weder weit über den Roboter hinaus ragen noch vom Kopf verdeckt werden soll, ist der zur Verfügung stehende Platz dort auf 14×10 cm beschränkt.

5.4.2.2 Rechteckige Marker in Teamfarbe

Die Farbe, die den geringsten Einfluss auf das Spielgeschehen darstellt, ist die jeweilige Teamfarbe. Die vorgegebenen Farben der Trikots sind ein dunkles Rot und ein dunkles Blau. Im ersten Entwurf für die Marker (Abb. 5.9) sollte nur der Platz auf dem Rücken des Roboters verwendet werden. Der Raum über den Hinterbeinen wurde also zunächst nicht genutzt. Daraus resultiert eine Maximalgröße der Marker von 10×7 cm. Zur Unterscheidung der verschiedenen Spieler einer Mannschaft sollte ein Schwarzweiß-Code verwendet werden. Die Fläche in Teamfarbe sollte in erster Linie dazu dienen, den Marker im Kamerabild zu lokalisieren und die Teamzugehörigkeit zu bestimmen. Zusätzlich sollte die farbige Fläche möglichst lang gezogen sein, um aus dem Flächenmoment die Ausrichtung des Roboters zu ermitteln. Entlang der rechten Seite der farbigen Fläche wurden schwarze und weiße Felder angeordnet, um die

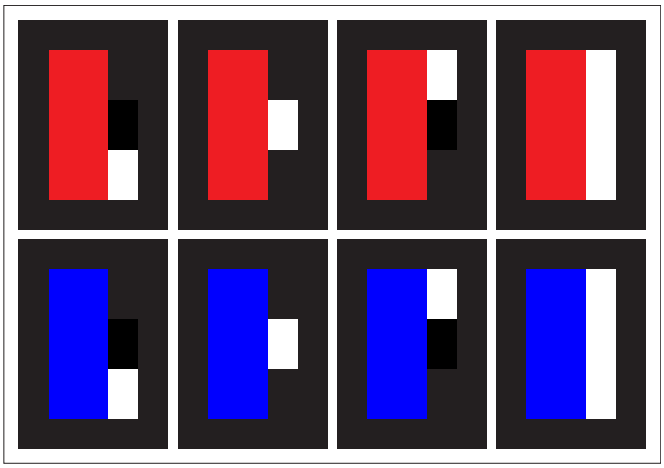

Abbildung 5.9: Entwurf rechteckiger Marker in Teamfarbe

Spielernummer innerhalb des Teams zu kodieren. Zur Kodierung der vier Roboter pro Mannschaft würden zwei Felder ausreichen. Um ungünstige Kombinationen wie einheitlich schwarz auszuschließen und die Verwechslungsgefahr zu minimieren, wurde aber ein drittes Feld hinzugefügt.

Der Roboterkörper ist in dem Bereich, in dem der Marker angebracht werden soll größtenteils weiß gefärbt. Um nicht irrtümlich diese weißen Flächen als Teil des Schwarzweiß-Codes zu interpretieren, wurde die farbige Fläche und der Schwarzweiß-Code zusätzlich mit einem schwarzen Rand umrahmt. Dadurch, dass nur auf einer Seite der Farbfläche nun weiße Bereiche angrenzen, sollte die Zweideutigkeit in der Ausrichtung, die durch die Spiegelsymmetrie der farbigen Fläche entsteht, aufgelöst werden.

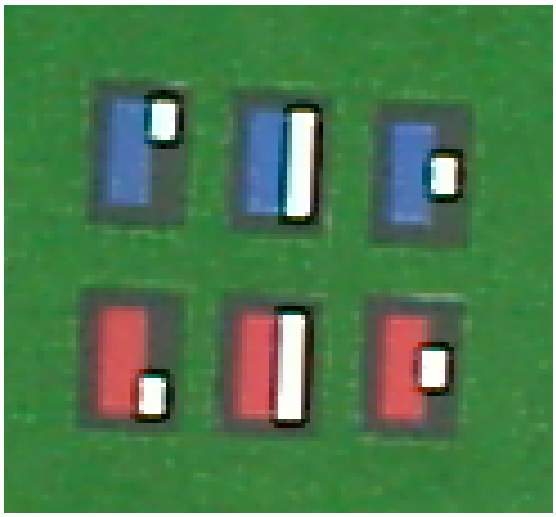

Abbildung 5.10: Erster Markerentwurf aus der Sicht der Deckenkamera

Mit einem Farbdrucker wurden Prototypen dieser Marker erstellt, um das Ergebnis der Überlegungen durch das Kameraauge zu betrachten. Da sofort klar wurde, dass diese Methode ein zu stark reflektierende Oberfläche erzeugt, wurde eine zweite Generation dieser Marker aus Tonpapier hergestellt. Nun waren die verschiedenen Flächen schon klarer zu erkennen, aber es traten noch immer einige Probleme auf. Bei Verwendung von Farben, die der Farbe der Trikots möglichst nahe kommt, erscheinen diese auf dem Kamarabild sehr dunkel und sind nur schwer von anderen dunklen Bereichen zu separieren. Außerdem bluten die weißen Bereiche stark in die umgebenden schwarzen Flächen aus (Abb. 5.10). Um zu verhindern, dass schwarze Teile der Kodierung und des Randes komplett überstrahlt werden, musste der Rand und der Schwarzweiß-Code sehr breit ausgelegt werden, so dass für die farbige Fläche nur noch ein Bereich von 5×3 cm zur Verfügung bleibt. Durch die stark ineinander laufenden Flächen werden im Kamerabild also nur noch wenige Pixel klar in der Teamfarbe dargestellt. Deshalb waren Schwierigkeiten beim Auffinden der Marker und vor allem beim Ermitteln einer möglichst exakten Ausrichtung des Roboters zu erwarten. Dies und die zu erwartende sehr komplexe Auswertung der Anordnung der Flächen auf dem Marker führten dazu, dass die Version der Marker fallen gelassen wurde.

5.4.2.3 Adaptierung eines Smallsize-Konzeptes

Da die Smallsize-Liga die größte Erfahrung im *RoboCup* mit Deckenkamerasystemen hat, wurde für den zweiten Markerentwurf ein Konzept dieser Liga an unsere Verhältnisse angepasst. Wie in Kapitel 3 beschrieben, eignet sich dafür am besten das System von 5dpo[4]. Dieses verwendet eine farbige Fläche in der Mitte des Markers zum Auffinden des Roboters und einen darum radial angeordneten Schwarzweiß-Code zur Ermittlung von Roboternummer und -orientierung.

Runder einfarbiger Marker zur Lokalisierung (Adaption des BallSpecialist) Nach den schlechten Erfahrungen mit den Teamfarben rot und dunkelblau, die vor allem durch ihre Dunkelheit schlecht zu erkennen sind, wurde für die einfarbige Fläche in der Mitte des Markers nun pinkes Tonpapier verwendet. Diese Farbe taucht zwar auch in den Landmarken in den Ecken des Spielfeldes auf, da sie dort aber immer in Kombination mit gelb oder hellblau verwendet werden, besteht kaum die Gefahr der Verwechslung für den Roboter. Durch das Auslösen des Landmarkenerkenners kann zwar zusätzliche Rechenzeit verbraucht werden; eine pinke Fläche ohne gelb oder hellblau in der Nähe, kann aber vom Roboter nicht einer bestimmten Landmarke zugeordnet werden, so dass die mit schwarz und weiß umgebenen Flächen des Markers auf dem Rücken von Mit- oder Gegenspielern nicht zu einer Fehllokalisierung des Roboters führen sollte. Auch für die Deckenkamera besteht keine Verwechslungsgefahr, da die pinken Flächen der Landmarken sich außerhalb des Spielfeldes befinden. Auf der anderen Seite ist diese stark gesättigte Farbe gut im Kamerabild zu erkennen und somit die Position der Marker zu bestimmen. Um diese Kreisflächen deutlich im Kamerabild erkennen zu können, wurde ein Durchmesser von 6 cm gewählt (Abb. 5.11).

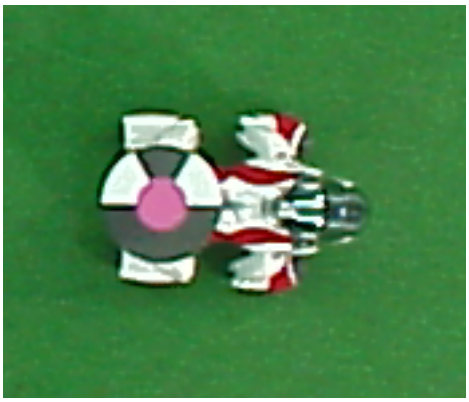

Abbildung 5.11: Zweite Markergeneration montiert auf einem Roboter aus Sicht der Deckenkamera

Das Lokalisieren einer einfarbigen kreisförmigen Fläche auf dem Spielfeld unterscheidet sich nicht wesentlich von der Aufgabe, ein kugelförmiges Objekt wie den Ball zu erkennen. Deswegen wurde das Verfahren zur Ballerkennung für diesen Teil der Roboterlokalisierung adaptiert. Im Wesentlichen musste dazu nur der Durchmesser des zu erkennenden Objektes und die Farbe desselben angepasst werden. Dabei stellte sich heraus, dass die Funktion, die für einen Farbwert die Ähnlichkeit zu dem Orangeton des Balles bestimmt, nicht direkt auf den Pinkton der Mittelflächen der Marker übertragbar ist. Die Ähnlichkeit zu Orange wird ermittelt, indem der Abstand zu einem idealen Orangeton im RGB-Farbraum bestimmt wird. Die Pinktöne der mittleren Markerfläche verteilen sich im Farbraum aber eher entlang einer geraden Linie und nicht kugelförmig um einen bestimmten Punkt (linker Teil von Abb. 5.12).

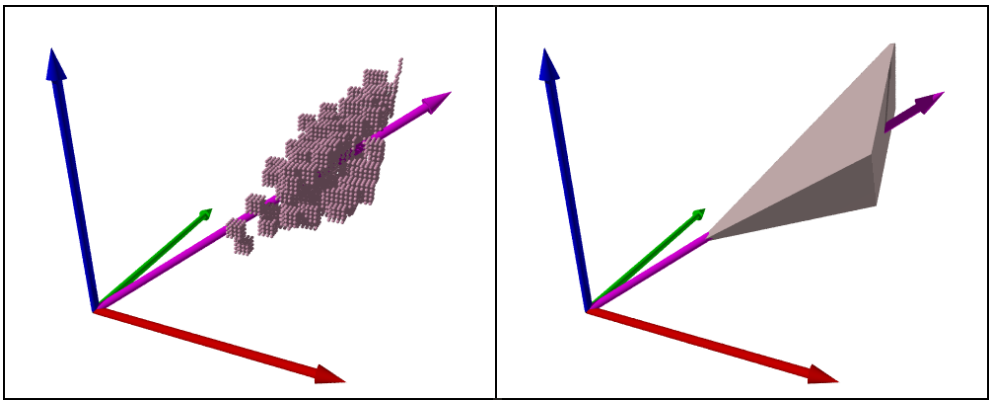

Abbildung 5.12: links: manuell erstellte Farbtabelle; rechts: Bereich mit „Pinkähnlichkeit“ über 100

Deshalb wurde zur Bestimmung der Ähnlichkeit zu Pink eine Funktion entwickelt, die grundsätzlich einen roten Farbanteil positiv und Abweichungen im Grün- und Blaukanal von dieser Geraden im Farbraum negativ bewertet. Durch die grundsätz-

liche positive Bewertung eines hohen Rotanteils werden gesättigtere Farbtöne höher bewertet als weniger gesättigte. Eine reine Bestimmung des Abstandes von der Geraden in Farbraum würde dagegen auch Farben in der Nähe von Schwarz als optimales Pink ansehen. Daraus ergibt sich für die Ähnlichkeitsfunktion folgende Formel, wobei Abweichungen im Grün- und Blaukanal doppelt bestraft werden:

$$f(r,g,b) = r - 2 * \left(\left| g - \frac{1}{2}\, r \right| + \left| b - \frac{5}{7}\, r \right| \right)$$

Der modifizierte Ballerkenner betrachtet dabei Werte oberhalb von 100 als Pink. Dies ergibt im RGB-Farbraum einen Bereich, wie er im rechten Teil von Abb. 5.12 zu sehen ist.

Damit werden die pinken Kreisflächen fast überall zuverlässig erkannt. Im Randbereich des Spielfeldes erscheint der pinke Teil des Markers allerdings im Gegensatz zum Ball nicht mehr als Kreis im Kamerabild, sondern stark gestreckt. Um die Marker auch in diesen Bereichen zuverlässig zu erkennen, werden die Randpunkte der Kreisfläche zuerst in Feldkoordinaten umgerechnet, und damit entzerrt, bevor der Mittelpunkt berechnet wird. Auch einige Plausiblitätstests werden im Unterschied zum Ballerkenner nicht in Bildkoordinaten, sondern in Feldkoordinaten durchgeführt. Nach diesen Anpassungen erkennt der Ballerkenner zuverlässig die Mittelpunkte der Marker, die verwendet werden, um im nächsten Schritt die Schwarzweiß-Codes, die die pinke Flächen umgeben, auszulesen.

Radialer Barcode zur Erkennung der Ausrichtung und zur Unterscheidung Damit die einzelnen schwarzen und weißen Felder auch dann noch gut zu erkennen sind, wenn die weißen Flächen im Kamerabild in die schwarzen Bereiche ausbluten, wurde für den gesamten Marker ein Durchmesser von 14 cm gewählt. Damit ragt der Marker seitlich über die Beine und rückwärtig über den Ansatz des abmontierten Schwanzes. Nach Abzug der farbigen Kreisfläche in der Mitte bleibt für den Schwarzweiß-Code also noch ein Ring von 4 cm Breite. Die Kodierung lehnt sich an das Prinzip des Teams 5dpo an. Es handelt sich also um sechs Sektoren zu jeweils 60°. Dadurch stehen neun verschiedene Codes zur Verfügung, die nicht durch Drehung in einen anderen Code übergeführt werden können und deren Ausrichtung eindeutig bestimmbar ist (Abb. 5.13). Zur Ausstattung des ersten (roten) Teams werden die vier Codes mit jeweils vier Schwarzweiß-Übergängen verwendet. Für das zweite (blaue) Team kommen vier der fünf Codes mit nur zwei Schwarzweiß-Übergängen zum Einsatz. Es stellte sich heraus, dass diese Codes keinen nennenswerten Nachteil gegenüber denjenigen mit mehr Übergängen aufweisen. Auf das Einführen einer weiteren Farbe als Markermitte zur Unterscheidung der beiden Teams konnte also verzichtet werden.

Um den Code auszulesen, wird der Marker in einem Abstand von 5 cm vom Mittelpunkt der pinken Kreisfläche in 5°-Schritten kreisförmig abgetastet. Eine feinere Abtastung ist nicht sinnvoll, da die Auflösung der Kamera dazu nicht ausreicht. Dieser Abtastkreis befindet sich also bei genauer Ermittlung des Mittelpunktes des Markers

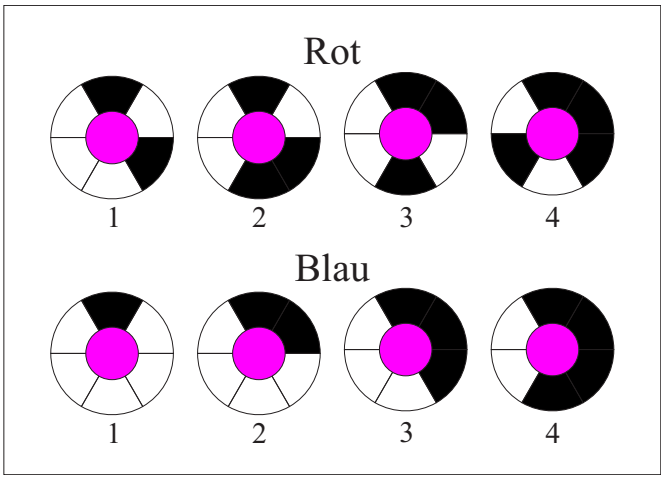

Abbildung 5.13: Zuordnung der Schwarzweiß-Codes zu den Spielernummern

genau in der Mitte des Rings, der durch die schwarzweißen Sektoren gebildet wird. Die Position der Scanpunkte wird dabei in Feldkoordinaten berechnet und dann mittels der in Kapitel 4 beschriebenen Funktionen in Pixelkoordinaten umgerechnet. Dadurch werden auch dann die richtigen Pixel ausgewählt, wenn der Marker durch die starke perspektivische Verzerrung im Randbereich des Spielfeldes stark gestreckt erscheint (Abb. 5.14).

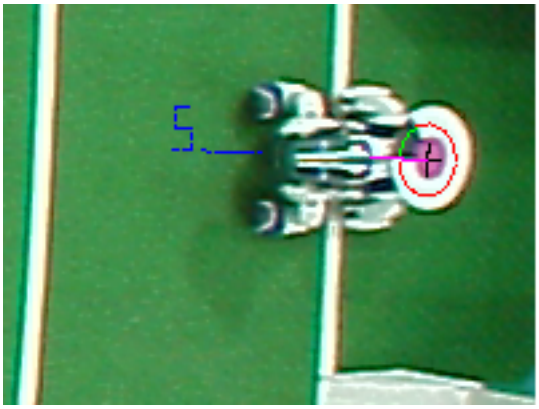

Abbildung 5.14: Verzerrter Marker in Tornähe mit angepasstem Scankreis

Für jeden dieser Pixel wird nun die Helligkeit berechnet, indem die Transformation in den YUV-Farbraum für den Y-Kanal vorgenommen wird:

$$Y = 0.2990 * R + 0.5870 * G + 0.1140 * B$$

Ab einem Helligkeitswert von 150 wird angenommen, dass sich der Pixel in einem weißen Sektor befindet. Bei einem Wert darunter wird der Pixel einem schwarzen Sektor zugeordnet. Die Abfolge der schwarzen und weißen Sektoren wird als Binärzahl

interpretiert. Dabei steht eine weiße Fläche für eine Null und eine schwarze Fläche für eine Eins. Ab dem zweiten Übergang zwischen weißen und schwarzen Pixeln wird bei jedem weiteren Übergang ermittelt, wie vielen Sektoren die Anzahl der zusammenhängenden Pixel entspricht. Danach wird die entsprechende Anzahl Nullen oder Einsen dem Code hinzugefügt. Nach 360° werden für den Bereich, in dem der Startpunkt liegt, die entsprechende Anzahl Sektoren hinzugefügt, die sich aus der Summe der Pixel seit dem letzten und bis zum ersten Übergang ergeben. Nachdem überprüft wurde, dass genau sechs Sektoren erkannt wurden, wird aus dem Binärcode und der Anzahl der Pixel bis zum ersten Schwarzweiß-Übergang die Spielernummer und die Ausrichtung des Roboters ermittelt.

Rot			Blau		
Spielernr.	*Code*	*Orientierung*	*Spielernr.*	*Code*	*Orientierung*
1	5	0	1	1	0°
	34	60		32	60°
	17	120		16	120°
	40	180		8	180°
	20	240		4	240°
	10	300		2	300°
2	13	0	2	3	0°
	38	60		33	60°
	19	120		48	120°
	41	180		24	180°
	52	240		12	240°
	26	300		6	300°
3	11	0	3	7	0°
	37	60		35	60°
	50	120		49	120°
	25	180		56	180°
	44	240		28	240°
	22	300		14	300°
4	23	0	4	15	0°
	43	60		39	60°
	53	120		51	120°
	58	180		57	180°
	29	240		60	240°
	46	300		30	300°

Tabelle 5.2: Zuordnung der Schwarzweiß-Codes zu Spielernummer und Winkel

Eine Spielernummer kann dabei durch sechs verschiedene Binärcodes repräsentiert werden, je nachdem, in welchem Sektor der Scankreis startet. Durch bis zu fünf zirkuläre Linksshifts wird derjenige dieser sechs Binärcodes ermittelt, der die kleinste

Zahl repräsentiert. Diese Zahl wird dann einer festgelegten Spielernummer zugeordnet (Tab. 5.2). Aus der Anzahl der Shifts, die zu dieser kleinsten Zahl führt, lässt sich schon mit einer Genauigkeit von 60° die aktuelle Ausrichtung des Roboters ermitteln. Eine Verbesserung auf eine 5°-Auflösung wird dadurch erreicht, dass aus der Anzahl der Pixel bis zum ersten Schwarzweiß-Übergang die Startposition innerhalb des ersten Bereichs ermittelt wird. Der so ermittelte Winkel entspricht der Position des ersten Pixels nach dem Übergang, der die Richtung nach vorne anzeigt. Um die Orientierung im Mittel näher an den tatsächlichen Schwarzweiß-Übergang zu legen, wird zum Schluss ein halber Scanschritt, also 2,5° abgezogen.

Nachdem so die Position des Markers und die Ausrichtung des Roboters ermittelt wurde, wird daraus die Position des Roboternullpunktes auf dem Feld berechnet. Dieser befindet sich direkt unter dem Kopfgelenk auf dem Boden. Die Position des Markermittelpunktes muss also um 16 cm in Richtung der ermittelten Orientierung des Roboters verschoben werden, um die Position des Roboternullpunktes auf dem Feld zu erhalten (Abb. 5.15).

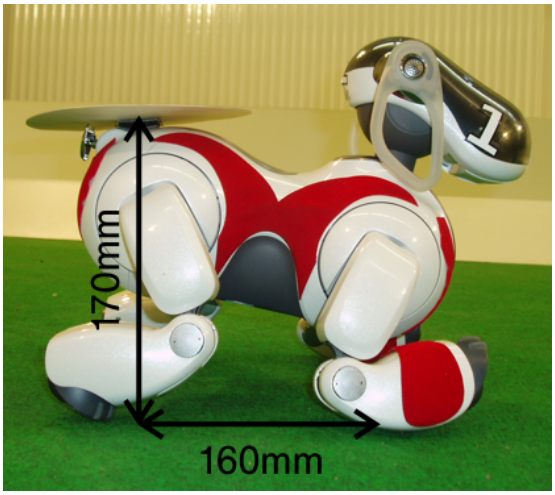

Abbildung 5.15: Translationen vom Markermittelpunkt zum Roboternullpunkt

5.4.3 Ergebnisse des Robotererkenners

Um die Genauigkeit des Robotererkenners zu bestimmen, wurden mehrere Messreihen aufgenommen. Um die Daten, die vom Deckenkamerasystem ermittelt wurden, überprüfen zu können, wurden dazu Positionen auf dem Feld gewählt, die sich leicht von Hand ausmessen lassen oder beim Anbringen der Feldlinien bereits ausgemessen wurden. Im Einzelnen sind dies die Eckpunkte der Strafräume, der Mittelpunkt sowie zwei Punkte auf der Mittellinie, die genau 1 m vom Mittelpunkt entfernt sind (Abb. 5.3, S. 27). An jeder dieser sieben Spielfeldpositionen wurden Messungen mit vier verschiedenen Ausrichtungen des Roboters gemacht. Um das manuelle Ausrich-

ten der Roboter zu vereinfachen und ein fehlerträchtiges Ausmessen der Winkel zu vermeiden, wurden dazu die vier Richtungen entlang der Feldlinien gewählt.

Somit sind die Sollpositionen und -orientierungen im Voraus bekannt und es müssen nur noch die Werte des Deckenkamerasystems erfasst werden. Hierzu wurde die im Kapitel 6 beschriebene Einbindung in *RobotControl* verwendet und insgesamt 28 Logdateien von jeweils gut 100 Messungen aufgezeichnet. Danach wurden diese Aufzeichnungen ins csv-Format exportiert und in einer Tabellenkalkulation analysiert. Für jede Position wurde so der mittlere absolute Fehler in der Bestimmung der Position und der Orientierung ermittelt. Außerdem wurde die Streuung der Messungen in Form der Standardabweichung wiederum für Position und Orientierung berechnet. Die auf diese Weise gewonnenen Ergebnisse sowie die entsprechenden Mittelwerte sind in Tabelle 5.3 dargestellt.

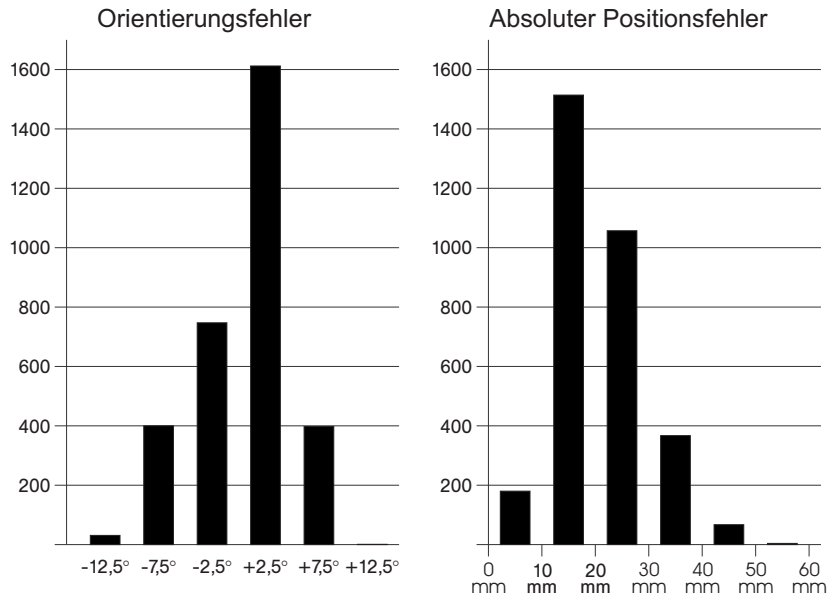

Abbildung 5.16: Verteilung von Orientierungs- und Positionsfehler

Bei der Betrachtung der Messwerte für die Fehler in der Orientierungsbestimmung ist zu beachten, dass diese nur in Schritten von 5° erfolgt. Eine feinere Auflösung ist aufgrund der Kameraauflösung nicht möglich. Außerdem muss beachtet werden, dass der Robotererkenner die vorgegebenen Sollorientierungen von 0°, 90°, 180° und 270° nicht exakt liefern kann, sondern nur um 2,5° versetzte Werte. Dies liegt daran, dass beim Auslesen des Schwarzweiß-Codes die Pixel auf diesen vier Positionen immer genau einem schwarzen oder weißen Sektor zugewiesen werden und der Übergang also nicht genau dort erfasst werden kann. Die Sollorientierungen liegen also immer genau in der Mitte zwischen zwei Rasterwerten, so dass die Fehler in den Messreihen immer mindestens 2,5° betragen. Vor diesem Hintergrund ist ein mittlerer Fehler von

			Position		*Orientierung*	
Pos	*Richtung*	*Messungen*	*Ø-Fehler*	*Stdaw.*	*Ø-Fehler*	*Stdaw.*
1	Nord	115	13,6 mm	1,8 mm	2,6°	0,7°
	Ost	120	19,8 mm	6,5 mm	6,0°	2,3°
	Süd	113	14,6 mm	10,9 mm	4,2°	3,3°
	West	113	12,2 mm	5,3 mm	3,1°	1,7°
2	Nord	109	11,4 mm	11,5 mm	3,5°	3,7°
	Ost	109	14,0 mm	4,6 mm	2,5°	2,0°
	Süd	113	18,7 mm	10,2 mm	3,7°	3,3°
	West	117	20,5 mm	13,7 mm	3,8°	4,9°
3	Nord	114	28,0 mm	5,1 mm	3,3°	1,8°
	Ost	115	14,1 mm	2,2 mm	2,6°	0,8°
	Süd	113	34,4 mm	6,9 mm	7,7°	2,0°
	West	115	29,4 mm	7,7 mm	4,5°	2,5°
4	Nord	113	34,1 mm	5,1 mm	3,0°	1,9°
	Ost	110	34,0 mm	6,1 mm	4,4°	2,4°
	Süd	113	25,8 mm	6,5 mm	6,3°	2,4°
	West	113	27,8 mm	10,7 mm	3,3°	3,8°
5	Nord	114	15,7 mm	2,9 mm	2,5°	1,2°
	Ost	113	14,1 mm	0,8 mm	2,5°	0,0°
	Süd	121	18,5 mm	1,6 mm	2,5°	0,5°
	West	113	11,9 mm	0,8 mm	2,5°	0,0°
6	Nord	114	22,2 mm	4,5 mm	2,8°	1,6°
	Ost	112	29,7 mm	5,7 mm	5,1°	2,5°
	Süd	114	10,0 mm	7,5 mm	4,3°	2,4°
	West	116	20,0 mm	6,8 mm	6,4°	2,1°
7	Nord	114	22,3 mm	7,0 mm	2,5°	2,5°
	Ost	114	15,5 mm	6,3 mm	2,6°	2,6°
	Süd	114	24,5 mm	1,2 mm	2,5°	0,0°
	West	115	14,4 mm	6,9 mm	7,0°	2,3°
Mittel		3189	20,4 mm	6,0 mm	3,8°	2,0°

Tabelle 5.3: Mittlere Fehler und Standardabweichungen des Robotererkenners

3,8° und eine mittlere Standardabweichung von 2,0° ein sehr gutes Ergebnis. Beide Werte liegen noch unterhalb der auflösungsbedingten Schrittweite von 5°. Aber auch die maximalen Fehler weichen nur maximal um 12,5° vom Sollwert ab (s. linkes Diagramm in Abb. 5.16). Das heißt, dass das Ergebnis des Deckenkamerasystems selbst im schlechtesten Fall nur 2,5 Scanschritte vom Sollwert entfernt liegt.

Bei der Betrachtung des Positionsfehlers fällt auf, dass die Standardabweichung fast immer deutlich unter dem mittleren Fehler liegt. Dies weist darauf hin, dass ein großer Teil des Fehlers, der im Mittel nur 20,4 mm beträgt, systematischer Natur ist. Ein Grund dafür könnten ungenau ausgemessene Feldlinien sein. Auf der anderen Seite ist es nur sehr schwer möglich, den Roboternullpunkt bei der Aufnahme der Messreihen exakt auf bestimmte Feldpunkte zu positionieren. Weiterhin besteht die Möglichkeit geringer Fehler in der Kalibrierung der Kamera, die dann zu einem Versatz der vom Kamerasystem ermittelten Positionen führt.

Ein großer Teil der Positionsfehler entsteht aber direkt aus den Orientierungsfehlern. Da der Roboternullpunkt durch Translation um 160 mm vom Mittelpunkt des Markers aus entlang der ermittelten Orientierung bestimmt wird, ergibt sich bei einem Fehler von 2,5° schon eine Verschiebung von ca. 7 mm. Da bei den gewählten Positionen der Orientierungsfehler wie oben beschrieben nicht unter diesem Wert liegen kann, erklärt dies auch die geringe Häufigkeit von absoluten Positionsfehlern im Bereich zwischen 0 und 10 mm (s. rechtes Diagramm in Abb. 5.16). Beim nächstgrößeren möglichen Orientierungsfehler von 7,5° beläuft sich der Einfluss auf die Positionierung schon auf ca. 22 mm. Damit erklärt sich ein großer Teil der Fehler bei der Positionsbestimmung bereits aus den Fehlern in der Orientierungserkennung.

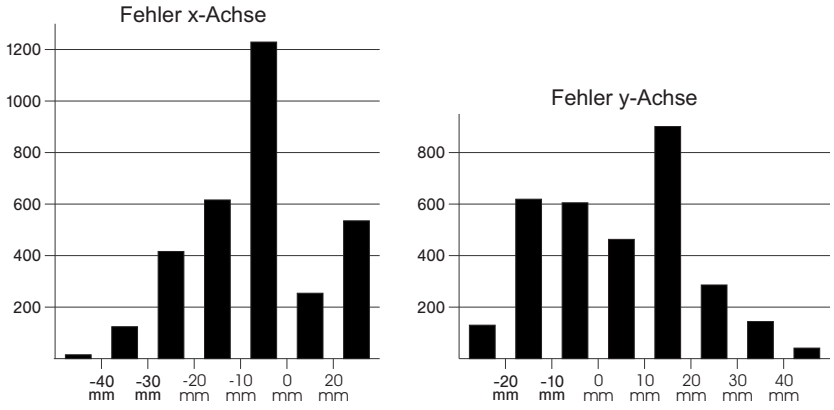

Abbildung 5.17: Verteilung von Positionsfehler in x- und y-Achse

Die Diagramme in Abb. 5.17 zeigen die Fehler bei der Bestimmung der Position aufgeschlüsselt nach x- und y-Achse. Dabei wird deutlich, dass die Fehler in der x-Achse etwas häufiger im negativen Bereich liegen, während die Fehler der y-Achse sich eher im positiven Bereich befinden. Diese Häufung auf einer Seite des Nullpunk-

tes könnten ein Hinweis darauf sein, dass evtl. die extrinsischen Parameter leicht verschoben gesetzt wurden.

Die Standardabweichung beträgt im Mittel nur 6,0 mm. Dieser geringe Wert weist darauf hin, dass die Erkennung der Marker im Bild sehr stabil ist. Ein Pixel des Kamerabildes entspricht auf Höhe des Spielfeldes zwischen 3 und 4 mm, so dass die Erkennung im Mittel also nur um wenige Pixel schwankt.

Insgesamt wurden während der 28 Messreihen 3202 Datenpakete vom Deckenkamerasystem empfangen. In 3189 davon wurde der Roboter auf dem Spielfeld korrekt erkannt. Dies entspricht einer zuverlässigen Erkennung in 99,59% aller Aufnahmen in den Bereichen, in denen die Messungen durchgeführt wurden. Die größten Fehler wurden bei der Bestimmung der Position festgestellt. Der maximale betragsmäßige Fehler lag dort bei gut 56 mm. Beim weitaus größten Teil der Messungen liegt der Betrag des Fehlervektors aber unter 30 mm (s. rechtes Diagramm in Abb. 5.16), wobei ein großer Teil dieses Fehlers auf Orientierungsfehler zurückzuführen ist. Die stärksten Fehler entstanden dabei auf den Positionen 3 und 4. Diese wurden auf der Strafraumlinie vor dem gelben Tor aufgezeichnet. Wie bereits in Abschnitt 5.3.2 erwähnt, konnten diese Fehler auf falsch positionierte Feldlinien und geringe Ungenauigkeiten bei der Kalibrierung der Kameraparameter zurückgeführt werden. Insgesamt liegen diese Fehler aber eine Größenordnung unter den mittleren Fehlern der Selbstlokatoren (siehe Abschnitt 7.1.7), so dass das Deckenkamerasystem zur Bewertung der Fehler der verschiedenen Verfahren zur Selbstlokalisierung herangezogen werden kann.

Kapitel 6

Einbindung in die Architektur des GermanTeam

Mit den in den letzten Abschnitten beschriebenen Verfahren zur Bildverarbeitung und Objekterkennung liegen dem Deckenkamerasystem Positionen und Geschwindigkeiten von bis zu acht auf dem Feld agierenden Robotern sowie des Balles vor. Diese Daten sollen zur Unterstützung der Entwicklung der Robotersoftware in die Debugumgebung des *GermanTeam RobotControl* übertragen werden.

6.1 Bereitstellung der Daten

In die Deckenkamerasoftware wurde eine Serverkomponente auf Basis verbindungsloser UDP Kommunikation integriert. Clients des Systems fordern durch die Zusendung eines Pakets Daten beim Server an. Alle in einem Frame vorliegenden Anforderungen werden mit einem Datenpaket, das neben den Zeitstempeln des Bildes und der letzten Synchronisierung die Positionen, Ausrichtungen und Geschwindigkeiten aller acht Roboter in globalen Koordinaten (sofern der entsprechende Roboter erkannt wurde) sowie Position und Geschwindigkeit des Balles enthält, beantwortet. Das Paket umfasst insgesamt 46 Werte in Form von Doubles, d.h. es ist daher insgesamt ca. 0,36 kB groß. Es werden in jedem Frame alle 46 Werte an jeden Client übertragen, der diese durch Zusendung eines Paketes an den Server beantragt hat. Dabei werden in diesem Frame nicht belegte Positionen, z.B. weil der betreffende Roboter oder der Ball nicht erkannt wurde, mit der Konstanten NaN (Not a Number) gekennzeichnet. Im Einzelnen ist die Struktur des Datenpakets in der Tabelle 6.1 spezifiziert.

Auf Seiten von *RobotControl* wurde mit der Klasse *RemoteCam* eine solche Clientanwendung realisiert. Die Steuerung erfolgt über die *RemoteCamToolBar* (Abb. C.1, S. 101), hierüber werden Adresse und Port des genutzten Deckenkamerasystems konfiguriert sowie die Anforderung der Pakete ein- bzw. wieder ausgeschaltet. Ebenfalls über die Toolbar kann das Deckenkamerasystem für die Zeitsynchronisie-

Offset	*Objekt*	*Wert*
0	Deckenkamera	Zeitstempel Bild
1		Zeitstempel Synchronisation
2	Ball	X-Position
3		Y-Position
4		X-Geschwindigkeit
5		Y-Geschwindigkeit
6	Roboter 1	X-Position
7		Y-Position
8		Orientierung
9		X-Geschwindigkeit
10		Y-Geschwindigkeit
⋮		
41	Roboter 8	X-Position
42		Y-Position
43		Orientierung
44		X-Geschwindigkeit
45		Y-Geschwindigkeit

Tabelle 6.1: Struktur des Datenpakets mit dem Weltbild der Deckenkamera

rung scharfgestellt werden, diese wird im folgenden Abschnitt ausführlicher beschrieben.

Alle Datenzugriffe erfolgen aus Sicht von *RobotControl* transparent, d.h. die Details der eigentlichen Kommunikation sind für alle Module und Dialoge in *RobotControl* nicht weiter relevant. Über die *RemoteCamToolBar* wird eine Roboter-ID eingestellt, die den Roboter identifiziert, bezüglich dessen *RobotControl* die Daten darstellen soll. Die Positionsdaten aus dem Deckenkamerasystem werden daraufhin in die in *RobotControl* gebräuchliche Form umgerechnet. Position und Ausrichtung des gewählten Roboters werden als *eigene* Position und Ausrichtung übernommen und die Daten über die übrigen Roboter werden entsprechend der Teamzugehörigkeit des ausgewählten Roboters als Teammitglieder, bzw. Gegenspieler verstanden.

Ähnlich dem Orakel aus dem in *RobotControl* integrierten Simulator kann nun beispielsweise im *FieldView* das Weltbild der Deckenkamera visualisiert werden. Kommuniziert *RobotControl* parallel dazu bereits mit einem realen Roboter auf dem Feld und werden von diesem Weltbilder angefordert, so ist schon in Echtzeit ein erster grober Vergleich der Daten möglich.

6.2 Zeitsynchronisierung

Zur genaueren Auswertung von Roboterdaten bezogen auf Daten vom Deckenkamerasystem ist eine Synchronisierung der Zeiten wünschenswert. Durch die unterschiedlichen Laufzeiten der Datenpakete vom Roboter und vom Deckenkamerarechner sowie die Verzögerungen in der Bildauswertung entsteht ein gewisser Zeitversatz zwischen den Weltbildern.

6.2.1 Detektion eines optischen Reizes

Um diesen Zeitversatz der Daten von Roboter und Deckenkamerasystem auszugleichen, wurde ein einfaches Verfahren zur Synchronisierung entwickelt. Über eine auf dem Spielfeld platzierte Glühlampe wird ein optischer Reiz erzeugt, der sowohl von den Robotern als auch der Deckenkamera erfasst werden kann. Sowohl für den Roboter als auch für die Deckenkamera wurden daher Algorithmen implementiert, die den Zeitpunkt des Einschaltens einer im Bild platzierten Lichtquelle anhand der rapide ansteigenden Helligkeit erkennen.

Die Bildverarbeitung des Deckenkamerasystems erfasst hierzu einen in 10 mm Schrittweite gerasterten Bereich von 20 cm mal 20 cm in der Mitte des Bildes und berechnet die Durchschnittshelligkeit. Da das Deckenkamerasystem im RGB-Farbraum arbeitet, wird als Maß für die Helligkeit der Mittelwert der drei Farbwerte verwendet. Wird das System scharf geschaltet, wird der Prozess initialisiert und die aktuelle Durchschnittshelligkeit als Referenzwert herangezogen. Sobald im weiteren Verlauf des Synchronisierungsvorgangs die mittlere Helligkeit um 30% des Differenzwertes zwischen maximal möglicher Helligkeit und Initialwert zugenommen hat, wird der Zeitstempel für die Synchronisierung auf die aktuelle Systemzeit des Deckenkamerasystems gesetzt. Dieser Zeitstempel wird daraufhin in allen neu erstellen Datenpaketen für die Clients, d.h. i.A. ein oder mehrere Rechner mit *RobotControl*, übertragen und die Synchronisierung ist erfolgreich beendet.

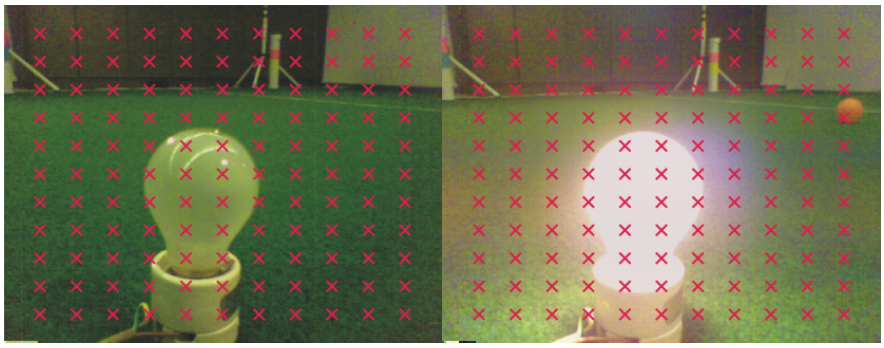

Abbildung 6.1: Raster zur Bestimmung der Durchschnittshelligkeit

Für den Roboter wurde ein ähnliches Verfahren als *SpecialVision*-Modul implementiert. In einem Raster, das als Schrittweite ein Zwölftel der Auflösung der Ro-

boterkamera verwendet, wird die durchschnittliche Helligkeit als Mittelwert des Luminanzkanals (Y) der Pixel des YUV-Bildes berechnet (Abb. 6.1). Als Schranke für den Synchronisierungszeitpunkt wird hierbei ebenfalls eine Erhöhung dieses Wertes um 30% der Differenz zwischen dem theoretischen Maximalwert und dem Initialisierungswert verwendet. Beim Überschreiten wird ein Synchronisierungszeitstempel mit der aktuellen Systemzeit des Roboters als Debug-Nachricht an ein mit dem Roboter per WLAN verbundenes *RobotControl* geschickt.

6.2.2 Synchronisation der Debugdaten

Der Zeitpunkt der Erfassung des optischen Signals wird also sowohl vom Roboter als auch der Deckenkamera mit einem Zeitstempel dokumentiert. Alle weiteren Datenpakete sind ebenfalls mit Zeitstempeln versehen. Durch Vergleich der Zeitstempel bei der Erfassung des Reizes ist der Offset zwischen Roboter und Deckenkamera bekannt und alle weiteren Pakete können demzufolge in einem gemeinsamen Bezugssystem betrachtet werden.

Aus praktischen Gründen erfolgt die Synchronisierung kompletter Logdateien offline, d.h. es werden zunächst die Daten von Roboter und Deckenkamera mit Versatz in einer Logdatei aufgezeichnet und anschließend die komplette Logdatei synchronisiert und die Datenpakete in ein gemeinsames Zeitsystem umgeschrieben. Solche synchronisierten Logdateien ermöglichen es beim Abspielen die aufgezeichneten Weltbilder zeitlich passender zueinander darzustellen als dies mit Zeitversatz „live“ möglich ist. Zudem werden die Synchronisationszeitpunkte natürlich auch bei der Auswertung kompletter Logdateien, etwa zur Analyse von Selbstlokatoren oder Ballmodellierungen herangezogen, um möglichst exakte Daten zu ermitteln.

Im Anhang C werden im Detail die Bedienung der *RemoteCamToolbar* sowie die Arbeitsschritte zum Aufzeichnen synchronisierter Logdateien mit Daten von Roboter und Deckenkamera beschrieben.

Kapitel 7

Beispielanwendung mit Ergebnissen

Nachdem die Daten des Deckenkamerasystems nahtlos in das Entwicklungswerkzeug *RobotControl* eingebunden wurden, kann nun einfach auf die Daten der Deckenkamera zugegriffen werden. Es steht also zu jeder Zeit ein weitgehend korrektes Modell des Spielgeschehens zur Verfügung, das auch in Logdateien aufgezeichnet werden kann. Darauf lassen sich leicht Verfahren aufbauen, die Situationen auf dem Spielfeld analysieren und mit den vom Roboter gelieferten Daten vergleichen.

7.1 Objektive Bewertung von Selbstlokatoren

Bei der Entwicklung verschiedener konkurrierender Ansätze zur Selbstlokalisierung im *GermanTeam* stellte sich immer wieder das Problem, dass die verschiedenen Module nur schwer verglichen werden konnten. Genaue Vergleichsdaten zu den Daten, die der Roboter liefert, ließen sich nur bei stehendem Roboter ausmessen. Bei einem Roboter in Bewegung war man auf das Augenmaß beim gleichzeitigen Blick auf Monitor und Spielfeld angewiesen. Es stand also keine Möglichkeit zur Verfügung, die Selbstlokatoren in Spielsituationen objektiv zu beurteilen und zu vergleichen. Dies führte oft dazu, dass eher bewährte Konzepte weiterverfolgt wurden, da neue Ansätze ihre Überlegenheit (oder Gleichwertigkeit bei geringerer Rechenzeit) nicht mit Fakten untermauern konnten. Außerdem bieten alle Ansätze zur Selbstlokalisierung eine Reihe von Parametern, die es zu optimieren gilt. Für den Vergleich verschiedener Parametersätze für den gleichen Lokalisierungsansatz stellen sich aber die gleichen oben beschriebenen Probleme. In einer ersten Beispielanwendung für das Deckenkamerasystem soll nun mit Hilfe von Logdateien die Güte eines Selbstlokalisierungsansatzes objektiv bewertet werden und mit Hilfe weniger Zahlenwerte erfasst werden. Alle Schritte zu einer solchen Analyse sollen dabei vollständig in der Entwicklungsumgebung *RobotControl* durchführbar sein.

7.1.1 Verfahren zum Aufzeichnen einer auszuwertenden Logdatei

Der erste Schritt zur Ermittlung der Güte eines Selbstlokalisierungsansatzes ist das Aufzeichnen einer Logdatei, die sowohl die vom Roboter berechneten Positionen als auch die Referenzdaten des Deckenkamerasystems enthält. Für die hier vorgestellte Beispielanwendung genügt es, sich die sogenannten *Worldstates* vom Roboter schicken zu lassen, da diese alle benötigten Daten enthalten. Dabei reicht es aus, diese nur ca. alle 130 ms zu übermitteln, da die Vergleichsdaten der Deckenkamera aufgrund deren Framerate von 7,5 Hz nur alle ca. 133 ms übertragen werden. Um später die Zeitstempel der beiden Datenquellen vergleichen zu können, wird außerdem von beiden Systemen der Zeitstempel zum Synchronisierungszeitpunkt benötigt. Das genaue Vorgehen bei einer solchen Aufzeichnung wird in Abschnitt C.2 beschrieben.

Damit die Deckenkamera die Positionen der Roboter erfassen kann, müssen natürlich zuvor die Marker auf dem Rücken der Roboter befestigt werden. Während der Aufzeichnung sollte man darauf achten, dass das Geschehen auf dem Spielfeld möglichst den Situationen entspricht, die man analysieren will. Möchte man also die Güte der Selbstlokalisierung während eines normalen Spieles analysieren, sollte man möglichst während der Aufzeichnung auch ein normales Spiel durchführen. Die Analyse einer auf diese Weise aufgezeichneten Logdatei startet man nun, indem man die Logdatei mit der RemoteCam-Toolbar öffnet.

7.1.2 Grafische Darstellung der Roboterbewegung

Zu Beginn der Analyse werden die Daten aus der Logdatei von den beiden unterschiedlichen Quellen getrennt gespeichert. Dazu werden für die Roboterpositionen vom Deckenkamerasystem und vom Roboter selbst getrennte Listen angelegt. Außerdem werden die jeweils letzten Synchronisierungszeitstempeln der beiden Datenquellen gespeichert. Die Differenz dieser beiden Zeitstempel wird dazu genutzt die Zeitstempel in den Positionsdaten von der Deckenkamera an die Systemzeit des Roboters anzugleichen. Bei Verwendung einer synchronisierten Logdatei entfällt dieser Schritt.

Danach werden die gespeicherten Informationen zuerst grafisch aufbereitet. Dazu werden in die Feldansicht von *RobotControl* der Positionsverlauf des Roboters eingezeichnet. Der Verlauf, der aus den Daten der Deckenkamera erzeugt wird, wird dabei durch eine dünne hellblaue Linie kenntlich gemacht. Die Daten vom Roboter selbst werden dunkelblau gefärbt. Dabei gibt die Stärke der Linie zusätzlich an, wie sicher sich der Roboter war, die richtige Position errechnet zu haben. Eine sehr dünne dunkelblaue Linie symbolisiert also eine hohe Unsicherheit der Selbstlokalisierung des Roboters, während eine dickere Linie für starke Zuversicht in die berechnete Position steht (Abb. 7.1).

Diese visuelle Darstellung erlaubt schon eine gute Einschätzung der Differenz der beiden Positionsverläufe, wenn sich der Roboter während der Aufzeichnung nicht sehr weit fortbewegt hat. Um zu einer realistischen Einschätzung der Qualität eines Selbstlokalisierungsverfahrens während eines *RoboCup*-Spiels zu kommen, ist es aber nötig

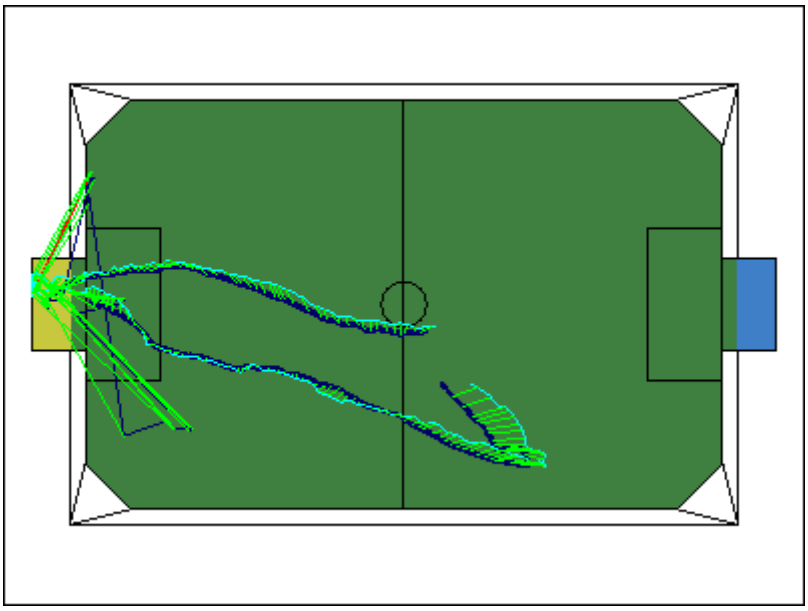

Abbildung 7.1: Grafische Darstellung einer kurzen Roboterbewegung mit starken Positionierungsfehlern im Tor (helle Linie: Deckenkameradaten; dunkele Linie: Daten des Roboters, Linienbreite abhängig von Validität; Verbindungslinien: Korrespondenz zwischen Datenpunkten)

einen längeren Abschnitt eines Spieles aufzuzeichnen. Ein solcher Verlauf ist allerdings auf einem einzelnen Abbild des Spielfeldes nicht mehr übersichtlich darstellbar.

7.1.3 Berechnung passender Deckenkamera-Daten zur Roboter-Selbstlokalisierung

Um die Genauigkeit eines Selbstlokalisierungsansatzes nun in Zahlenwerten erfassen zu können, muss zuerst der Fehler für jede einzelne aufgezeichnete Roboterposition ermittelt werden. Dazu benötigt man den entsprechenden Referenzwert vom Deckenkamerasystem. Da beide Systeme aber ihre Daten selten genau zum gleichen Zeitpunkt liefern, wird für jede Position, die vom Roboter aufgezeichnet wurde eine korrespondierende Position des Deckenkamerasystems für diesen Zeitpunkt interpoliert. Hierfür werden zuerst diejenigen Positionen von der Liste der Deckenkamera herausgesucht, die direkt vor (Zeitstempel t_1, Position $\vec{p_1}$, Orientierung α_1) und nach (t_2, $\vec{p_2}$, α_1) dem Zeitstempel des zurzeit betrachteten Roboterdatensatzes liegen. Falls sich dabei herausstellt, dass alle Daten der Deckenkamera weiter als 100 ms von dem Zeitstempel dieser Roboterdaten entfernt sind, wird dieser Datensatz verworfen. Andernfalls wird aus den beiden zeitlich angrenzenden Datensätzen der Deckenkamera für den Zeitpunkt t_i, zu dem die Daten vom Roboter aufgezeichnet wurden, eine Position $\vec{p_i}$ und Orientierung α_i des Roboters linear interpoliert.

$$\vec{p_i} = \frac{t_2 - t_i}{t_2 - t_1}\vec{p_1} + \frac{t_i - t_1}{t_2 - t_1}\vec{p_2}$$
$$\alpha_{temp} = \frac{t_2 - t_i}{t_2 - t_1}\alpha_1 + \frac{t_i - t_1}{t_2 - t_1}\alpha_2$$

Dabei wird darauf geachtet, dass sich die interpolierte Orientierung innerhalb des spitzen Winkels zwischen den Ausgangswinkeln befindet.

$$\alpha_i = \begin{cases} \alpha_{temp} - \pi & \text{falls } |\alpha_1 - \alpha_2| > \pi \wedge \alpha_{temp} > 0 \\ \alpha_{temp} + \pi & \text{falls } |\alpha_1 - \alpha_2| > \pi \wedge \alpha_{temp} \leq 0 \\ \alpha_{temp} & \text{sonst} \end{cases}$$

Zur Illustrierung des Fehlers wird eine grüne Verbindungslinie zwischen der Roboterposition und der interpolierten Position in die Feldansicht von *RobotControl* eingezeichnet (Abb. 7.1).

7.1.4 Grafische Darstellung der Verteilung des Positionierungsfehlers

Um auch die Analyse längerer Logdateien darstellen zu können, wurde eine zweite Analyseausgabe in die Feldansicht implementiert. Diese kann entweder zusätzlich zu den Bewegungslinien oder einzeln angezeigt werden. Bei dieser zweiten Visualisierung werden nicht die eigentlichen Positionen dargestellt, sondern nur die Fehler, also die Differenzen zwischen Daten des Roboters und Daten des Deckenkamerasystems. Dieser Differenzvektor wird für jede verglichene Position an den Feldmittelpunkt angesetzt und an das Ende ein pinker Punkt gezeichnet. Arbeitet die Selbstlokalisierung zuverlässig, so konzentriert sich die Wolke der pinken Punkte um den Mittelkreis. Treten hingegen viele grobe Fehler bei der Berechnung der eigenen Position auf, so erscheinen diese Punkte weit über das Spielfeld verstreut (Abb. 7.2).

Zusätzlich zu dieser Darstellung der Positionsfehler können auch die Fehler in der Bestimmung der Orientierung mit dieser zweiten Ansicht visualisiert werden. Dazu werden die Fehler in der Orientierungsbestimmung des Roboters auf einem Kreis um den Feldmittelpunkt mittels gelber Punkte eingezeichnet. Dabei deutet eine Konzentration der Punkte auf einem schmalen Sektor des Kreises in Spielrichtung des Roboters auf geringe Fehler bei der Bestimmung der Ausrichtung hin. Sind die Punkte hingegen weit über den Kreis verstreut, so weiß der Roboter häufig nicht, in welche Richtung er sich gerade bewegt.

7.1.5 Berechnung von mittlerem Fehler und Standardabweichung

Zur Berechnung des mittleren Fehlers und der Standardabweichung wird zu jeder vom Roboter gelieferten Position und Orientierung bei zeitlich nahe liegenden Daten vom Deckenkamerasystem der aktuelle Fehler berechnet. Der Positionsfehler wird dabei

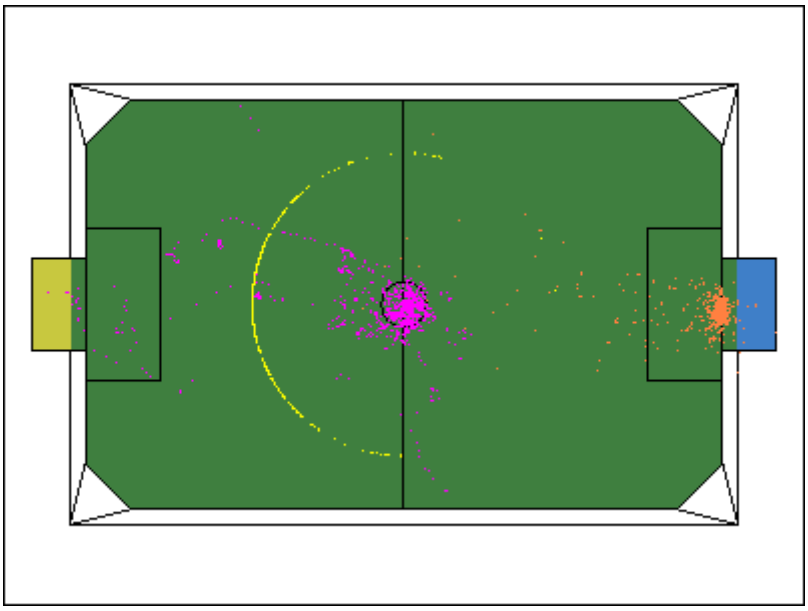

Abbildung 7.2: Visualisierung von Positionierungs- und Orientierungsfehlern mit Punktwolken

durch den Vektor $\vec{f}_i$ zwischen der vom Roboter errechneten und vom Deckenkamerasystem bereitgestellten Position repräsentiert. Über diese Vektoren selbst, deren Betrag und Quadrat wird jeweils eine Summe zur späteren Verrechnung gebildet.

Zur Ermittlung des Fehlers in der Orientierung wird zunächst die Differenz δ_i zwischen dem vom Roboter gelieferten und vom Deckenkamerasystem gesendeten Winkel gebildet. Diese Differenz wird anschließend auf den Bereich zwischen $-\pi$ und $+\pi$ normalisiert. Auch für die Orientierung werden die Einzelfehler, deren Betrag und Quadrat aufsummiert. Außerdem wird die Anzahl n solcher Vergleiche gezählt.

Nachdem dies für alle in der Logdatei zur Verfügung stehenden Daten durchgeführt wurde, werden zunächst die mittleren Fehler der Position $\bar{f}$ und Orientierung $\bar{\delta}$ durch Division der Betragssummen durch die Anzahl der Vergleiche berechnet.

$$\bar{f} = \frac{1}{n} \sum_{i=1}^{n} |\vec{f}_i|$$

$$\bar{\delta} = \frac{1}{n} \sum_{i=1}^{n} |\delta_i|$$

Im nächsten Schritt werden aus den aufsummierten Fehlern die Erwartungswerte für den Positions- $E(\vec{f})$ und Orientierungsfehler $E(\delta)$ ermittelt.

$$E(\vec{f}) = \frac{1}{n} \sum_{i=1}^{n} \vec{f}_i$$

$$E(\delta) = \frac{1}{n}\sum_{i=1}^{n}\delta_i$$

Auf gleiche Weise werden auch die Erwartungswerte der quadratischen Fehler gebildet.

$$E(|\vec{f}|^2) = \frac{1}{n}\sum_{i=1}^{n}|\vec{f}_i|^2$$

$$E(\delta^2) = \frac{1}{n}\sum_{i=1}^{n}\delta_i^2$$

Daraus werden wiederum die Standardabweichungen der Fehler in der Position σ_P und der Orientierung σ_R berechnet.

$$\sigma_P = \sqrt{E(|\vec{f}|^2) - E(\vec{f})^2}$$

$$\sigma_R = \sqrt{E(\delta^2) - E(\delta)^2}$$

Die mittleren Fehler, die Erwartungswerte sowie die Standardabweichungen für Position und Orientierung werden nach der Analyse in der *Message console* ausgegeben (Abb. 7.3). Diese Zahlenwerte können nun als objektiver Bewertungsmaßstab unterschiedlicher Selbstlokalisierungsansätze dienen oder zur Optimierung verschiedener Parameter eines Ansatzes herangezogen werden.

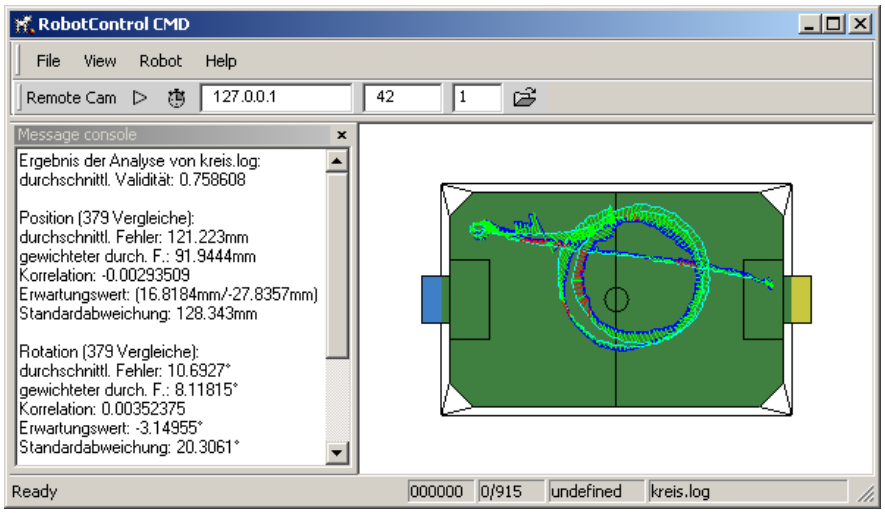

Abbildung 7.3: Ausgabe einer Logdateianalyse

7.1.6 Einbeziehung von Validitäten

Die verschiedenen Methoden zur Selbstlokalisierung berechnen neben Position und Orientierung des Roboters noch zusätzlich eine sogenannte Validität der aktuell gelieferten Lokalisierung. Dieser Wert v_i zwischen 0 und 1 gibt an, wie zuverlässig die aktuell ermittelte Position ist. Falls ein Lokalisierungsalgorithmus die Position nur sehr vage bestimmen konnte, geht dieser Wert gegen 0. Konnte die Position hingegen mit hoher Sicherheit ermittelt werden, so nähert sich die Validität dem Wert 1.

Um diesen zusätzlichen Informationen der Selbstlokatoren Rechnung zu tragen und um die Qualität dieser Informationen zu bestimmen, werden weitere Beurteilungskriterien berechnet. Als erstes wird neben dem mittleren Fehler auch der mittlere *gewichtete* Fehler $\bar{f}_g$bestimmt. Dazu wird der Betrag des Fehlers mit dem Validitätswert multipliziert, bevor der Mittelwert gebildet wird.

$$\bar{f}_g = \frac{1}{n} \sum_{i=1}^{n} |\vec{f}_i| v_i$$

Dadurch fallen Fehler weniger ins Gewicht, falls gleichzeitig die Information zur Verfügung steht, dass die Lokalisierung zurzeit nicht zuverlässig ist. Behauptet die Selbstlokalisierung hingegen, dass die von ihr gelieferten Werte sicher sind, spielen trotzdem auftretende Fehler eine größere Rolle.

Dieser Wert muss aber mit Vorsicht betrachtet werden, da für Selbstlokatoren, die grundsätzlich sehr niedrige Werte angeben, dadurch auch ein niedriger mittlerer gewichteter Fehler errechnet wird. Deswegen wird zusätzlich auch der Mittelwert der Validitäten angegeben.

$$\bar{v} = \frac{1}{n} \sum_{i=1}^{n} v_i$$

Um weiterhin beurteilen zu können, wie gut die übermittelten Validitäten zu dem tatsächlichen aktuellen Fehler passen, wird als letztes Beurteilungskriterium der Korrelationskoeffizient zwischen diesen beiden Werten ermittelt. Dazu werden zunächst die Standardabweichungen von Validität σ_v und betragsmäßigem Fehler $\sigma_{|\vec{f}|}$ berechnet.

$$\sigma_v = \sqrt{\frac{1}{n} \sum_{i=1}^{n} v_i^2 - \bar{v}^2}$$

$$\sigma_{|\vec{f}|} = \sqrt{E(|\vec{f}|^2) - \bar{f}^2}$$

Daraus ergibt sich der Korrelationskoeffizient $\varrho(|\vec{f}|, v)$ folgendermaßen:

$$\varrho(|\vec{f}|, v) = \frac{\bar{f}_g - \bar{f} * \bar{v}}{\sigma_v * \sigma_{|\vec{f}|}}$$

Eine gute Übereinstimmung von Validitätsinformationen und tatsächlichem Fehler resultiert dabei in einem negativen Korrelationskoeffizient, da bei großen Fehlern der Validitätswert niedrig sein sollte und umgekehrt. Ein Wert von -1 spiegelt also eine optimale Korrelation wieder.

Analog zu den obigen Berechnungen erfolgt auch für die Orientierung die Ermittlung von mittlerem gewichtetem Fehler und Korrelationskoeffizient. Diese zusätzlichen Analysewerte werden zusammen mit den in 7.1.5 erwähnten Werten in der *Message console* ausgegeben (Abb. 7.3).

7.1.7 Ergebnisse bei verschiedenen Selbstlokatoren

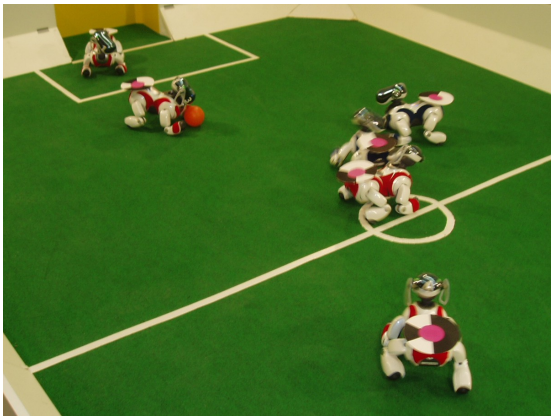

Abbildung 7.4: Testspiel mit Markern

Im Folgenden soll nun getestet werden, wie gut sich das vorgestellte Analyseverfahren zum Beurteilen verschiedener Selbstlokatoren eignet. Außerdem soll festgestellt werden, welche Aussagekraft die verschiedenen berechneten Werte bei der Bewertung der verschiedenen Verfahren haben.

Um die unterschiedlichen Ansätze zur Selbstlokalisierung vernünftig beurteilen zu können, müssen die Verfahren in der Umgebung getestet werden, in der sie später auch zum Einsatz kommen sollen. Zur Aufzeichnung der verschiedenen Logdateien müssen also reale Spiele durchgeführt werden. Dabei sollten alle Roboter Marker tragen, um möglichst viele Logdateien gleichzeitig aufzuzeichnen und den Einfluss anderer Roboter später in den Logdateien erkennen zu können (Abb. 7.4). In einem solchen Spiel kommt es zu vielen unterschiedlichen Situationen, in denen die Güte der den Selbstlokatoren zur Verfügung stehenden Daten sehr starken Schwankungen unterliegt. Bei einem Kampf um den Ball am Rande des Spielfeldes stehen kaum Daten zur Selbstlokalisierung zur Verfügung. Steht der Roboter hingegen weitgehend unbedrängt auf dem Feld, so werden viele Landmarken und Feldlinien gesehen, die zu einer guten Lokalisierung führen können. Um solche äußeren Einflüsse möglichst gut auszugleichen, dauerte jede Aufzeichnung mindestens sechs Minuten. Dies klingt nicht nach

Lokator	*Spieler*	*Vergl.*	*Ø-Valid.*	*Ø-Fehler*	*Ø-gew. F*	*Korr*	*Stdaw*
Single	Blau 3	2231	24,4%	411 mm	101 mm	0,01	308 mm
Landmark	Blau 4	2476	23,4%	310 mm	63 mm	-0,34	352 mm
	Rot 3	1997	9,1%	845 mm	35 mm	-0,32	1370 mm
	Rot 4	2155	21,8%	438 mm	83 mm	-0,28	521 mm
GT 2003	Blau 3	2095	65,0%	380 mm	186 mm	-0,65	483 mm
	Blau 4	2199	71,9%	333 mm	205 mm	-0,58	489 mm
	Rot 3	2401	66,4%	299 mm	174 mm	-0,36	388 mm
	Rot 4	1942	75,2%	328 mm	201 mm	-0,6	571 mm
Fusion 2003	Blau 3	2047	67,9%	276 mm	177 mm	-0,27	320 mm
	Blau 4	2126	75,1%	326 mm	239 mm	-0,26	339 mm
	Rot 3	2151	69,8%	331 mm	163 mm	-0,69	548 mm
	Rot 4	2352	75,8%	233 mm	167 mm	-0,36	284 mm
GT 2004	Blau 3	2323	64,1%	261 mm	155 mm	-0,34	324 mm
	Blau 4	1862	51,9%	352 mm	80 mm	-0,60	654 mm
	Rot 3	2023	57,4%	516 mm	204 mm	-0,50	949 mm
	Rot 4	2119	49,7%	252 mm	91 mm	-0,47	346 mm
	Rot 1	3318	71,5%	124 mm	47 mm	-0,57	389 mm
	Rot 2	3290	57,3%	337 mm	153 mm	-0,35	583 mm
	Rot 3	3431	59,9%	268 mm	105 mm	-0,52	523 mm
	Rot 4	3535	55,1%	663 mm	150 mm	-0,62	1269 mm

Tabelle 7.1: Ergebnisse verschiedener Selbstlokatoren (Position)

einer langen Zeitspanne, führt aber schon zu weiteren Problemen. Im normalen Spielverlauf kommt es immer wieder vor, dass Roboter neu gestartet werden müssen. In einem solchen Fall ist dann keine zusammenhängende Aufzeichnung mehr möglich und die Aufzeichnung muss wiederholt werden, da der neugestartete Roboter neu mit dem Deckenkamerasystem synchronisiert werden muss.

Zur Bewertung wurden vier Selbstlokatoren ausgewählt, die in den letzten Jahren im *GermanTeam* entwickelt wurden. Den einfachsten Ansatz stellt *Single Landmark* dar. Er beruht darauf, die aktuelle Position in jedem Durchlauf durch einzelne neu erkannte Landmarken oder Tore zu korrigieren. *GT 2003* ist der im Jahr 2003 vom *GermanTeam* verwendete Ansatz. Er verwendet eine MonteCarlo-Lokalisierung, deren Partikel anhand der gesehenen Landmarken und Feldlinien beurteilt werden. Eine Kombination dieser beiden Verfahren wurde mit *Fusion 2003* realisiert. Dabei wurden die Partikel, die beim Sensor Resetting neu in *GT 2003* eingefügt wurden, durch einen parallel laufenden *Single Landmark*-Lokator erzeugt. Die Details zu diesen drei Ansätzen finden sich im Teamreport des *GermanTeam* aus dem Jahr 2003 [8]. Im Weltmeisterschaftsjahr 2004 wurde vom *GermanTeam* der Lokator *GT 2004* verwendet. Dieser ist eine Weiterentwicklung von *GT 2003*, bei der vor allem die Feldlinien eine grö-

ßere Rolle spielen. Eine Beschreibung des Weltmeisterschaftscodes des *GermanTeam* findet man im Teamreport des Jahres 2004 [9].

Für jedes dieser Verfahren wurden vier Logdateien von vier verschiedenen Feldspielern aufgezeichnet. Für das aktuelle Verfahren *GT 2004* wurden zusätzlich weitere vier Logdateien aufgezeichnet, bei denen alle vier Roboter eines Teams über eine komplette Halbzeit beobachtet wurden.

Lokator	*Spieler*	*Vergl.*	*Ø-Valid.*	*Ø-Fehler*	*Ø-gew. F*	*Korr*	*Stdaw*
Single Landmark	Blau 3	2231	24,4%	17,5°	3,5°	-0,28	26,4°
	Blau 4	2476	23,4%	18,8°	2,7°	-0,41	34,0°
	Rot 3	1997	9,1%	32,1°	1,3°	-0,34	51,2°
	Rot 4	2155	21,8%	19,2°	2,4°	-0,46	33,8°
GT 2003	Blau 3	2095	65,0%	19,9°	8,4°	-0,55	38,4°
	Blau 4	2199	71,9%	13,8°	9,0°	-0,28	24,6°
	Rot 3	2401	66,4%	20,2°	10,8°	-0,34	37,4°
	Rot 4	1942	75,2%	18,0°	12,1°	-0,28	35,0°
Fusion 2003	Blau 3	2047	67,9%	16,2°	8,3°	-0,45	32,3°
	Blau 4	2126	75,1%	11,0°	8,1°	-0,07	17,6°
	Rot 3	2151	69,8%	15,7°	8,8°	-0,40	28,9°
	Rot 4	2352	75,8%	13,9°	9,2°	-0,37	27,2°
GT 2004	Blau 3	2323	64,1%	13,5°	7,7°	-0,25	28,1°
	Blau 4	1862	51,9%	22,3°	6,3°	-0,53	39,3°
	Rot 3	2023	57,4%	20,6°	9,1°	-0,38	37,7°
	Rot 4	2119	49,7%	24,3°	6,6°	-0,54	41,4°
	Rot 1	3318	71,5%	9,8°	5,4°	-0,43	21,6°
	Rot 2	3290	57,3%	20,1°	9,1°	-0,37	33,5°
	Rot 3	3431	59,9%	16,8°	6,6°	-0,53	32,2°
	Rot 4	3535	55,1%	26,4°	8,1°	-0,53	48,8°

Tabelle 7.2: Ergebnisse verschiedener Selbstlokatoren (Orientierung)

Die Ergebnisse der Analysen dieser 20 Logdateien sind in den Tabellen 7.1 und 7.2 aufgeführt. Bei Betrachtung dieser Daten fällt sofort auf, dass die Schwankungen zwischen den verschiedenen Logdateien eines Lokalisierungsansatzes stärker ausfallen, als zwischen den unterschiedlichen Verfahren. Bei den Positionswerten schwankt der mittlere absolute Fehler zwischen 124 mm und 845 mm, die Standardabweichung zwischen 284 mm und 1370 mm. Bei den Daten für die Orientierungsfehler liegt der durchschnittliche absolute Fehler zwischen 9,8° und 32,1°, die Standardabweichung zwischen 17,6° und 51,2°.

Die Spielsituationen haben also über diese Zeiträume einen größeren Einfluss auf die Qualität der Lokalisierung als die Unterschiede zwischen den verschiedenen Ansätzen. Um zu visualisieren, wie sich die Fehler über den Spielverlauf verteilen, wurden die Einzelfehler jedes Vergleichs sowie die jeweiligen Validitäten in CSV-Dateien

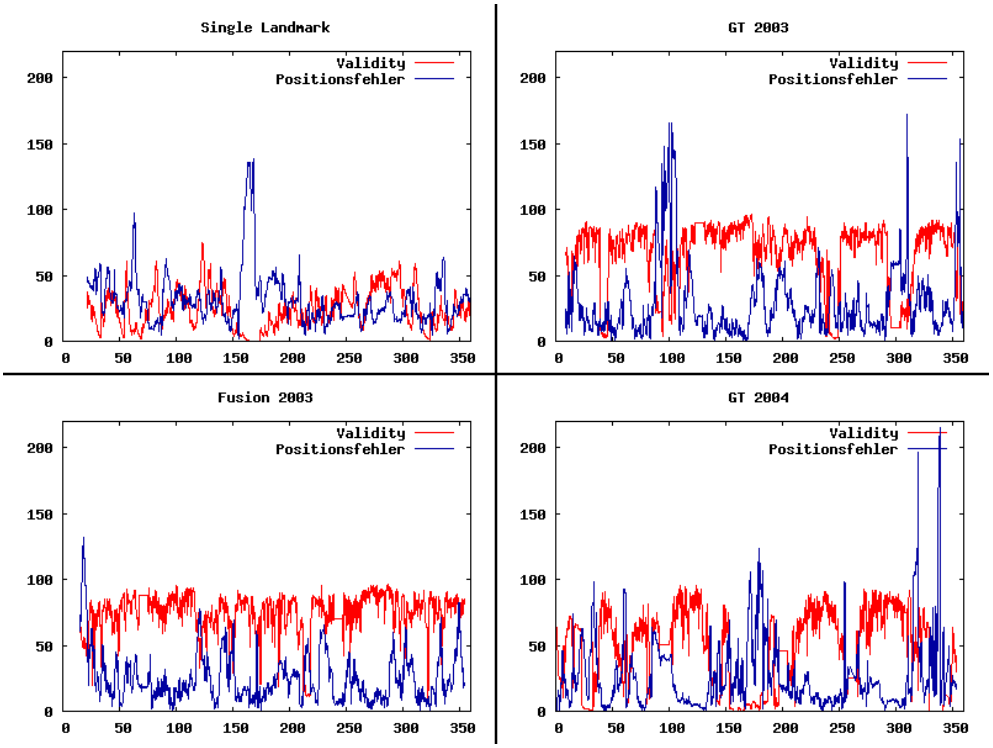

Abbildung 7.5: Diagramme der Logdateien mit den jeweils besten Positionsergebnissen der verschiedenen Selbstlokatoren

ausgegeben und geplottet. Abb. 7.5 zeigt diese Diagramme für die Logdateien mit den jeweils besten Positionsergebnissen für die verschiedenen Verfahren. Auf der x-Achse ist jeweils der Zeitverlauf in Sekunden angegeben. Auf der y-Achse wurde der Positionsfehler in Zentimeter und die Validität in Prozent aufgetragen, so dass beide Werte in einer ähnlichen Größenordnung verlaufen. Beim Vergleich dieser Diagramme sind deutliche Unterschiede im Verlauf der Kurven erkennbar.

Die Kurven von *Single Landmark* steigen und fallen eher langsam. Dies weißt darauf hin, dass die Werte für Position und Validität in diesem Verfahren stark geglättet werden, so dass keine größeren Sprünge auftreten. Insgesamt bewegen sich die Validitäten, die das *Single Landmark*-Verfahren zurückgibt, auf einem niedrigeren Niveau als bei den anderen drei Lokatoren. Dies liegt daran, dass hier die Validitäten hauptsächlich aus der Güte der Eingangsdaten und den im jeweiligen Schritt durchgeführten Korrekturen berechnet wird. Bei den MonteCarlo-Ansätzen wird die Validitäten hingegen aus der Verteilung der Partikel bestimmt. Es liegen also grundsätzlich verschiedene Verfahren zur Berechnung dieses Gütewertes vor.

Die Kurven der Positionsfehler von *GT 2003* und *GT 2004* ähneln sich sehr stark, so dass die Differenzen wohl hauptsächlich auf den entsprechenden Spielverlauf zurückzuführen sind. Bei beiden Verfahren treten öfters kurzzeitige starke Positionsfehler auf, die in kurze hohe Peaks in den Kurvenverläufen resultieren. Dies ist ein Hinweis

darauf, dass diese Verfahren starke Sprünge in den Messwerten weniger stark glätten und es dadurch öfters zu extremen Fehllokalisierungen kommt. Diese Fehllokalisierungen dauern dann aber auch nur für sehr kurze Zeit an und werden schnell wieder korrigiert. Die Validitäten liegen meist auf einem recht hohen Niveau, korrelieren aber auch sehr gut mit den Positionsfehlern. Das heißt, bei starken Positionsfehlern sinkt die Validität in den meisten Fällen auch entsprechend ab.

Das Diagramm des *Fusion 2003*-Lokators zeigt erwartungsgemäß Charakteristika von den beiden verschiedenen Ansätzen. Es treten weniger starke Fehlerpeaks auf, da diese wahrscheinlich durch die Glättung des *Single Landmark*-Verfahrens herausgefiltert werden. Die Validitätskurve hingegen ähnelt eher den Verläufen der *GT 200x*-Ansätzen. Sie ist meist auf hohem Niveau, fällt aber bei Fehlern schnell ab. Dies ist allerdings nicht so ausgeprägt sichtbar, da keine extremen Positionsfehler in dieser Logdatei auftreten.

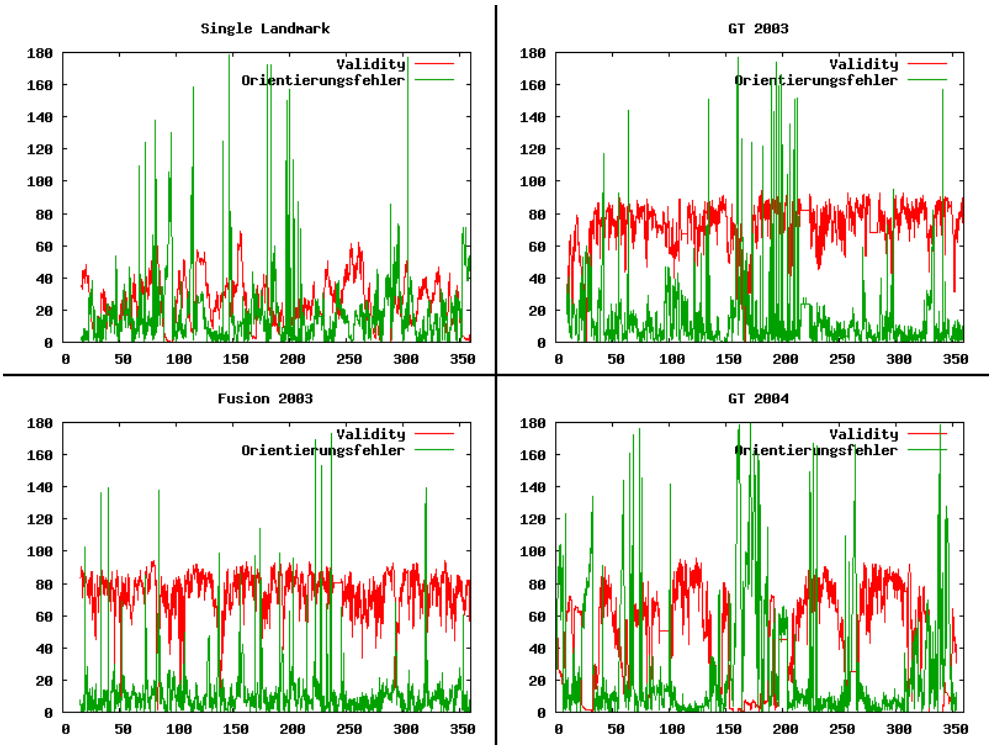

Abbildung 7.6: Diagramme der Logdateien mit den jeweils besten Orientierungsergebnissen der verschiedenen Selbstlokatoren

Abb. 7.6 zeigt ähnliche Diagramme für die Logdateien mit den jeweils besten Orientierungsergebnissen für die verschiedenen Verfahren. Die Angabe des absoluten Orientierungsfehlers erfolgt dabei in Grad, so dass sich dieser wieder in ähnlichen Größenordnungen bewegt wie die Angabe der Validitäten in Prozent. Da die Selbstlokatoren keine getrennten Validitätswerte für Position und Orientierung liefern, sind die Charakteristika dieser Kurven ähnlich der zuvor beschriebenen. Die Kurven der Orien-

tierungsfehler unterscheiden sich zwischen den Verfahren weniger stark als die zuvor beschriebenen Kurven der Positionsfehler. Es treten bei allen Verfahren sehr häufig hohe Peaks auf, die oft bis in die Nähe des maximalen Fehlers von 180° reichen. Außerhalb dieser Peaks erscheint die Fehlerkurve des *Single Landmark*-Verfahrens etwas welliger auf einem höheren Niveau zu verlaufen als bei den anderen drei Verfahren. Das mittlere Niveau außerhalb der Peaks liegt beim *Fusion 2003*-Ansatz am niedrigsten. Die Unterschiede sind hier aber sehr gering.

Der Fusion-Ansatz scheint also sowohl bei den Positions- als auch bei den Orientierungsfehlern eine gute Kombination zu sein. Auf der anderen Seite sind die Ergebnisse stark vom Spielverlauf des jeweiligen Roboters abhängig, so dass anhand der vorliegenden Daten kein endgültiges Ergebnis ermittelt werden kann.

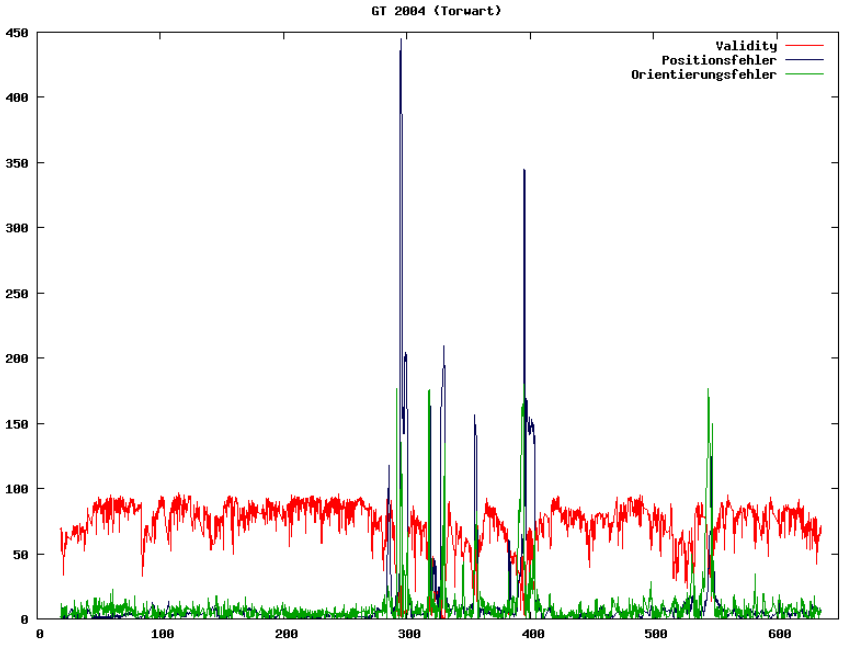

Abbildung 7.7: Diagramm einer Logdatei von einem Torwart

Alle obigen Kurven wurden aus den Daten von Feldspielern gewonnen. Wie stark die Ergebnisse von dem Spielverlauf eines Roboters abhängen, zeigt ein kombiniertes Diagramm aus Positions- und Orientierungsfehlern für einen Torwart (Abb. 7.7). Die meiste Zeit des Spiels positioniert sich der Torwart unbedrängt in der Mitte des Tores. Daraus resultiert, dass die Fehler fast über den gesamten Spielverlauf auf einem minimalen Niveau sind. Kommt allerdings ein Stürmer mit dem Ball vor das Tor, konzentriert er sich nur noch auf den Ball und wird in einen Kampf um diesen verwickelt. In dieser Situation bekommt er kaum noch Daten für die Selbstlokalisierung und die Fehler steigen rapide an. Teilweise erstreckt sich dann der Positionsfehler über mehr als die Spielfeldlänge, und auch die Orientierungsfehler reichen an ihren maximal möglichen Wert heran. Der Roboter ist in diesen Situationen also fast völlig orientierungslos.

Dies kann im schlechtesten Fall zu Eigentoren führen, falls der Torwart zum Schuss kommt. Diese Erkenntnisse lassen sich auch auf einen Feldspieler übertragen. Ist er in viele Zweikämpfe verwickelt, so wird das Gesamtergebnis der Analyse eher schlecht ausfallen. Positioniert er sich hingegen über weite Strecken des Spiels nur im freien Raum, so ergeben sich wesentlich bessere Ergebnisse. Mit wesentlich mehr Daten ließen sich evtl. signifikante Unterschiede zwischen den verschiedenen Verfahren feststellen. Es liegt allerdings nahe, dass Verbesserungen bei der Selbstlokalisierung mehr von einer intelligenten Blicksteuerung abhängt, die auch in kritischen Situationen noch genug Merkmale des Spielfeldes erfasst, als von den verschiedenen Verfahren. Außerdem werden Verfahren, die auf Landmarken beruhen, bald nicht mehr einsetzbar sein, da diese schon jetzt im Verhältnis zur Spielfeldgröße sehr dünn gesät sind und bald komplett entfernt werden.

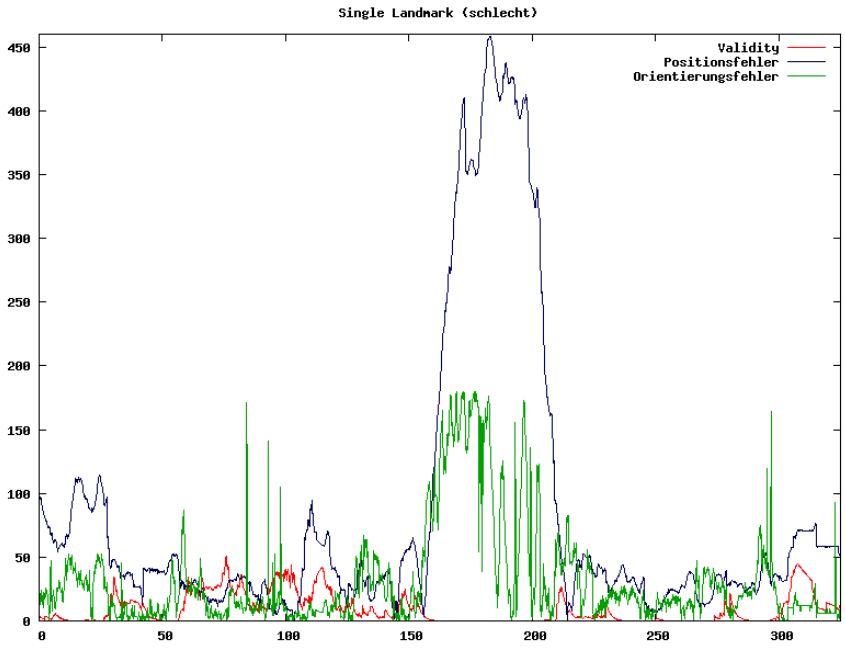

Abbildung 7.8: Diagramm von der schlechtesten Logdatei des Single Landmark Selbstlokators. Trotzdem ergeben sich gute gewichtete Mittelwerte

Die Ergebnisse der Analysen der 20 Logdateien geben auch Aufschluss über die Aussagekraft des mittleren gewichteten Fehlers. Bei der Berechnung dieses Wertes wurden die Fehler jeweils mit den entsprechenden Validitätswerten multipliziert. Die Logdatei, die von dem roten Spieler mit der Nummer 3 bei Verwendung von *Single Landmark* aufgezeichnet wurde, zeigt bei allen ungewichteten Werten die schlechtesten Ergebnisse. Die gewichteten mittleren Fehler von 35 mm bzw. 1,3° sind hingegen mit Abstand die geringsten Werte. Ein kombiniertes Diagramm aus Positions- und Orientierungsfehlern (Abb. 7.8) erklärt, wie es dazu kommt. Generell ist das Niveau der Validitäten in dieser Aufzeichnung sehr gering und beträgt im Mittel nur 9,1%.

In der Mitte des Spielverlaufs kommt es zu einer lang andauernden extremen Fehllokalisierung. Wahrscheinlich resultiert diese aus einem langen Zeitraum, in der keine Landmarke erkannt wurde. Dieser ausgedehnte Fehler führt zu den schlechten Ergebnissen. Da aber dieser Umstand erkannt wurde und die Validität in diesem Zeitraum auf nahezu null absinkt, wird dieser Spielabschnitt für den gewichteten Fehler wie eine fehlerfreie Lokalisierung gewertet. Dies führt zu den extrem niedrigen gewichteten Fehlerwerten. Niedrige Werte für die gewichteten Fehler sollten also nur dann positiv gewertet werden, wenn die Validität im Mittel auf einem hohen Niveau liegt.

Die Analysen der verschiedenen Verfahren haben gezeigt, dass sich die Qualität verschiedener Selbstlokatoren nur schwer in wenige Zahlenwerte fassen lassen. Um mit den verwendeten Verfahren zu einem aussagekräftigen Schluss zu kommen, sind auf jeden Fall wesentlich mehr Daten nötig, was zu einem erheblichen Aufwand bei der Erfassung führt. Um die verschieden Lokatoren schon mit geringerem Aufwand beurteilen zu können, müsste sicher gestellt werden, dass die Daten unter ähnlichen Bedingungen erhoben werden. Dies ist bei einem normalen Spielverlauf nicht möglich. Eine Alternative wäre, einen im Voraus festgelegten Spielverlauf zu verwenden. Es ist allerdings sehr schwierig, einen solchen für acht Roboter fest zu programmieren, so dass er immer wieder exakt gleich abläuft. Außerdem ist nicht sichergestellt, dass die Ergebnisse dieses Referenz-Spielverlaufs auf ein reales Spiel übertragbar sind. Die beste Beurteilungsmöglichkeit stellen also im Moment die Diagramme des Spielverlaufs dar. Bei Betrachtung dieser Diagramme lassen sich zumindest diverse Charakteristika der verschiedenen Ansätze feststellen. Anhand dieser Feststellungen lassen sich dann entweder Fehler beseitigen oder das beste Verfahren auswählen.

7.2 Analoge Anwendung für den Ball

Die Bestimmung der Ballposition erfolgt in der Architektur des *GermanTeams* in zwei Schritten. Zuerst wird mit Hilfe des Ballerkenners aus dem Kamerabild eine aktuelle Position des Balls relativ zum Roboter bestimmt, falls dieser sich im aktuellen Bildausschnitt befindet. Im zweiten Schritt, der Ballmodellierung, wird aus dieser aktuellen Position, den zuvor erkannten Positionen, von Teammitgliedern übermittelten Informationen und einem Modell des Fehlers in der Erkennung des Balls die aktuelle absolute Position des Balls auf dem Spielfeld und dessen Richtung und Geschwindigkeit berechnet. Dazu diente in dem zur Weltmeisterschaft 2004 gewählten Ansatz ein Kalmanfilter.

Eine Beurteilung des Fehlers der Ballerkennung im Bild kann also zum einen dazu eingesetzt werden, diese Ballerkennung zu beurteilen und zu optimieren. Auf der anderen Seite hilft die Kenntnis über den genauen Fehler des Ballerkenners der Ballmodellierung bei der Wahl der richtigen Parameter.

Das Aufzeichnen einer Logdatei erfolgt dazu auf die gleiche Weise, wie bei der Beurteilung der Selbstlokatoren, bzw. es können die gleichen Logdateien für beide Zwecke verwendet werden. Damit Fehler in der Selbstlokalisierung keinen Einfluss auf die

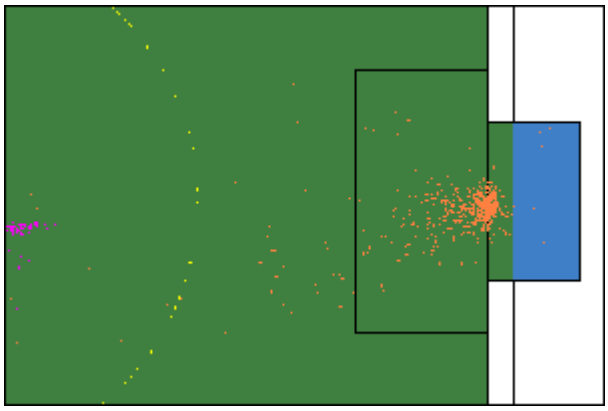

Abbildung 7.9: Visualisierung der Fehler der Ballerkennung

Beurteilung der Qualität des Ballerkenners haben, werden die Daten von Roboter und Deckenkamerasystem einheitlich in Positionen relativ zum betrachteten Roboter wieder in getrennten Listen abgelegt. Die Ballbewegungen lassen sich nur schwer in der Feldansicht visualisiert werden, da dort nur absolute Feldpositionen dargestellt werden können. Die einzelnen Fehler in den durchgeführten Vergleichen können aber auch hier als Punktwolke dargestellt werden. Um möglichst wenig die Darstellung der Positionsfehler zu stören, wurden hierfür orange Punkte gewählt und der Koordinatenursprung an die Torlinie verlegt (Abb. 7.9).

Die Berechnung passender Vergleichsdaten $\vec{b_i}$ aus dem Deckenkamerasystem zu den Ballpositionen, die vom Roboter geliefert werden, erfolgt analog zur Berechnung passender Roboterpositionen mittels linearer Interpolation der zeitlich angrenzender Daten $\vec{b_1}$ und $\vec{b_2}$:

$$\vec{b_i} = \frac{t_2 - t_i}{t_2 - t_1}\vec{b_1} + \frac{t_i - t_1}{t_2 - t_1}\vec{b_2}$$

Daraus wird ebenfalls analog zu den Positionsdaten der aktuelle Fehler $\vec{f_{B,i}}$ der relativen Ballpositionen zwischen Roboterdaten und Daten der Deckenkamera ermittelt. Die Einzelfehler werden auch hier zur späteren Verrechnung sowohl unverändert als auch betragsmäßig und quadriert aufsummiert. Aus diesen Daten und der Anzahl der Ballvergleiche m können dann analog zu 7.1.5 mittlerer Fehler $\bar{f_B}$ sowie Erwartungswert $E(\vec{f_B})$ und Standardabweichung σ_B des Fehlervektors berechnet werden:

$$\bar{f_B} = \frac{1}{m}\sum_{i=1}^{m} |\vec{f_{B,i}}|$$

$$E(\vec{f_B}) = \frac{1}{m}\sum_{i=1}^{m} \vec{f_{B,i}}$$

$$E(|\vec{f_B}|^2) = \frac{1}{m}\sum_{i=1}^{m}|\vec{f_{B,i}}|^2$$

$$\sigma_B = \sqrt{E(|\vec{f_B}|^2) - E(\vec{f_B})^2}$$

Diese Werte werden ebenfalls in der Ausgabe der Logdateianalyse in der *Message console* aufgeführt (Abb. C.4, S. 105). Sie können nun als objektive Beurteilung des Ballerkenners, zu dessen Optimierung und zur Weiterverwendung bei der Ballmodellierung dienen.

Da es im *GermanTeam* zurzeit keine sich wesentlich unterscheidenden Verfahren zur Ballerkennung gibt, wurde auf einen Vergleich verschiedener Ansätze verzichtet.

Kapitel 8

Zusammenfassung und Ausblick

Im folgenden Kapitel soll eine kurze Zusammenfassung der Ergebnisse dieser Arbeit sowie ein Ausblick auf zukünftige Anwendungsfelder des Deckenkamerasystems gegeben werden.

8.1 Zusammenfassung

Die Ziele der Diplomarbeit wurden erreicht, d.h. das erstellte Deckenkamerasystem ist in der Lage die Positionen und Ausrichtungen der Roboter auf dem Spielfeld zu erfassen, sowie Position und Geschwindigkeit des Balles zu ermitteln. Im Einzelnen wurden dazu die Teilprobleme der Kameramodellierung, Bildverarbeitung, Objekterkennung und der Anbindung an die Architektur des *GermanTeam* gelöst.

Die Verzerrungen durch das Kameraobjektiv wurden durch die Modellierung der intrinsischen Kameraparameter kompensiert. Zur Ermittlung der Kameraeigenschaften wurde hierbei eine Matlab Toolbox von Jean-Yves Bouguet genutzt [2]. Die Perspektivischen Verzerrungen des Kamerabildes wurden durch Aufstellen der Transformationsmatrix von Kamera- ins Feldkoordinatensystem herausgerechnet. Nach diesen beiden Stufen der Vorverarbeitung des Bildes kann jedem Pixel eine Position im Feldkoordinatensystem zugeordnet werden. Die Ergebnisse der Objekterkenner, wie sie in den Kapiteln 5.3.2 und 5.4.3 erläutert wurden, zeigen allerdings, dass hinsichtlich der Kalibrierung des Systems, wie sie für diese Experimente durchgeführt wurde, noch ein Spielraum für weitere Optimierungen gegeben ist. Die Ergebnisse lassen noch kleinere systematische Fehler in der Positionsbestimmung vermuten, die wahrscheinlich auf Ungenauigkeiten bei dieser Kalibrierung zurückzuführen sind.

Zur Erkennung des Balles wurden die Algorithmen zur Ballerkennung aus dem vom *GermanTeam* entwickelten Programmcode für die Aibo-Roboter auf die Bedingungen des Deckenkamerasystems angepasst. Hierbei wurde im Gegenzug auch der Ballerkenner auf dem Roboter nochmals verbessert. Die Ergebnisse, wie sie im Kapitel 5.3.2 dargelegt wurden zeigen, dass das System hinsichtlich der Erkennung des Balles im Bild sehr gut funktioniert. Die Standardabweichung liegt hierbei im Bereich

der durch die Auflösung des Bildes vorgegebenen Grenzen, d.h. der Mittelpunkt des Balles wird sehr stabil und nur etwa um einen einzigen Pixel schwankend erkannt.

Für die Detektion der Roboter auf dem Feld wurden diese mit Markern ausgestattet, deren Design in Anlehnung an die Marker einiger Teams der Smallsize-Liga erstellt wurde. Dabei erfolgt die Erkennung des Markers in zwei Schritten. Zum prinzipiellen Auffinden der Marker im Bild wird ein leicht modifizierter Ballerkenner eingesetzt, der einen pinken Kreis auf dem Marker sucht. Im zweiten Schritt wird über einen rotationsresistenen Binärcode, der in Sektoren um diesen pinken Kreis angeordnet ist, der Roboter identifiziert und seine Ausrichtung bestimmt. Die Ergebnisse der Experimente wurden im Kapitel 5.4.3 präsentiert und zeigen, dass auch bei der Erkennung der Marker und damit der Roboter im Bild eine sehr hohe Genauigkeit erreicht wird. Wiederum fiel die Standardabweichung relativ gering aus und lässt damit den Schluss zu, dass die Erkennung im Bild stabil funktioniert.

Zusammenfassend lässt sich zur Objekterkennung also sagen, dass das Teilproblem der Detektion im Bild gut gelöst wurde, aber das Gesamtergebnis im Feldkoordinatensystem noch Fehler enthält. Diese fallen allerdings vergleichsweise gering aus und lassen dennoch eine Nutzung des Systems zu den angestrebten Zwecken zu. Zudem ist anzunehmen, dass durch eine sorgfältige Kalibrierung des Systems eine weitere Erhöhung der Genauigkeit möglich ist. Da das System zur Ermittlung von Referenzdaten im Vergleich zu den durch die Roboter gewonnenen Positionen und Ausrichtungen eingesetzt werden soll, und die Roboterdaten um ein vielfaches größere systematische Fehler enthalten sind die verbleibenden Fehler in den durch das Deckenkamerasystem ermittelten Referenzdaten akzeptabel.

Zur Anbindung an die Architektur des *GermanTeam* stellt das Deckenkamerasystem seine Daten per UDP beliebig vielen Clients mit *RobotControl* zur Verfügung. Das hierfür verwendete Protokoll wurde im Kapitel 6 vorgestellt. Die Anbindung der Clients arbeitete im Rahmen der Experimente ohne Zwischenfälle wobei auch der parallele Einsatz von vier client-Rechnern an einem Deckenkamera-Server stabil funktionierte.

Als *Proof-of-concept* wurde wie im Kapitel 7 beschrieben unter *RobotControl* eine exemplarische Auswertung der Ergebnisse von Selbstlokalisierung und Ballmodellierung implementiert. Durch die Erstellung synchronisierter Logdateien von Roboter- und Deckenkameradaten kann hiermit eine Aussage über die Qualität der Algorithmen im Robotercode getroffen werden, die die Position des Roboters bzw. des Balles modellieren. Bei den durchgeführten Experimenten wurde dabei auch gezeigt, dass das Deckenkamerasystem die nötigen Referenzdaten in normalen Spielsituationen liefern kann und so die angestrebte praxisnahe Bewertung der Ergebnisse erlaubt.

8.2 Ausblick auf zukünftige Anwendungsfälle

Das in dieser Arbeit entwickelte System hat das Potential, in vielen Teilbereichen der Entwicklung gewinnbringend eingesetzt zu werden. Einige der möglichen Anwen-

dungsfälle werden im Folgenden vorgestellt und die Bedeutung der Deckenkamera im konkreten Fall erläutert.

8.2.1 Bewertung von Modulen

Die Softwarearchitektur des *GermanTeam* kapselt die unterschiedlichen Teilprobleme, die zur Programmierung der Roboter gelöst werden müssen, in einzelne Module. Deren Schnittstellen zum restlichen Code sind festgelegt und somit ist es möglich, einfach zwischen verschiedenen Lösungsansätzen zu einem Modul umzuschalten. Im Rahmen der kooperativen Softwareentwicklung an den einzelnen Universitäten werden so immer wieder neue Lösungsansätze entworfen oder bestehende weiterentwickelt. Für die Entwickler ist es hierbei nicht immer einfach zu erkennen, ob der neue oder veränderte Ansatz im Vergleich zu bisherigen oder konkurrierenden Lösungen wirklich besser abschneidet, bzw. in welchen Situationen er dies tut und in welchen nicht. Eine objektive Bewertung der einzelnen Lösungen ist somit spätestens dann notwendig, wenn man sich für den Einsatz einer Lösung zum Wettkampf entscheiden muss. Sowohl innerhalb der einzelnen Teams vor den *German Open* als auch im gesamten *GermanTeam* im Vorfeld der Weltmeisterschaft stehen die Entwickler vor dem Problem, auf Grund von subjektiven Beobachtungen während Testspielen oder konstruierten Testsituationen eine Entscheidung zwischen mehreren Lösungsansätzen treffen zu müssen.

Das Deckenkamerasystem erlaubt es nicht nur in speziellen Testfällen, sondern während natürlicher Spielsituationen die Leistung vieler Module zu beurteilen. Von der Objekterkennung (Ball, Landmarken, Feldlinien...) über die Modellierung (Ball, Selbstlokalisierung, Freiraum um den Roboter, ...) bis zu Entscheidungen der Verhaltenssteuerung lassen sich die Daten des Roboters nun mit einer objektiven Sicht der Situation zeitlich korrelieren und auswerten.

Das globale Weltbild der Deckenkamera erlaubt es, die Erkennungsleistung der Roboter zu beurteilen, indem man prüft, ob sich die detektierten Objekte tatsächlich in derselben oder zumindest einer ähnlichen relativen Position zum Roboter befinden. Hierbei muss allerdings beachtet werden, dass natürlich auch das Deckenkamerasystem gewissen Fehlern unterliegt; so ist die erkannte Ausrichtung der Roboter natürlich nicht vollständig korrekt und somit setzen sich diese Fehler gerade bei relativ zum Roboter gemessenen Objekten fort. Einfacher gestaltet sich der Abgleich auf Ebene der Weltmodellierung. Der Roboter erstellt aus seinen relativen Daten ebenfalls ein globales Weltbild, dieses kann direkt mit dem Weltbild der Deckenkamera verglichen werden. Die Verhaltenssteuerung schlussendlich müsste in der Theorie in der Lage sein, mit dem quasi idealen Weltbild der Deckenkamera bestmögliche Ergebnisse zu erzielen. Denkbar wäre hier z.B. den Robotern dieses Weltbild per WLAN zu übermitteln und so das Verhalten losgelöst vom Rest der Software zu testen.

8.2.2 Parametertuning

Neben der reinen Bewertung der Leistungsfähigkeit von Modulen kann das globale Weltbild, das durch die Deckenkamera erfasst wird, auch zur Verbesserung bestehender Module eingesetzt werden. Die Arbeit vieler Algorithmen im Programmcode der Roboter lässt sich über Parameter beeinflussen, beispielsweise wie lange die Modellierung der eigenen Position im Selbstlokator auf einer Hypothese beharrt, bevor sie auf Grund abweichender Sensordaten zu einer neuen übergeht. Ein weiteres Beispiel wären die Kovarianzmatrizen im Kalman-Filter der Ballmodellierung. Diese Werte wurden bisher rein von Hand eingestellt und liegen daher höchstwahrscheinlich noch nicht im für das Spielgeschehen optimalen Bereich.

Über die Rückkopplung der Ergebnisse mit der globalen Weltsicht des Deckenkamerasystems lassen sich bessere Werte bestimmen oder gar durch automatische Optimierungsverfahren erlernen. Die Leistung von Modulen, die auf Parameter angewiesen sind, die sich nicht direkt aus der Natur der Sache ableiten lassen, kann somit weiter verbessert werden.

8.2.3 Laufoptimierung

Zur Steuerung der Laufbewegung des Roboters hat das *GermanTeam* das Modul *WalkingEngine* programmiert. Dieses interpretiert einen Parametersatz, der das Bewegungsmuster der einzelnen Beine beschreibt und generiert daraus die Gelenkwinkel zur Ansteuerung der Motoren des Roboters. Die automatische Optimierung dieser Parametersätze wurde bereits an einigen Standorten des *GermanTeams* in Angriff genommen. Hierbei wurde sowohl eine komplett simulationsbasierte Lösung, die den kompletten Roboter mit allen relevanten physikalischen Größen im Rechner nachbildet, als auch diverse in der realen Welt agierende Lern- und Optimierungsmethoden angewandt. Bei letzteren läuft ein Roboter zur Probe mit den einzelnen Kandidaten an Parametersätzen. Damit deren Geschwindigkeit ermittelt werden kann, muss Start und Zielposition des Testlaufs gemessen werden (Abb. 8.1). Diese Messungen werden entweder von Hand durchgeführt - was allerdings die Anzahl der erfassbaren Werte angesichts des Aufwandes sehr einschränkt - oder der Roboter erfasst seine Position autonom über die Selbstlokalisierung.

Letztere Lösung hat den Vorteil, dass weitere Lernzyklen auch vor Ort auf Wettkämpfen durchgeführt werden können. Dies ist angesichts der immer etwas abweichenden Bodenbeschaffenheit sehr wichtig, da eine Laufbewegung, die auf dem heimischen Teppich im Labor funktioniert, auf einem anderen Material am Wettkampfort womöglich nicht genug Grip erzielt und die Roboter durch diesen Schlupf nicht vorankommen. Umgekehrt kann der Teppich auf dem Wettkampffeld aber auch günstigere Bedingungen aufweisen und extremere Laufbewegungen mit höheren Geschwindigkeiten gestatten. Ein Team, das in der Lage ist, sich so an die vorherrschenden Bedingungen kurz vor den Spielen noch einmal anzupassen, kann einen weiteren Vorteil gegenüber den anderen Teams herausholen.

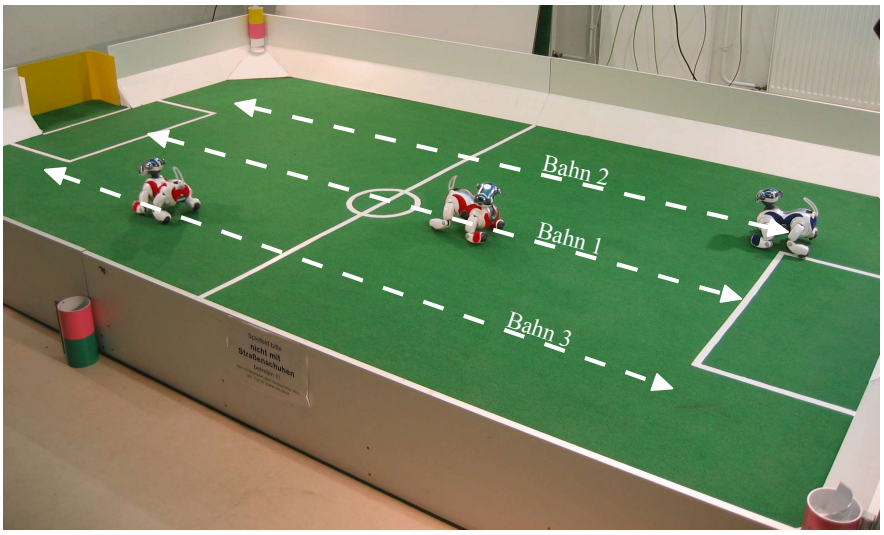

Abbildung 8.1: Roboter bei Testläufen zur Laufoptimierung

Das Lernen von Laufparametern auf Basis der robotereigenen Selbstlokalisierung hat allerdings auch einige entscheidende Nachteile. Während der Bewegung wackeln die Kamerabilder des Roboters und es kommt zu Messfehlern in den von den Sensoren ermittelten Gelenkwinkeln, welche zur Berechnung der Ausrichtung der Kamera relativ zum initialen Koordinatensystem auf dem Boden notwendig sind. Dadurch lokalisiert sich ein in Bewegung befindlicher Roboter weitaus ungenauer als ein stehender Roboter. Für das Lernverfahren kann daher meist nur Start und Zielpunkt berücksichtigt werden, wobei man davon ausgeht, dass der Roboter geradlinig dorthin gelaufen ist. Die Geschwindigkeit wird dann über die zurückgelegte Strecke ermittelt.

Vom Deckenkamerasystem hingegen wird der Roboter auch während der Bewegung erfasst und die Stabilität der Bewegung kann direkt beurteilt werden. Zum Beispiel, ob der Roboter geradeaus oder vielmehr einen Bogen läuft oder ob er dabei stark hin und her schwankt. Start und Zielposition erfasst die Deckenkamera mit höherer Genauigkeit als der Roboter. Dieser ist darauf angewiesen, dass er von der betreffenden Position gute Sicht auf genügend Landmarken, Feldlinien, Feldbegrenzungen etc. hat. Für die Deckenkamera ist hierbei lediglich entscheidend, dass der Roboter den Sichtbereich, also das Spielfeld nicht verlässt. Solange es zu keinen Kollisionen mit Feldbegrenzungen oder anderen Robotern kommt, können so auch mehrere Roboter gleichzeitig Testläufe durchführen, ob sie sich dabei gegenseitig die Sicht auf Landmarken und Feldlinien nehmen, spielt keine Rolle. Bei der Entwicklung im Labor können somit genauere und vor allem weitaus zahlreichere Daten für die Laufoptimierung erfasst werden. Zusätzlich bestünde die Möglichkeit, auch Seiteneffekte zu berücksichtigen; beispielsweise, ob eine bestimmte Laufbewegung die Positionierungsgenauigkeit des Selbstlokators beeinflusst, weil sie z.B. wackliger ausfällt als konkurrierende Parametersätze. Ein Nachteil ist allerdings sicherlich, dass das System nicht mit an den Wettkampfort genommen werden kann. Dort sind keine externen

Rechner und Deckenkameras erlaubt, eine letzte Optimierung vor Ort muss also weiterhin von Hand, bzw. automatisch durch den Roboter selbst bewertet werden.

8.2.4 Erkennung von Programmierfehlern

Zum Testen der Algorithmen während der Entwicklung hat das *GermanTeam* einen Simulator entwickelt, der es erlaubt, den Roboter-Code offline auf Basis von simulierten Bildern und Sensordaten auszuführen. Dieser Mechanismus erlaubt es, relativ schnell und unproblematisch Neuentwicklungen und Veränderungen kurz zu testen und prinzipielle Programmierfehler aufzudecken. Allerdings kann die simulierte Umgebung nie Tests in der realen Welt ersetzen. Ähnlich wie der Simulator den Vergleich der Ergebnisse des Roboterprogramms mit einem perfekten Weltbild erlaubt, kann das Deckenkamerasystem zu einem Vergleich in der Realität verhelfen. Durch das Aufzeichnen von Debugdaten vom Roboter und globalem Weltbild der Deckenkamera können beim anschließenden zeitlich synchronisierten Abspielen einzelne Ausreißer durch Fehlerkennungen oder Programmierfehler im Robotercode aufgedeckt werden. In Abb. 8.2 sieht man beispielsweise eine Abfolge von drei Weltbildern jeweils vom Roboter (dunklere Farben) und vom Deckenkamerasystem (hellere Farbe). Der Roboter meldete dabei beim Spiel an der Bande kurzzeitig für mehrere Sekunden, seine eigene Position sei im gegnerischen Strafraum. Eine auf diese Behauptung gestützte Verhaltensentscheidung des Roboters hätte dementsprechend grobe spieltechnische Fehler zur Folge und der zugrunde liegende Fehler in der Bildauswertung, bzw. in der Modellierung der Selbstlokalisierung sollte daher möglichst aufgespürt und beseitigt werden. Der entscheidende Vorteil der Erfassung realer Situationen auf dem Feld gegenüber den Tests im Simulator besteht hierbei darin, dass auch nicht vorhergesehene und daher nicht simulierte Problemfälle auftauchen können.

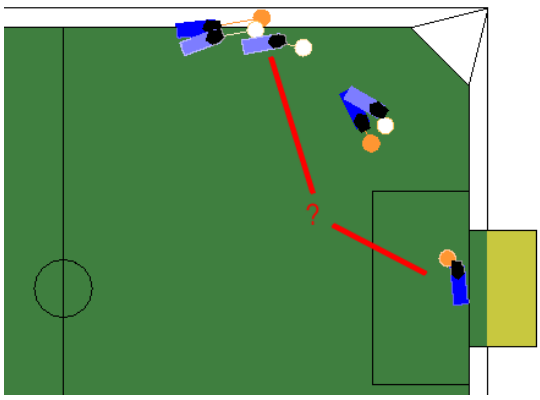

Abbildung 8.2: Kurzzeitiger grober Fehler in der Selbstlokalisierung

Die Geschwindigkeit der WLAN-Verbindung zum Roboter, bzw. die dafür von diesem aufzuwendende zusätzliche Rechenzeit, begrenzt allerdings die Menge der in Echtzeit übertragbaren Daten. Werden zu große Datenmengen angefordert, so werden

diese entweder nicht alle übertragen oder der Roboter reagiert insgesamt langsamer und kann dadurch einzelne Bilder der Kamera nicht mehr auswerten oder Bewegungen nicht länger in normaler Geschwindigkeit ausführen. Gegebenenfalls könnten hier Mechanismen entwickelt werden, um größere Datenmengen zuerst im Arbeitsspeicher des Roboters zu sammeln und dann stoßweise am Stück zu übertragen. So würden zumindest unverfälschte Daten für begrenzte Zeitabschnitte zur Verfügung stehen. Der Vorteil bei solchen umfangreicheren Daten bestünde darin, dass auch die Ausgangsdaten wie Bild und Sensordaten mit gesammelt werden könnten und anschließend beispielsweise ein Algorithmus zur Erkennung der Landmarke so lange angepasst und verfeinert werden könnte, bis er alle auftauchenden Problemfälle korrekt verarbeitet. Die Daten der Deckenkamera dienen hierbei als Referenz für das gewünschte Ergebnis.

8.2.5 Ausmessen der Odometrie

Führt der Roboter eine Bewegung aus, so schreibt er seine alte Position sowie entsprechend die relativen Positionen der Feldobjekte anhand der von ihm bei dieser Bewegung erwarteten Änderung seiner Position und Ausrichtung fort. Dieses Verfahren wird als Odometrie bezeichnet. Ändern sich die Laufparameter oder werden neue Bewegungsabläufe programmiert, so müssen im Vorfeld diese Daten erfasst werden. Momentan geschieht dies in teilweise mühsamer Handarbeit, d.h. der Roboter wird von Hand auf dem Feld positioniert, die Bewegung wird ausgeführt und anschließend werden Position und Ausrichtung des Roboters wiederum von Hand nachgemessen. Um Messfehler zu minimieren, sind hierbei mehrere Durchläufe notwendig. Über das globale Weltbild der Deckenkamera lassen sich die Odometriedaten bequem und ohne manuelles Eingreifen ermitteln. Der Roboter kann programmgesteuert die verschiedenen Bewegungsabläufe beliebig oft durchführen und über die ständige Erfassung durch die Deckenkamera lassen sich ausreichend viele Messdaten gewinnen, um durch Mittelung der Werte sehr genaue Ergebnisse zu erhalten.

Prinzipiell ist hierbei sogar denkbar, nicht nur einen einzigen Odometriewert pro Bewegungsablauf festzulegen, sondern in Zukunft womöglich sogar die komplette Wahrscheinlichkeitsverteilung nebst Streuung zu berücksichtigen. Dies würde der Verhaltenssteuerung auf dem Roboter ermöglichen, in bestimmten Situationen das Risiko eines Fehlschlags einer bestimmten Aktion zu berücksichtigen. Entwickelt man beispielsweise einen Schuss, bei dem eine gewisse Wahrscheinlichkeit besteht, dass er in eine andere als die anvisierte Richtung erfolgt, so könnte auch dieser Umstand bewertet werden und je nach Spielsituation könnte dann ggf. eine weniger riskante Aktion ausgewählt werden.

8.2.6 Taktikanalyse

Das größte Potential zur Verbesserung der Leistung der Roboter im Spiel sehen die Entwickler derzeit in der Verhaltenssteuerung. Bereits kleine Fortschritte, z.B. bei

der taktischen Aufstellung der Roboter auf dem Feld oder aktiveres Teamspiel bringen entscheidende Vorteile im Wettkampf. Durch die Aufzeichnung der Roboter und Ballpositionen kompletter Spiele über das Deckenkamerasystem, zusammen mit den lokalen Debugdaten der Roboter lassen sich taktische Schwächen des programmierten Verhaltens aufdecken. Die anschließende Analyse der Daten könnte es erlauben, das Verhalten zu optimieren und bei der Entwicklung noch nicht bedachte Situationen zu berücksichtigen. Taktische Positionierungsfehler, ungedeckte Freiräume und ungenutzte Chancen bei ebensolchen Schwächen des Gegners lassen sich im globalen Weltbild der Deckenkamera schneller erkennen. Der Abgleich mit den Daten der Roboter erlaubt es festzustellen, ob diese auf Grund der Leistungen der Objekterkennung und Weltmodellierung ebenfalls in der Lage waren diese Situation zu erfassen. Falls sie dort auch erkennbar gewesen wäre, kann man das Verhalten anpassen und in Zukunft in ähnlichen Fällen ein besseres taktisches Spiel der Roboter anstreben. Bisher konnte ein Spiel natürlich auch auf Basis von Videoaufzeichnungen analysiert werden, aber erst die exakte Erfassung des globalen Weltbildes und die Synchronisation mit den Daten der einzelnen Roboter erlaubt eine fundierte Analyse.

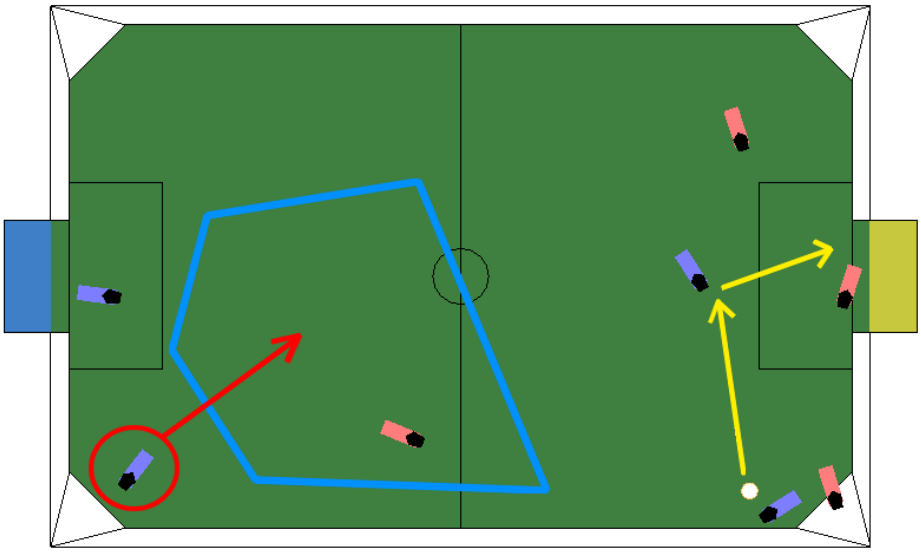

Abbildung 8.3: Taktikanalyse auf Basis von Deckenkameradaten

In Abb. 8.3 sieht man eine Beispielsituation aus einem mit der Deckenkamera erfassten Spiel. Der rot markierte Roboter links unten im Bild steht beispielsweise an einer taktisch sinnlosen Position. Man könnte ihn sinnvollerweise in seiner Rolle als Verteidiger zwischen Ball und eigenem Tor positionieren. Eine besser geeignetere Position könnte z.B. mit Blick auf den Ball im blau markierten Bereich eingenommen werden. Der Roboter am Ball rechts unten hätte die Chance durch einen gezielten Pass an seinen Teamkollegen vor dem gegnerischen Tor diesem eine Vorlage für einen

Schuss zu liefern (gelbe Markierung). Im weiteren Spielverlauf dieser konkreten Situation spielt der blaue Roboter im Ballbesitz allerdings selbst in Richtung Tor, das sehr erfolgreich vom roten Torhüter blockiert und verteidigt wird. Im Endeffekt landet der Ball wieder in der gleichen Ecke rechts unten im Bild, aus der er kurz vor der gezeigten Spielsituation erst befreit wurde. Wenig später geht er dann an die rote Mannschaft verloren, die zum Sturm auf das blaue Tor ansetzt. Positionierungsfehler wie der des blauen Verteidigers und das mangelnde Teamplay des Stürmers könnten also bei der Weiterentwicklung der Verhaltenssteuerung in Zukunft beseitigt und die Spielstärke damit erhöht werden. Zu beachten ist allerdings, dass diese Analyse auch unter Berücksichtigung des Weltbildes des betreffenden Roboters durchgeführt werden müsste, dessen Verhalten man kritisiert. Schließlich kann dieser seine Entscheidungen eben nicht auf das idealisierte globale Weltbild der Deckenkamera stützen, sondern ist auf die eigenen möglicherweise fehlerhaften und sehr wahrscheinlich unvollständigen Informationen angewiesen. Allerdings wird bei der Analyse auch ersichtlich, welche Daten die Roboter in Zukunft für ein besseres Spielverhalten möglichst erfassen sollten und sie kann somit neue Ansätze zur Entwicklung entsprechender Algorithmen zur Bildauswertung und Verbesserung der Weltmodellierung liefern.

8.2.7 Ausweitung auf größeres Feld

Das Spielfeld in der Sony-Vierbeiner-Liga soll für das kommende Entwicklungsjahr 2005 weiter vergrößert werden. Die Liga erhofft sich dadurch eine Entzerrung des Spielgeschehens und größere Vorteile bei der Entwicklung von taktischen Spielzügen, wie Passspiel und ähnlichem. Damit muss durch das Kamerasystem in Zukunft eine Fläche von insgesamt ca. 6 m auf 4 m abgedeckt werden. Derzeit kann mit der verwendeten Kamera und dem Weitwinkelobjektiv das komplette Feld in den Dimensionen der Regeln von 2004 abgedeckt werden, was einer Fläche von 5 m auf 2,8 m entspricht. Durch die Vergrößerung des Spielfeldes wird somit für 2005 auch eine Erweiterung des Kamerasystems notwendig.

Eine denkbare Lösung wäre hierbei, den Blickwinkel der Kamera zu verändern, so dass eine größere Bodenfläche im flacheren Winkel betrachtet wird, allerdings ginge dies deutlich zu Lasten der Genauigkeit der Erkennung. Zudem würde diese Perspektive sehr oft dazu führen, dass die Roboter den Marker auf ihrem Rücken gegenüber der Deckenkamera mit dem Kopf verdecken und so eine Erkennung erschweren, bzw. teilweise komplett verhindern würden. Selbiges gilt für den Einsatz noch weitwinkligerer Objektive. Es würde nicht nur die Verzerrung durch Perspektive bzw. Objektiv zunehmen, sondern zudem auch die Ortsauflösung deutlich geringer ausfallen, d.h. ein Pixel im Kamerabild würde einem größeren Bereich auf dem Spielfeld entsprechen, als dies im Moment der Fall ist. Daher erscheint einzig und allein der Einsatz weiterer Kameras sinnvoll. Beispielsweise eine Erweiterung des System um eine zweite Kamera, so dass anschließend je eine der beiden Kameras für eine der Feldhälften zuständig ist (Abb. 8.4). Wird ein ausreichend leistungsfähiger Rechner zur Auswertung der Bilder genutzt, könnten beide Kameras an einem Rechner angeschlossen werden,

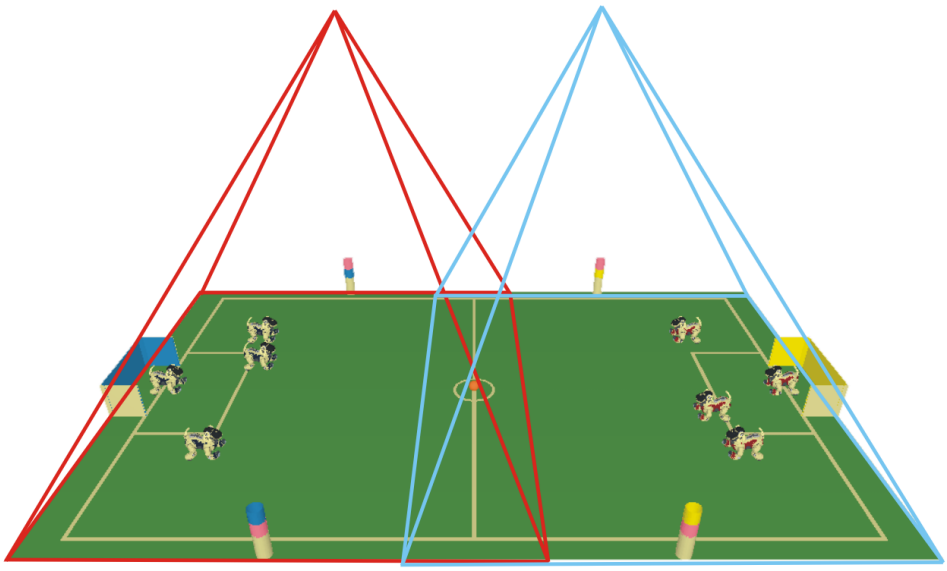

Abbildung 8.4: Dualkamerasystem zur Erfassung des größeren Spielfeldes

der anschließend die im Prinzip kaum zu verändernde Software für jede Kamera nutzt, und diese Teilergebnisse anschließend zu einem kompletten Weltbild fusioniert. Ist der gleichzeitige Anschluss beider Kameras an einem Rechner nicht möglich, könnten die Ergebnisse von zwei Kamerasystemen auch über das Netzwerk anhand der beiden erstellten Teil-Weltbilder zu einem kompletten globalen Weltbild kombiniert werden. In jedem der Fälle müssen Mehrdeutigkeiten wie das gleichzeitige Erkennen ein und desselben Roboters an unterschiedlichen Positionen mit verschiedenen Ausrichtungen oder das Erkennen eines zweiten Balles vermieden werden. Im doppelt abgedeckten Übergangsbereich wird es aufgrund von Rauschen zu leicht unterschiedlich wahrgenommenen Robotern bzw. Bällen kommen, hier könnte eine Mittelung der Werte sogar zu einer höheren Genauigkeit des fusionierten Weltbildes führen.

Anhang A

Kameramontage und Kalibrierung

Zur Nutzung der in dieser Arbeit vorgestellten Software sind einige Vorarbeiten notwendig. Die verwendete Kamera muss montiert und das System an die lokalen Gegebenheiten angepasst werden. Das folgende Kapitel beschreibt die nötigen Schritte, die zur Installation und Kalibrierung des Kamerasystems erfolgen müssen.

Abbildung A.1: Kamera mit Montagewinkel zur Befestigung an der Decke

A.1 Anbringen und Ausrichten der Kamera

Die Kamera sollte möglichst mittig über dem Feld mit annähernd geradliniger Ausrichtung nach unten angebracht werden, um die perspektivischen Verzerrungen in Grenzen zu halten. Dabei ist darauf zu achten, dass die höher auflösende Achse des Bildes möglichst auch der Längsachse des Feldes entspricht. Montagewinkel mit Stativaufsatz zur Befestigung der Kamera an der Decke (Abb. A.1) sind im Fotofachhandel erhältlich und verfügen meist auch über Stellschrauben zur genaueren Ausrichtung bzw. zum

Schwenken. Das Objektiv ist im Optimalfall so zu wählen, dass möglichst das komplette Feld inklusive der Torräume erfasst werden kann. Dabei sollte allerdings auch nicht zu viel überschüssige Umgebung rund um das Feld mit ins Bild gebracht werden, da so nur unnötig Auflösung und damit Genauigkeit verschenkt würde (Abb. A.2).

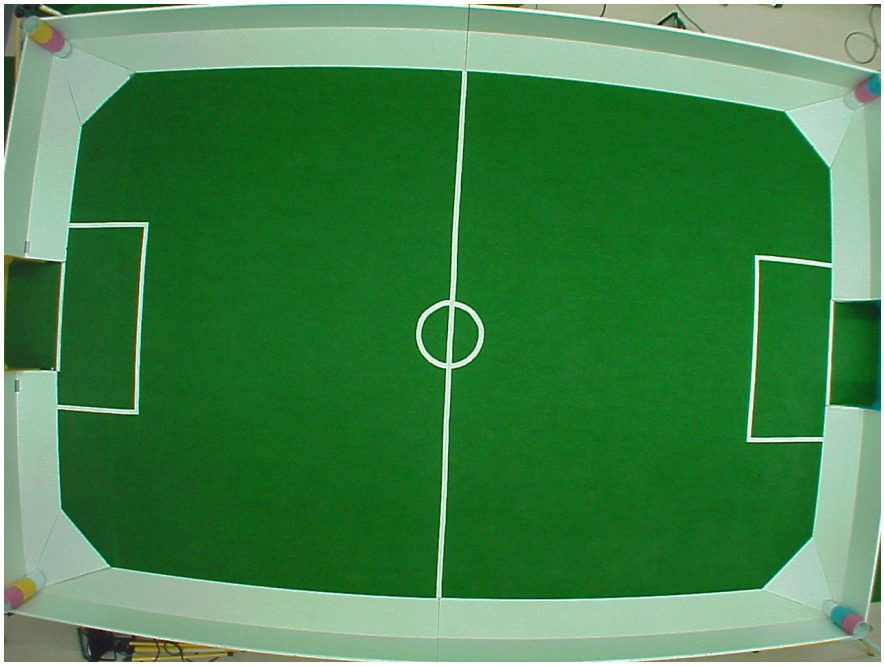

Abbildung A.2: Vollständiges Kamerabild

A.2 Bestimmung der intrinsischen Parameter

Aus praktischen Gründen empfiehlt es sich für den Kalibrierungsprozess der intrinsischen Parameter, die Kamera vorübergehend wieder abzumontieren. Bei der Verwendung der Camera Calibration Toolbox müssen diverse Bilder eines Schachbrettmusters aufgenommen werden. Wahlweise kann hierzu die Kamera bewegt werden oder das Schachbrett vor der Kamera. Letzteres gestaltet sich deutlich einfacher, wenn man dazu nicht auf einer Leiter unter der an der Decke montierten Kamera hantieren muss.

A.2.1 Aufzeichnen einer Bildfolge zur Kalibrierung

Bevor mit der Optimierung der Kalibrierungsparameter durch die Matlab Toolbox begonnen werden kann, müssen eine Reihe von Bildern eines planaren, gleichmäßigen, schwarzweiß Schachbrettmusters aufgezeichnet werden. Die Vorlage dafür sollte möglichst in einer ausreichenden Größe erstellt werden, damit auch in den äußeren Bereichen des Bildes später noch Messpunkte zur Optimierung beisteuern. Die Kantenlänge der Felder lässt sich in der Matlab Toolbox später konfigurieren, es muss also keine

spezielle Größe gewählt werden, sondern diese kann individuell an die verwendete Kamera, bzw. das Objektiv und den Abstand, in dem man die Bilder aufzeichnen möchte, angepasst gewählt werden. Das Muster kann mit einem beliebigen Grafikprogramm gezeichnet und ausgedruckt werden. Anschließend empfiehlt es sich, den Ausdruck auf einer festen und ebenen Trägerfläche zu arretieren, um die Handhabung während der folgenden Arbeitsschritte zu vereinfachen. Ist ein ausreichend großes Schachbrett erstellt und auf einer geeigneten ebenen Unterlage befestigt, kann mit der Aufzeichnung der Bilder begonnen werden. Debug-Einblendungen im Bild sollten hierfür möglichst vorher im Menü deaktiviert werden, da sie sonst auch in den gespeicherten Bildern auftauchen würden. Anschließend kann man damit beginnen, das Schachbrett der Kamera zu präsentieren und im File-Menü mit der Option *Save Bitmap* (Abb. A.3) jeweils das aktuelle Kamerabild speichern. Die Dateinamen sollten aus einem kurzen Text so-

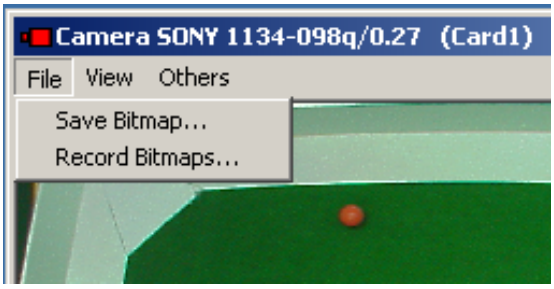

Abbildung A.3: File-Menü zur Speicherung von Kamerabildern

wie einer fortlaufenden Zahl bestehen, z.B. beginnend mit calib01.bmp. Für jedes neue Bild wird das Schachbrett in einem anderen Winkel bzw. einer anderen Entfernung gehalten, bzw. die Kamera bezüglich eines stationären Schachbretts bewegt. Es empfiehlt sich, ca. 20 bis 30 Bilder aufzuzeichnen und dabei darauf zu achten, dass in einigen der Aufnahmen das Schachbrettmuster auch die äußeren Bildbereiche erreicht, in denen die intrinsischen Verzerrungen am stärksten wirken. Ist so die Aufzeichnung einer Bildserie (Abb. A.4) abgeschlossen, kann mit der Auswertung der Bilder in Matlab begonnen werden.

A.2.2 Installation und Verwendung der Matlab Toolbox

Unter [2] steht die *Camera Calibration Toolbox for Matlab* zum Download bereit. An einem Rechner, an dem bereits Matlab in den Versionen 5.x bis 6.x installiert ist, entpackt man die Matlab Dateien (**.m*) aus dem Toolbox-Archiv in einen Unterordner (z.B. *TOOLBOX_calib*). Diesen Ordner fügt man anschließend in Matlab zum Pfad hinzu, damit die Routinen der Toolbox unter Matlab global zur Verfügung stehen. Unter Windows benutzt man hierfür einfach die Menüfunktion zum Editieren des Pfades, unter Linux und Unix die Kommandos *path*, bzw. *addpath*. Jetzt kann unter Matlab mit dem Kommando *calib_gui* die graphische Oberfläche der Toolbox initialisiert werden. Nach Auswahl des bevorzugten Speichermodells (Abb. A.5) erscheint das Hauptmenü

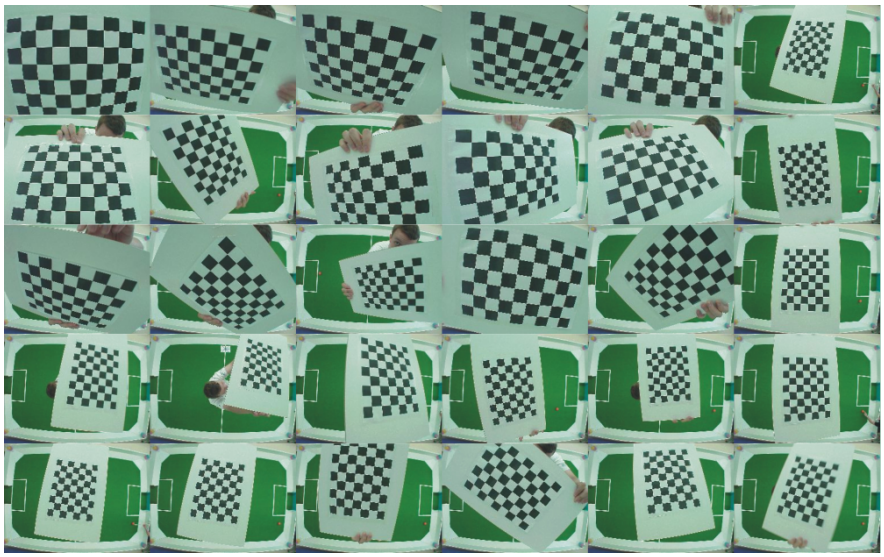

Abbildung A.4: Schachbrettmuster zur Kalibrierung der intrinsischen Parameter

Abbildung A.5: Calibration Toolbox - Auswahl für Speichermodell [2]

der Calibration Toolbox (Abb. A.6). Auf der Matlab-Kommandozeile wechselt man nun in das Verzeichnis, in dem die aufgezeichneten Bilder abgelegt wurden. Anschließend wird der Button *Image names* im Hauptmenü der Matlab Toolbox betätigt und nach Eingabe des Namenspräfixes der Bilder (z.B. calib) und des Formates (z.B. b für Bitmap) werden die Bilder geladen. Falls während dieser Operation eine Speicherfehlermeldung auftaucht, sollte man statt des schnellen Standard-Speichermodells auf die speichereffiziente langsamere Version ausweichen.

Den nächsten Schritt stellt die Detektion der Ecken des Teilbereiches der Schachbrettbilder dar, der zur Berechnung und Optimierung der Parameter herangezogen werden soll. Mit dem Button *Extract grid corners* wird der Prozess eingeleitet. Matlab fragt nun einige Parameter ab, durch Drücken der Enter-Taste wird jeweils der voreingstellte Wert für den betreffenden Parameter übernommen. Im Normalfall sollten die Standardparameter hier zu einem vernünftigen Ergebnis führen. Bei Problemen sei auf die genauere Beschreibung in der Dokumentation zur Matlab Toolbox unter [2] verwiesen. Anschließend präsentiert Matlab das erste Bild und bittet darum, die vier Ecken des auszuwertenden Bereichs anzugeben. Dabei handelt es sich *nicht* um die äußeren Ecken des Schachbretts, sondern um das größtmögliche innerhalb des Schach-

Abbildung A.6: Calibration Toolbox - Hauptmenü [2]

bretts liegende Rechteck, d.h., wenn das komplette Schachbrett im Bild ist, die nach innen gerichteten Eckpunkte der vier äußersten Felder (Abb. A.7).

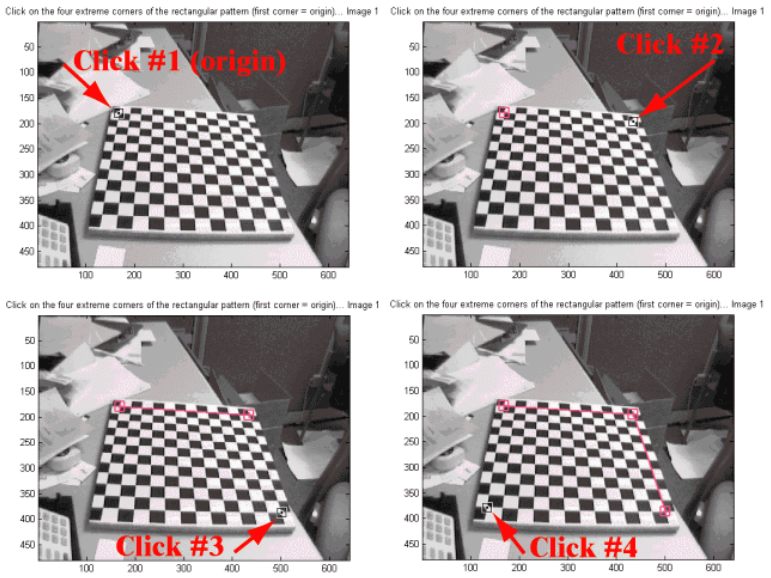

Abbildung A.7: Calibration Toolbox - Anklicken der Ecken [2]

Matlab fragt nun die Größe der Schachbrettfelder des verwendeten Musters in Millimetern ab und versucht selbständig zu bestimmen, wie viele Felder in vertikaler (x) und horizontaler (y) Richtung innerhalb des soeben definierten Bereiches liegen. Schlägt die automatische Zählung der Felder fehl, so müssen die Werte von Hand angegeben werden. Jetzt werden mittels roter Markierungen im Bild die geschätzten Positionen der Gitterpunkte angezeigt (Abb. A.8). Diese sollten bereits sehr nahe an den realen Positionen der Eckpunkte der einzelnen Schachbrettfelder liegen. Ist dies der Fall, betätigt man auf die Nachfrage des Programms, ob ein initialer Wert für den Verzerrungsfaktor geraten werden muss, die Enter-Taste und fährt nach der folgenden automatischen Extraktion der Eckpunkte mit dem nächsten Bild fort. Liegen die roten Markierungen allerdings nicht in der Nähe der realen Gitterpunkte, muss man einen ungefähren Wert für die Verzerrung von Hand einstellen, denn die automatische Extraktion nutzt nur einen kleinen Bildausschnitt als Fenster, innerhalb dessen sie nach dem Gitterpunkt sucht. Gibt man auf die entsprechende Nachfrage also irgendein beliebiges Zeichen ein, statt Enter zu drücken, kann man den Faktor für die

radiale Verzerrung von Hand eingeben. Dabei wird nach jeder Neuangabe das Bild mit den roten Markierungen aktualisiert und man kann so lange neue Werte ausprobieren, bis die Markierungen möglichst nahe an den realen Gitterpunkten liegen. Hierbei ist es sinnvoll, zuerst sehr kleine Werte zu versuchen, z.B. größenordnungsmäßig -0.2, und in kleinen Schritten von Zehnteln oder Hundertsteln zu tunen. Ist ein zufriedenstellendes Ergebnis erreicht, bestätigt man die entsprechende Nachfrage und fährt wiederum mit dem nächsten Bild fort. Die beschriebenen Schritte werden für sämtliche Bilder der Serie durchlaufen. Dabei macht es nichts, wenn nicht auf jedem Bild das komplette Schachbrett abgedeckt werden kann. Auch Teilausschnitte werden zur Auswertung herangezogen, ggf. muss dem Programm nur wie beschrieben mitgeteilt werden, wie groß der entsprechende Ausschnitt in Feldern ist, falls es dies nicht automatisch zählen konnte.

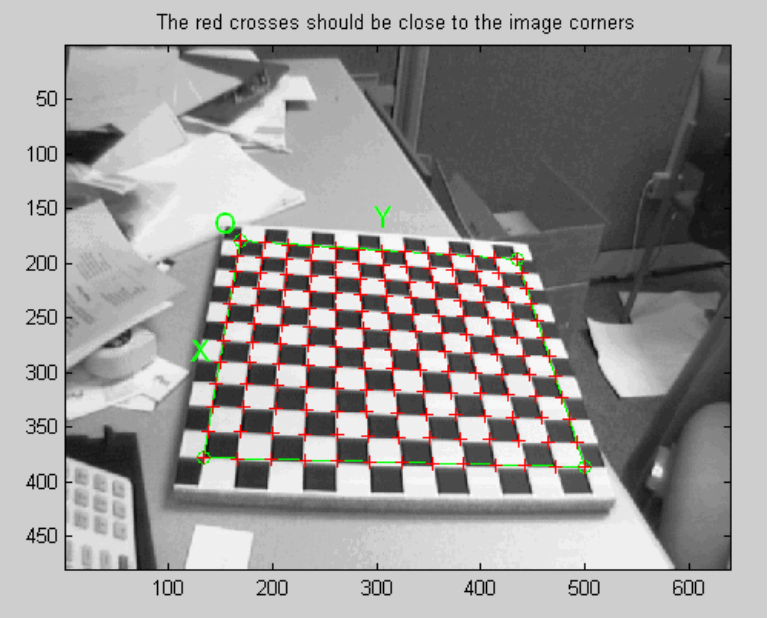

Abbildung A.8: Calibration Toolbox - Geschätzte Position der Gitterpunkte [2]

Nachdem die Gitterpunkte in allen Bildern der Serie detektiert wurden, kann mit der Berechnung und Optimierung der Kameraparameter begonnen werden. Hierzu betätigt man den Button *Calibration* im Hauptmenü der Camera Calibration Toolbox. Die erste Kalibrierung, deren Ergebnis nun dargestellt wird führt hier üblicherweise noch zu recht hohen Fehlern, ersichtlich an den Werten im Vektor *Pixel error*. Nachdem man sich durch die verschiedenen Visualisierungen des Ergebnisses geklickt hat, sollte man daher zur Verbesserung mindestens eine automatische Korrektur der Gitterpunkte durchführen, dies geschieht durch Betätigen des Buttons *Recomp. corners*. Alle Nachfragen nach Parametern und Methoden bestätigt man dabei am besten wieder mit Return und führt somit eine automatische Korrektur auf allen Bildern mit den soeben berechneten Kalibrierungsdaten als Ausgangspunkt durch. Nach Abschluss der

Operation klickt man wieder auf *Calibration*, um auf den korrigierten Gitterdaten neu zu kalibrieren. Der Pixelfehler sollte nun bereits deutlich geringer ausfallen als beim ersten Kalibrierungsdurchlauf. An diesem Punkt empfiehlt es sich, das Ergebnis der Kalibrierung durch Betätigen des Buttons *Save* in einer Matlab-Datei zu speichern. Zeigt die Visualisierung des Ergebnisses durch die Rückprojektion der Punkte auf das Schachbrett in den verschiedenen Bildern noch immer starke Abweichungen, sind weitere Korrekturen an der Extraktion der Gitterpunkte notwendig. Die Camera Calibration Toolbox verfügt zu diesem Zweck über diverse Möglichkeiten, den Fehler über die einzelnen Bilder zu visualisieren und so gezielt die Extraktion der Gitterpunkte in den am stärksten abweichenden Bildern zu korrigieren. Eine ausführliche Beschreibung aller Möglichkeiten würde den Rahmen dieser Arbeit sprengen und daher sei für solche Problemfälle auf die Webseite des Autors der Toolbox unter [2] verwiesen.

A.2.3 Übernahme der Ergebnisse

In der gespeicherten Matlab-Datei mit den Kalibrierungsergebnissen stehen nun alle benötigten Werte zum Ausgleichen der intrinsischen Verzerrungen des Systems aus Kamera und Objektiv. Im Menü *Others* der Deckenkamerasoftware (Abb. A.9) ruft man über den Menüpunkt *Camera Setup* den Dialog (Abb. A.10) zur Konfiguration der Kameraparameter auf. Tabelle A.1 zeigt die Korrespondenz der Werte zwischen Matlab-Datei und Camera-Setup Dialog.

Matlab-Datei	Camera-Setup (intrinsic)
fc [a ;	focallengthX
b]	focallengthY
cc [c ;	opticalCenterX
d]	opticalCenterY
alpha_c	ungenutzt
kc [e ;	2nd order radial distortion
f ;	4th order radial distortion
g ;	ungenutzt
h]	ungenutzt

Tabelle A.1: Zuordnung der intrinsischen Parameter

Die extrinsischen Werte im unteren Bereich des Dialogs werden erst wie im folgenden Abschnitt beschrieben ermittelt. Nachdem die intrinsischen Werte nun entsprechend den Angaben der Tabelle übernommen wurden, verlässt man den Dialog durch Bestätigen mit dem OK-Button. Die Daten werden daraufhin in der Registry hinterlegt und automatisch beim nächsten Programmstart auch wieder geladen. Wird das Programm mit verschiedenen Useraccounts verwendet, so ist es ggf. nötig, die Kalibrierungsdaten beim ersten Aufruf unter einem anderen Benutzernamen zu kontrollieren und ggf. dort zu konfigurieren. Im Grundzustand initialisiert das Programm

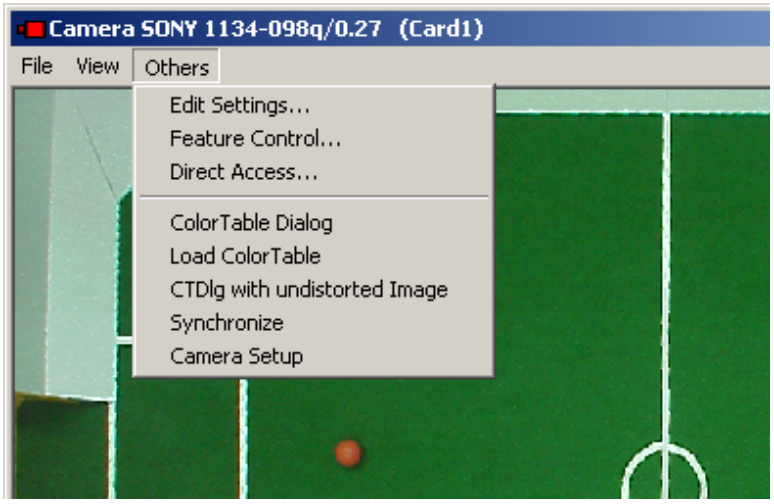

Abbildung A.9: Others-Menü

mit den Werten, die im Rahmen dieser Diplomarbeit für das Kamerasystem im Labor der *Darmstadt Dribbling Dackels* ermittelt wurden.

A.3 Bestimmung der extrinsischen Parameter

Um die extrinsischen Kameraparameter, d.h. die für die Kompensierung perspektivischer Verzerrungen des Kamerabildes, zu bestimmen, genügt es im Wesentlichen, die Kameramatrix aufzustellen. Als Kameramatrix wird hierbei die Transformationsmatrix bezeichnet, die die Umrechnung von Punkten im Kamerakoordinatensystem in Punkte im Feldkoordinatensystem erlaubt und umgekehrt. Das Feldkoordinatensystem ist hierbei als Rechtssystem definiert mit Ursprung am Feldmittelpunkt, x-Achse in Richtung des blauen Tores und z-Achse nach oben.

A.3.1 Ausmessen der Parameter

Der erste Schritt zum Ausmessen der Parameter besteht darin, den Fußpunkt der Kamera auszumessen, d.h. den Punkt auf dem Spielfeld, der direkt unter dem Objektiv der Kamera liegt. Nach der Montage der Kamera an der Decke des Labors kann dies z.B. durch Auspendeln des Fußpunktes mittels eines Senkbleis erreicht werden. Technisch versierteres Equipment wie Laserwasserwagen und Entfernungsmessgeräte würden natürlich ggf. zu genaueren Ergebnissen führen. Benötigt werden die Feldkoordinaten des Fußpunktes sowie die Höhe der Kamera über dem Feld, zusammen ergeben diese Werte den Translationsvektor für die Kameramatrix.

Darüber hinaus werden zum Aufstellen der Transformationsmatrix noch die Feldkoordinaten des optischen Zentrums, sowie die Drehung der Kamera bezüglich dem Feld benötigt. Letztere wird in Gestalt der Feldkoordinaten eines im Kamerabild vom

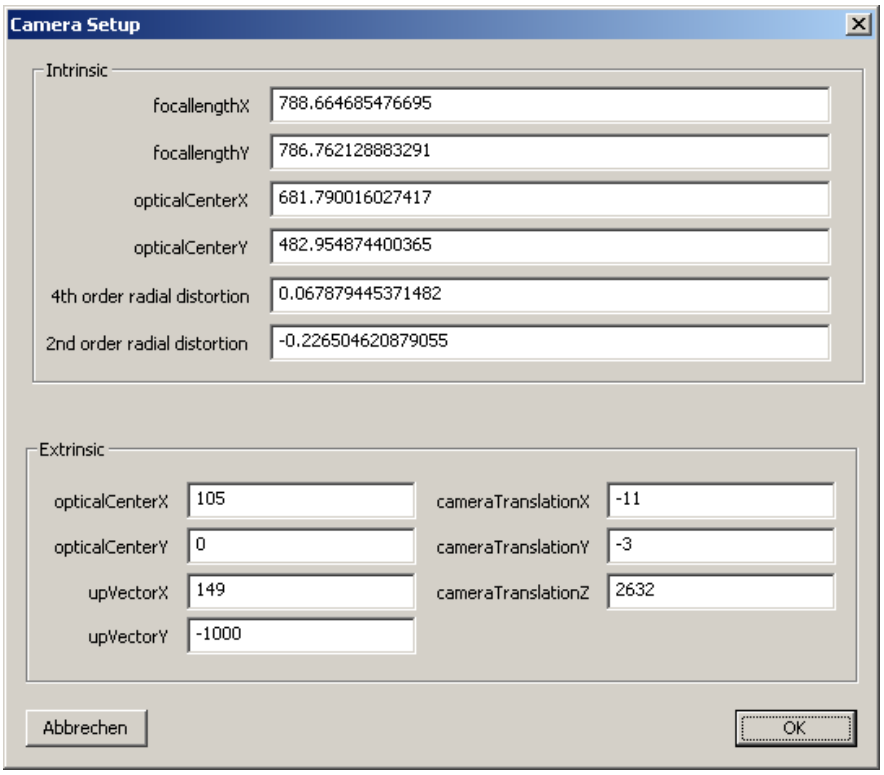

Abbildung A.10: Camera Setup Dialog

optischen Zentrum aus geradlinig nach unten (positive Richtung im Bildkoordinatensystem) führenden Vektors ermittelt. Im View-Menü der Deckenkamerasoftware (Abb. A.11) können mit dem Menüpunkt *Calibration Lines* Hilfslinien zur Kalibrierung in das Bild eingeblendet werden (Abb. A.12), die beim Ermitteln dieser Werte von Nutzen sind. Das blaue Kreuz kennzeichnet dabei das optische Zentrum der Kamera, das mit Hilfe der Camera Calibration Toolbox ermittelt wurde. Da es sich hier um einen von Objektiv und Kamera abhängigen Wert handelt, ist es notwendig, die intrinsischen Kameraparameter zu bestimmen und zu konfigurieren, bevor die in diesem Abschnitt beschriebene Ermittlung der extrinsischen Parameter durchgeführt werden kann. Der Punkt auf dem Feld, der dem blauen Kreuz entspricht, muss in Feldkoordinaten ausgemessen werden. Der Up-Vektor wird durch einen Punkt auf der roten Linie angegeben, dabei sollte dieser möglichst weit vom optischen Zentrum entfernt abgemessen werden, damit die Messfehler geringer ins Gewicht fallen. Beispielsweise kann dazu also 1m vom Mittelpunkt des Feldes in Y-Richtung entfernt der X-Wert des Schnittpunktes mit der roten Linie ermittelt werden. Diese Koordinaten ergeben den Up-Vektor.

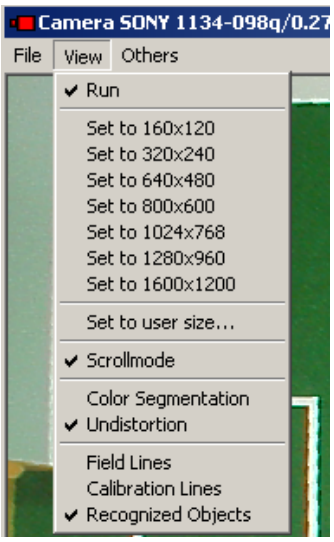

Abbildung A.11: View-Menü

A.3.2 Übernahme der Ergebnisse

Zur Übernahme der Messwerte ruft man wie schon im vorherigen Abschnitt bei der Übernahme der Werte für die intrinsischen Kameraparameter im Menü *Others* den Menüpunkt *Camera Setup* auf (Abb. A.9). Im Dialog zur Konfiguration der Kameraparameter (Abb. A.10) gibt man nun im unteren Bereich für die extrinsischen Kameraparameter die gemessenen Werte zur Berechnung der Transformationsmatrix ein. Die einzelnen Messergebnisse sind jeweils in mm anzugeben und korrespondieren mit den Dialogfelddaten wie in Tabelle A.2 angegeben. Nach Bestätigung der Eingaben mit dem OK-Button sind die Änderungen aktiv. Im View-Menü können zur Überprüfung der Kalibrierung mit der Option *Field Lines* die Feldlinien in das Bild projiziert werden. Diese sollten sich nun möglichst mit den realen Markierungen auf dem Feld decken. Zudem kann im View-Menü über den Menüpunkt *undistortion* auf die entzerrte Ansicht des Feldes umgeschaltet werden. Das Programm rechnet daraufhin die intrinsischen und extrinsischen Verzerrungen heraus und präsentiert ein vollständig entzerrtes Bild des Feldes.

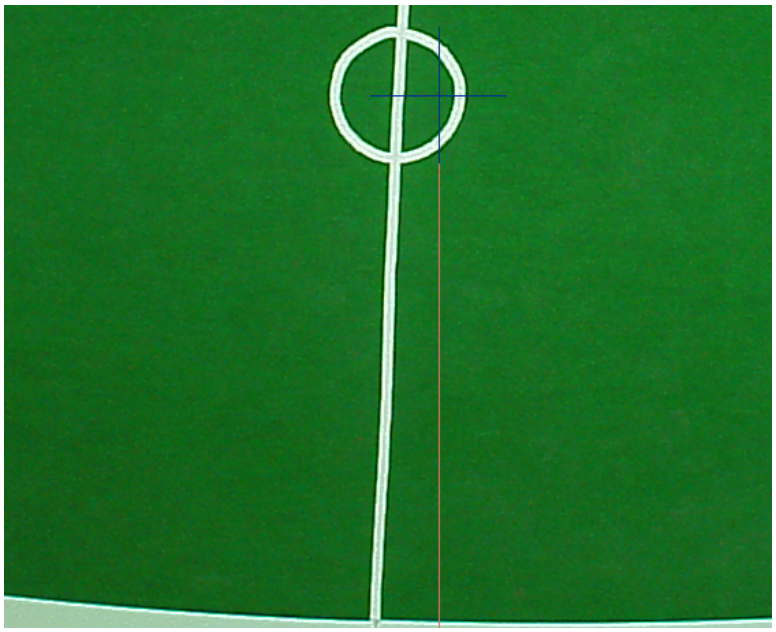

Abbildung A.12: Hilfslinien zur Kalibrierung

Messungen	Camera-Setup (extrinsic)
Ausgependelter Fußpunkt	
x	cameraTranslationX
y	cameraTranslationY
Kamera Höhe über dem Feld	
z	cameraTranslationZ
Feldposition des blauen Kreuzes	
x	opticalCenterX
y	opticalCenterY
Punkt auf der roten Linie	
x	UpVectorX
y	UpVectorY

Tabelle A.2: Zuordnung der extrinsischen Parameter

Anhang B

Serversoftware

Zur Erfassung der Objekte auf dem Spielfeld muss auf dem Rechner, an den die Deckenkamera via Firewire angeschlossen ist, die Serversoftware gestartet werden. Nach dem Start dieses Programms und wenigen Einstellungen benötigt man keinen weiteren Zugriff auf diesen Rechner, um mit dem Deckenkamerasystem arbeiten zu können. Die Anzeige des Kamerabildes kann aber zur Visualisierung der Arbeitsweise und zur Problembehandlung von Erkennungsfehlern dienen.

B.1 Das Kamerafenster

Nach dem Start des Programms erscheint zunächst ein kleines Hauptfenster, das zur Auswahl der Kameraschnittstelle und der richtigen Kamera dient. Darin sind die richtigen Parameter schon ausgewählt und es öffnet sich sofort ein weiteres, größeres Fenster, in dem nach einer kurzen Initialisierungsphase das Kamerabild zu sehen ist. In der Grundeinstellung passt sich das Kamerabild der Größe des Fensters an. Um die Orginalauflösung der Kamera zu erhalten und Stauchungen des Bildes zu verhindern, kann mithilfe des View-Menüs (Abb. B.1) in den sogenannten *Scrollmode* gewechselt werden. In diesem Modus sind evtl. außerhalb des Fensters liegende Bildteile über Scrollbalken zu erreichen. Zusätzlich kann über den Menüpunkt *Undistortion* im View-Menü eine Entzerrung des Bildes erreicht werden. Ist dieser Menüpunkt aktiviert, wird nur das Spielfeld ohne Umgebung angezeigt. Intrinsische und perspektivische Verzerrungen des Kamerabildes werden dabei eliminiert, so dass das Spielfeld als gerades Rechteck erscheint.

Zur Überprüfung der korrekten Arbeitsweise der intrinsischen und extrinsischen Korrekturen des Kamerabildes können außerdem Linien für die Feldbegrenzungen und verschiedene Spielfeldlinien eingezeichnet werden. Diese aktiviert man mit dem ebenfalls im View-Menü befindlichen Menüpunkt *Field Lines*. Decken sich diese Linien mit den Entsprechungen im Kamerabild, so arbeiten diese Algorithmen bestimmungsgemäß.

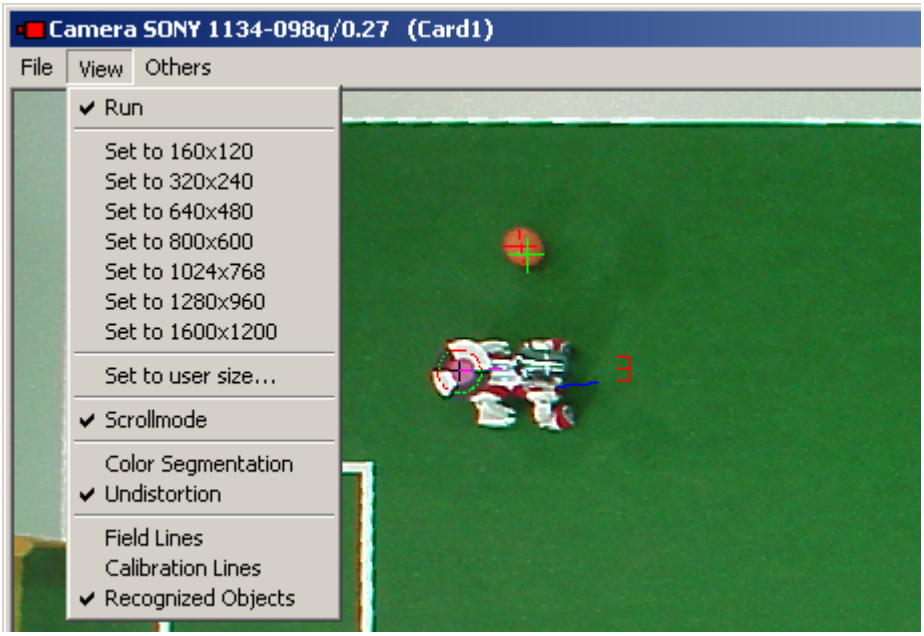

Abbildung B.1: Das Kamerafenster mit View-Menü und erkanntem Ball und Roboter

Weitere Linien, die zur Kalibrierung der Kameraparameter dienen, können über den Menüpunkt *Calibration Lines* eingeblendet werden. Eine ausführliche Beschreibung, wie eine solche Kalibrierung durchgeführt wird, liefert Anhang A.

Wurde eine zu den aktuellen Lichtverhältnissen passende Farbtabelle geladen (mehr dazu im nächsten Abschnitt) und der Punkt *Recognized Objects* im View-Menü aktiviert, werden die erkannten Objekte im Kamerabild gekennzeichnet (Abb. B.1). Orange Bälle werden durch rote und grüne Kreuze gekennzeichnet. Die roten Kreuze stellen dabei die Mittelpunkte der erkannten orangenen Kreisfläche dar, während die grünen Kreuze eine Projektion des Ballmittelpunktes auf die Spielfeldfläche repräsentieren. Wurde nur ein Ball auf dem Spielfeld erkannt, so wird dessen Position sowie aktuelle und Maximalgeschwindigkeit in der Statuszeile ausgegeben. Die Mittelpunkte der pinken Markerflächen der Roboter werden durch schwarze Kreuze angezeigt. Um diese schwarzen Kreuze ist der Kreis markiert, auf dem der Schwarzweiß-Code ausgelesen wurde. Außerdem wird die erkannte Roboternummer angegeben und eine Linie von dieser Nummer zum ermittelten Nullpunkt des Roboterkoordinatensystems eingezeichnet.

B.2 Erstellen einer Farbtabelle

Trotz der Versuche, die Objekterkennungsalgorithmen möglichst unabhängig von den aktuellen Lichtverhältnissen zu implementieren, wird zur Triggerung der Algorithmen zum Erkennen von Ball und Robotern auf eine Farbtabelle zurückgegriffen. Diese kann

sehr einfach durch Aufruf des ColorTable Dialogs (Abb. B.2) im Others-Menü erstellt werden. Dabei kann auch auf die Algorithmen zurückgegriffen werden, die innerhalb der Objekterkenner zur Berechnung der Ähnlichkeit zu einer bestimmten Farbe herangezogen werden.

Um eine Farbtabelle damit zu erstellen, betätigt man zuerst den *Capture*-Knopf um das aktuelle Kamerabild zu erfassen. Dieses wird dann im oberen linken Teil des Dialogfensters eingeblendet. Direkt darunter erscheint das Bild nach der aktuellen Farbtabelle klassifiziert. Um nun eine bestimmte dargestellte Farbe einer Farbklasse zuzuordnen, wählt man diese Farbklasse aus dem Dropdownfeld direkt unter dem *Capture*-Knopf aus und klickt danach im Kamerabild auf die entsprechende Farbe. Außerdem ist es möglich, auch im farbklassifizierten Bild einen Punkt anzuklicken. Es wird dann die Farbe an der entsprechenden Stelle im Originalbild in die ausgewählte Farbklasse aufgenommen. Um nicht jeden Farbwert einzeln einer Farbklasse zuordnen zu müssen, kann die Größe eines Bereichs um die ausgewählte Farbe im Farbraum angegeben werden. Alle Farbwerte innerhalb dieses Bereichs werden dann der aktuellen Farbklasse zugeordnet. Die Auswahl dieser Größe erfolgt direkt unterhalb des Auswahlfeldes der aktuellen Farbklasse neben der Bezeichnung *Range*.

Direkt darunter befinden sich vier Knöpfe zur Verwaltung von Farbtabellen. Diese bewirken im einzelnen:

Load: Hiermit kann eine bereits abgespeicherte Farbtabelle geladen werden.

Save: Mit diesem Knopf speichert man den aktuellen Stand der Farbtabelle.

New: Eine neue Farbtabelle wird angelegt. Eventuelle Änderungen an der aktuellen Farbtabelle werden verworfen.

Clear: Alle der ausgewählten Farbklasse zugeordneten Farbwerte werden aus der Farbtabelle gelöscht.

Als nächstes folgt ein Dropdown-Menü zur Auswahl der Zoomstufe des Kamerabildes. Damit lässt sich das Kamerabild stark vergrößert darstellen, um kleine Farbflächen besser auswählen zu können.

Der folgende Bereich dient zur Auswahl der Anzeige im unteren linken Bereich des Dialogfensters. Wurde hier *Segmented* ausgewählt, so erscheint in diesem Bereich ein gemäß der aktuellen Farbtabelle klassifiziertes Abbild des darüber angezeigten Kamerabildes. Wird stattdessen eine der Farbklassen darunter ausgewählt, bezieht sich die Anzeige nur noch auf diese Farbklasse. Zum einen werden die Farben, die schon dieser Farbklasse zugeordnet sind, in einer einheitlichen Farbe gekennzeichnet. Außerdem werden farbklassenspezifische Algorithmen angewandt, um die Ähnlichkeit zu dieser Farbklasse mithilfe eines Farbverlaufes anzuzeigen. Anhand eines Reglers kann nun ein Schwellwert eingestellt werden. Farbwerte oberhalb dieses Wertes werden im unteren linken Anzeigebereich sodann in einer anderen Farbe markiert. Um alle Farbwerte oberhalb dieses Schwellwertes in die aktuell ausgewählte Farbklasse aufzunehmen, betätigt man den über dem Regler befindlichen *Generate*-Knopf.

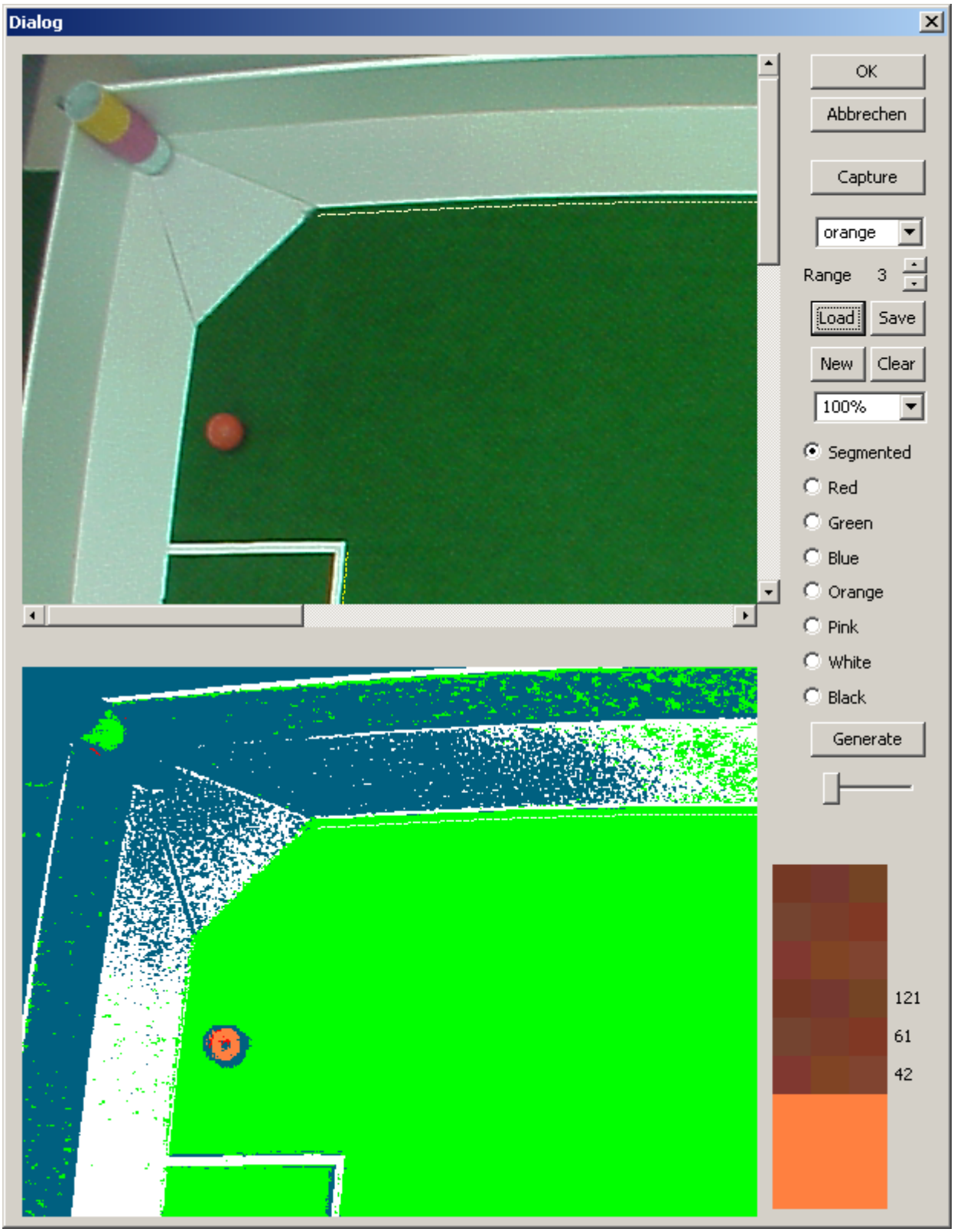

Abbildung B.2: Der ColorTable Dialog

Im unteren rechten Bereich des ColorTable Dialogs befinden sich drei übereinander liegende Felder, die jeweils aus neun Unterflächen bestehen. Die mittlere Fläche des mittleren Feldes ist in der Farbe gefüllt, die sich unter dem Mauzeiger befindet, falls sich dieser im Bereich des Kamerafeldes befindet. Falls diese Farbe in die Farbtabelle aufgenommen wird, werden zusätzlich alle Farben hinzugefügt, die sich im Farbraum in einem Würfel um diesen Farbwert befinden. Die Größe dieses Würfels wird wie oben beschrieben eingestellt. Die Farbwerte der Eckpunkte dieses Würfels werden in den acht Flächen angezeigt, die um die Fläche der aktuellen Farbe angeordnet sind. Im oberen Neuner-Feld werden die gleichen Informationen für den Punkt des letzten Klicks angezeigt. Im unteren Feld werden dagegen die Farbklassen angezeigt, die laut aktueller Farbtabelle den neun darüber liegenden Farbfeldern zugeordnet sind. Diese Anzeigen sollen dazu dienen, vor dem Klick zu erkennen, welche Farben genau in die aktuelle Farbklasse aufgenommen würden und welchen Farbklassen diese bisher zugeordnet sind. Zu Debugzwecken der Farbähnlichkeitsberechnungen werden zusätzlich neben dem mittleren dieser drei Felder die aktuellen RGB-Farbwerte angezeigt.

Um mit diesem Werkzeug möglichst schnell eine Farbtabelle aufzustellen, benutzt man am besten zuerst die Ähnlichkeitsfunktionen und befüllt die Farbtabelle mit dem Schwellwert-Schieberegler und dem Generate-Knopf. Zur Verfeinerung klickt man dann die fehlenden Farben für die einzelnen Farbklassen im Kamerabild an. Dies wiederholt man für einige weitere Bilder, die man durch erneutes Betätigen des Capture-Knopfes erhält.

B.3 Zeitsynchronisierung

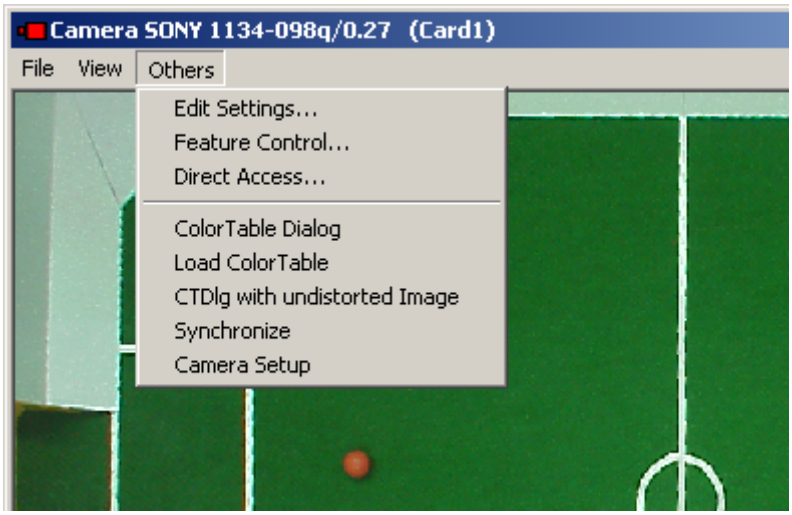

Abbildung B.3: Das Others-Menü zum Starten des ColorTable Dialogs und der Synchronisierung

Zur Erzeugung des Synchronisierungszeitstempel müssen sowohl Roboter als auch Serversoftware in den Synchronisierungsmodus versetzt werden. Dies kann auch für die Serversoftware von *RobotControl* aus geschehen. Man kann diesen Modus aber auch direkt in der Serversoftware über das Others-Menü (Abb. B.3) aktivieren. Während sich die Serversoftware im Synchronisierungsmodus befindet, werden in der Statusleiste anstatt der Balldaten Daten zur Synchronisierung ausgegeben. Im Einzelnen sind das die Helligkeit beim Start des Synchronisierungsmodus, die aktuelle Helligkeit, der Grad der Helligkeitsänderung und der Zeitstempel der letzten Synchronisierung. Soll der Synchronisierungsmodus der Serversoftware abgebrochen werden, so geht dies nur direkt im Kamerafenster, indem man den entsprechenden Punkt in Others-Menü deaktiviert. Der genaue Ablauf der Synchronisierung wird im Einzelnen in Anhang C beschrieben.

B.4 Überblick über die Implementierung

Die Software des Kamerarechners basiert auf dem Programm *FireDemo* der Firma Intek [6]. Der gesamte Rahmen und die Dialoge zu den Kameraeinstellungen wurden daraus übernommen. Das *View*-Menü wurde um einige Anzeigeoptionen ergänzt und im *Others*-Menü wurden Menüpunkte zur Steuerung der Bildauswertung hinzugefügt. Hauptsächlich wurden aber einige neue Klassen implementiert, die zur Bildauswertung und Übertragung der Ergebnisse dienen.

B.4.1 CameraView

CameraView ist Teil des aus dem Demoprogramm übernommenen Rahmens. Hier wurden die zusätzlichen Menüpunkte und Änderungen an der Anzeige des Kamerabildes eingefügt. Außerdem wird hier der *ImageProcessor* aufgerufen, in dem die eigentliche Bildbearbeitung stattfindet.

Die wichtigsten geänderten oder hinzugefügten Methoden von *CameraView* sind:

`OnDraw:` Hier werden bei Bedarf zusätzliche Linien eingezeichnet und das Bild farbsegmentiert oder entzerrt dargestellt.

`colorSegmentation:` Hier findet die eigentliche Farbsegmentierung statt.

B.4.2 RGBColorTable64

Die Klasse *RGBColorTable64* repräsentiert die verwendete Farbtabelle. Sie bildet Farben im RGB-Farbraum auf Fabklassen ab und fasst dabei jeweils 64 Farben zu einem Block zusammen, um Speicherplatz zu sparen.

Die wichtigsten Methoden von *RGBColorTable64* sind:

`addColorClass:` Eine bestimmten Farbe (und ggf. ein Bereich darum) wird einer Farbklasse zugeordnet.

`getColorClass:` Die Farbklasse zu einer Farbe wird ausgelesen.

`clearChannel:` Alle Zuordnungen zu einer bestimmten Farbklasse werden entfernt.

B.4.3 RGBImage

Das Kamerabild wird durch die Klasse *RGBImage* repräsentiert. Sie dient dazu, bestimmte Pixel auszulesen und in das Bild zu zeichnen.

Die wichtigsten Methoden von *RGBImage* sind:

`getPixel:` Der Farbwert eines Pixels wird ausgelesen.

`setPixel:` Der Farbwert eines Pixels wird gesetzt.

`DrawLine:` Zwischen zwei Punkten wird eine Linie gezeichnet.

`DrawCross:` An einen gegebenen Punkt wird ein Kreuz gegebener Größe gezeichnet.

B.4.4 RGBFieldImage

Das entzerrte Bild des Spielfeldes wird durch *RGBFieldImage* repräsentiert. Alle Koordinaten beziehen sich hier auf das Feldkoordinatensystem. In dieser Klasse befinden sich auch alle Algorithmen zur Entzerrung des Kamerabildes.

Die wichtigsten Methoden von *RGBFieldImage* sind:

`getPixel, setPixel, DrawLine, DrawCross:` Diese Methoden entsprechen denjenigen von *RGBImage*. Alle Angaben beziehen sich allerdings auf das Feldkoordinatensystem.

`pixelToFieldpoint:` Umrechnung von Bild- in Feldkoordinaten.

`fieldpointToPixel:` Umrechnung von Feld- in Bildkoordinaten.

`fieldpointToImageOffset:` Umrechnung von Feldkoordinaten in einen Offset im Kamerbild. Dies dient zur Beschleunigung der Anzeige des entzerrten Bildes.

`calculateFieldpointToPixelLookupTable:` Füllen von Lookup-Tabellen zur Beschleunigung der Anzeige des entzerrten Bildes.

`undistortIntrinsic:` Intrinsischer Teil der Umrechnung von Bild- in Feldkoordinaten.

`inverseIntrinsic:` Intrinsischer Teil der Umrechnung von Feld- in Bildkoordinaten.

`pixelToFieldExtrinsic:` Extrinsischer Teil der Umrechnung von Bild- in Feldkoordinaten.

`fieldToPixelExtrinsic:` Extrinsischer Teil der Umrechnung von Feld- in Bildkoordinaten.

`rotationCalibration:` Aufstellen der Rotationsmatrix aus den Kameraparametern.

`updateCameraParameters:` Auslesen der aktuellen Kameraparameter aus der Registry.

`checkLUT:` Testet die Lookup-Tabellen auf Aktualität und aktualisiert sie gegebenenfalls.

B.4.5 ColorTableDlg

Der ColorTable-Dialog, repräsentiert durch die Klasse *ColorTableDlg*, dient zum Aufstellen und Bearbeiten einer Farbtabelle. Seine genaue Bedienung wurde in Abschnitt B.2 beschrieben.

Die wichtigsten Methoden von *ColorTableDlg* sind:

`OnLButtonDown:` Bei einem Klick auf einen Punkt im Bild wird die entsprechende Farbe der eingestellten Farbklasse zugewiesen.

`colorSegmentation:` Die Farbsegmentierung wird bei entsprechender Auswahl für das untere Bild durchgeführt.

`getColorSimilarity:` Durch verschiedene Algorithmen wird für eine gegebene Farbe die Ähnlichkeit zu einer bestimmten prototypischen Farbe bestimmt.

`Show`**Farbklasse**`only:` Mithilfe der Algorithmen zur Farbähnlichkeit und der vorhandenen Einträge in der Farbtabelle wird im unteren Bereich ein auf die **Farbklasse** spezialisiertes Bild angezeigt. **Farbklasse** kann dabei `Red`, `Green`, `Blue`, `Black`, `White`, `Orange` oder `Pink` sein.

`OnBnClickedGenerate:` Aus den gegebenen Einstellungen wird mithilfe der Algorithmen zur Farbähnlichkeit die Farbtabelle für die ausgewählte Farbklasse gefüllt.

B.4.6 CameraSetupDlg

Der Camera-Setup-Dialog, repräsentiert durch die Klasse *CameraSetupDlg*, dient zum Eintragen der intrinsischen und extrinsischen Kameraparameter. Seine genaue Bedienung wird in den Abschnitten A.2.3 und A.3.2 beschrieben.

Die wichtigsten Methoden von *CameraSetupDlg* sind:

`OnInitDialog:` Auslesen der Kameraparameter aus der Registry.

`OnBnClickedOk:` Zurückschreiben der geänderten Kameraparameter in die Registry.

B.4.7 ImageProcessor

Die gesamte Bildverarbeitung wird durch den *ImageProcessor* gesteuert. Nach der Objekterkennung werden die Ergebnisse via UDP im Netzwerk bereitgestellt.

Die wichtigsten Methoden von *ImageProcessor* sind:

`execute:` Hier wird das Bild in einem groben Raster farbsegmentiert und bei den entsprechenden Farbklassen die Spezialisten für die Ballerkennung und zur Erkennung der pinken Robotermarker gestartet. Falls entsprechende Objekte gefunden wurden, werden die Lokatoren für Ball und Roboter gestartet, die Ballgeschwindigkeit, Roboterindentifikation und -ausrichtung ermitteln. Die Ergebnisse werden bei entsprechenden Einstellungen im Kamerabild eingezeichnet und im Netzwerk via UDP zur Verfügung gestellt.

`enableDebugDrawings:` Das Einzeichnen der erkannten Objekte wird aktiviert.

`disableDebugDrawings:` Das Einzeichnen der erkannten Objekte wird deaktiviert.

`synchronize:` Das Deckenkamerasystem führt die Zeitsynchronisierung durch.

`estimateBrightness:` Die durchschnittliche Helligkeit im Mittelkreis wird zur Zeitsynchronisierung abgeschätzt.

B.4.8 BallList

Die Klasse *BallList* repräsentiert die Liste der erkannten Bälle. Sie dient auch zur Speicherung der erkannten pinken Teile der Robotermarker.

Die wichtigsten Methoden von *BallList* sind:

`add:` Ein neuer Ball wird in die Liste eingefügt.

`isInside:` Es wird überprüft, ob ein gegebener Punkt innerhalb einer der bereits erkannten Bälle in der Liste liegt.

B.4.9 GT2004BallSpecialist, PinkBallSpecialist

GT2004BallSpecialist und *PinkBallSpecialist* sind die beiden Spezialisten zum Erkennen von Bällen und den runden pinken Markern auf den Robotern. Sie suchen ausgehend vom übergebenen Bildpunkt nach den Grenzen des Balls bzw. der runden pinken

Fläche und fügen Mittelpunkt und Radius in eine Liste ein, falls bestimmte Kriterien erfüllt werden. Beide Spezialisten arbeiten weitgehend auf die gleiche Weise. Sie unterscheiden sich fast nur in der Farbe und der Größe der zu erkennenden Objekte.

Die wichtigsten Methoden von *GT2004BallSpecialist* und *PinkBallSpecialist* sind:

`searchBall:` Start der Suche nach den Ballrandpunkten. Falls sich daraus ein Ball bilden lässt, wird der erkannte Ball der Liste hinzugefügt.

`scanForBallPoints:` Spinnennetzartiges Suchmuster nach den Randpunkten des Balls.

`findEndOfBall:` Sucht nach dem Rand des Balls entlang einer vorgegebenen Richtung.

`getSimilarityTo[Orange,Pink]:` Berechnung der Ähnlichkeit einer gegebenen Farbe zu dem Orange des Balls bzw. dem Pink der Robotermarker.

`createBallPerceptLevenbergMarquardt:` Aus den gefundenen Ballrandpunkten wird der Mittelpunkt und Radius bestimmt.

`checkIfPointsAreInsideBall:` Test, ob alle Ballrandpunkte zu gefundenem Mittelpunkt und Radius passen.

`is[Orange,Pink]FilledDisk:` Test, ob innerhalb des gefundenen Balls hauptsächlich Orange bzw. Pink vorkommt.

`addBallPercept:` Falls ein gefundener Ball/Marker verschiedenen Kriterien entspricht, wird dieser in die Liste eingefügt. Dabei werden bereits vorhandene Einträge in der Liste, die sich mit dem neuen Eintrag überschneiden, entfernt.

B.4.10 BallLocator

Der *BallLocator* berechnet für alle Bälle in der Ballliste die Feldkoordinaten. Wurde nur ein Ball erkannt, wird für diesen aus einer Liste der vorangegangenen Positionen dessen Richtung und Geschwindigkeit berechnet. Beides passiert in der Methode `calculateSpeeds`.

B.4.11 RobotLocator

Der *RobotLocator* ermittelt für jeden erkannten pinken Marker mithilfe des darum liegenden Schwarzweiß-Codes die Identität und Ausrichtung des dazugehörigen Roboters.

Die wichtigsten Methoden von *RobotLocator* sind:

`execute:` Für jeden erkannten pinken Marker wird der Schwarzweiß-Code ausgelesen.

`addSectors:` Bei jedem Schwarzweiß-Übergang wird die entsprechende Anzahl Sektoren dem Binärcode hinzugefügt.

`getRobotNumberAndOrientation:` Mithilfe des ausgelesenen Binärcodes wird die Identität und Ausrichtung des Roboters ermittelt.

Anhang C

Clientsoftware

Der Zugriff auf die Daten des Deckenkamerasystems wurde vollständig in das Entwicklungsprogramm *RobotControl* des *GermanTeams* integriert. Die Daten können von dort angefordert und dargestellt werden. Auch die Synchronisierung kann vollständig von *RobotControl* aus durchgeführt werden. Nachdem die Anwendung auf dem Server gestartet wurde, muss also zum weiteren Arbeiten nicht mehr auf diesen Rechner zugegriffen werden

C.1 Bedienung der RemoteCamToolbar

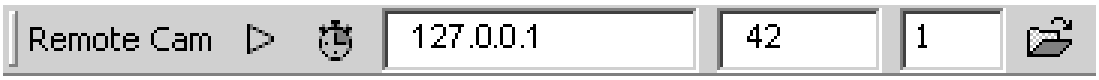

Abbildung C.1: Die RemoteCam-Toolbar

Als grafische Schnittstelle zur Steuerung des Deckenkamerasystems dient die RemoteCamToolbar (Abb. C.1). Sie besteht aus drei Eingabefeldern zur Konfiguration und drei Knöpfen zum Auslösen diverser Aktionen. Von links nach rechts haben diese folgende Bedeutung:

Start-Knopf: Mit diesem Knopf startet man die Anforderung von Daten vom Deckenkamerasystem. Ein weiterer Klick beendet die Datenübertragung wieder.

Synchronisations-Knopf: Mit diesem Knopf versetzt man das Deckenkamerasystem in den Synchronisations-Modus. Wird ein Synchronisations-Zeitstempel vom Deckenkamerasystem empfangen, wird dieser Knopf automatisch wieder zurückgesetzt. Ansonsten kann man den Synchronisationsknopf auch durch einen weiteren Klick zurücksetzen. Das Deckenkamerasystem verlässt dadurch allerdings nicht den Synchronisationsmodus.

IP-Eingabefeld: Hier trägt man die IP-Adresse des Rechners ein, an den die Deckenkamera angeschlossen ist und auf dem die Serversoftware läuft.

Port-Eingabefeld: Hier trägt man die Port-Nummer ein, auf der die Serversoftware Anforderungen für Deckenkameradaten erwartet. Der Port, auf dem die Serversoftware die Anforderung des Synchronisations-Modus erwartet, liegt um eine Nummer höher.

Spielernummer-Eingabefeld: Hier kann angegeben werden, aus der Sicht welcher Spielernummer die Daten von der Deckenkamera dargestellt werden sollen. Dabei wird aufsteigend zuerst das rote und dann das blaue Team durchnummeriert. Ist man mithilfe der WLANToolbar mit einem bestimmten Roboter verbunden, wird automatisch dieser Spieler auch von der RemoteCamToolbar ausgewählt. Eingaben in diesem Feld werden in diesem Fall nicht beachtet.

Analyse-Knopf: Nach Betätigung dieses Knopfes kann eine Log-Datei ausgewählt werden. Diese sollte entweder schon synchronisiert sein oder Synchronisationszeitstempel von Roboter und Deckenkamerasystem enthalten. Nach Auswahl einer solchen Datei wird eine Analyse der Selbstlokator- und Balldaten des Roboters durchgeführt und die Ergebnisse im Message-Viewer ausgegeben. Außerdem wird der Weg des Roboters aus eigener Sicht und aus Sicht der Deckenkamera in der Feldansicht visualisiert.

C.2 Erstellen einer synchronisierten Logdatei

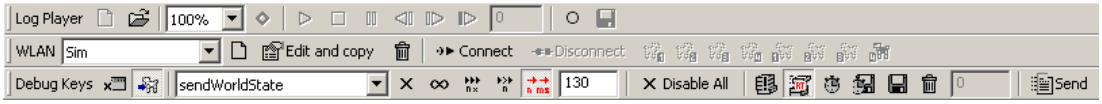

Abbildung C.2: Toolbars zur Aufzeichnung von Logdateien

Um eine Logdatei auszuwerten und die Daten, die der Roboter selbst erfasst hat, mit den Daten des Deckenkamerasystems zu vergleichen, muss diese Datei folgendes enthalten:

- Einen Zeitstempel des Sychronisationszeitpunktes vom Roboter
- Einen Zeitstempel des Sychronisationszeitpunktes vom Deckenkamerasystem
- Die Daten, die via WLAN vom Roboter übertragen wurden
- Die Daten, die vom Deckenkamerasystem übertragen wurden

Zusätzlich zur RemoteCam-Toolbar benötigt man zur Aufzeichnung einer solchen Logdatei weitere Toolbars von *RobotControl* (Abb. C.2). Im Einzelnen sind dies:

- Die *Logplayer-Toolbar* dient zum Starten, Unterbrechen und Beenden der Aufzeichnung. Außerdem kann man mit dieser Toolbar die Logdatei in verschiedenen Formaten speichern und vorhandene Logdateien öffnen und abspielen.
- Mit der *WLAN-Toolbar* verbindet man sich mit den Robotern und wählt einen Roboter aus, dessen Daten aktuell angezeigt werden sollen.
- Die *DebugKey-Toolbar* wird benötigt, um Daten vom Roboter via WLAN anzufordern.

Hinweise zur genauen Funktionsweise dieser Toolbars finden sich im Teamreport des *GermanTeams* [9].

C.2.1 Aufzeichnung der Synchronisationszeitstempel

Um die Synchronisationszeitstempel von Roboter und Deckenkamerasystem aufzuzeichnen, müssen zuerst beide Systeme in den Synchronisationsmodus versetzt werden. Zusätzlich platziert man eine ausgeschaltete Lampe in der Mitte des Spielfeldes. Der Roboter wird in den Synchronisationsmodus versetzt, indem man den Taster am Kinn des Hundes gedrückt hält und dabei den vorderen Taster auf dem Rücken betätigt. Dann richtet man den Roboter auf die Lampe aus und betätigt den mittleren Taster auf dem Rücken, um die aktuelle Helligkeit als Referenzwert zu speichern. Das Deckenkamerasystem schaltet man am einfachsten durch Betätigung des entsprechenden Knopfes in der RemoteCam-Toolbar in den Synchronisationsmodus.

Um die beiden Zeitstempel aufzuzeichnen, startet man nun die Aufnahme einer Logdatei mit der Logplayer-Toolbar. Nun ist alles für die Synchronisation vorbereitet und man kann die Lampe kurz einschalten. In der message console kann man dabei beobachten, ob die Synchronisationszeitstempel der beiden Systeme eintreffen. Danach kann die Aufzeichnung durch eine weitere Betätigung des Aufnahme-Knopfes in der Logplayer-Toolbar unterbrochen werden. Um den Roboter wieder in den normalen Spielmodus zu versetzen, muss der Taster auf dem Kopf etwas länger gedrückt werden.

C.2.2 Aufzeichnung der Daten

Bevor man die Daten vom Roboter und dem Deckenkamerasystem in einer Logdatei aufzeichnen kann, müssen diese von den beiden Systemen angefordert werden. Zur Anforderung der Daten von der Deckenkamera muss dazu nur der Play-Knopf in der RemoteCam-Toolbar betätigt werden. Danach treffen die Datensätze vom Deckenkamerasystem ungefähr mit der Framerate der Kamera (7,5 Hz) in RobobotControl ein. Dies entspricht einem Abstand von ca. 133 ms zwischen zwei Datensätzen. Deshalb genügt es, die Daten vom Roboter mit einem Abstand von 130 ms anzufordern. Hierzu verwendet man die DebugKeys-Toolbar. Um die Daten der Selbstlokalisierung und

der Ballerkenner und -modellierer zu erhalten, genügt es, die sogenannten Worldstates anzufordern, da diese alle erforderlichen Daten enthalten.

Die eintreffenden Daten kann man nun in der Feldansicht visualisieren und überprüfen, ob sie auch wirklich ankommen. Dazu muss im Kontextmenü der Feldansicht der Punkt *remote camera world state* ausgewählt werden. Wenn dies der Fall ist, kann die Aufzeichnung der Logdatei fortgesetzt werden. Zum Beenden der Aufzeichnung betätigt man ein weiteres Mal den Aufnahme-Knopf der Logplayer-Toolbar und speichert die Logdatei ab.

C.2.3 Abspeichern der Logdatei in verschiedenen Formaten

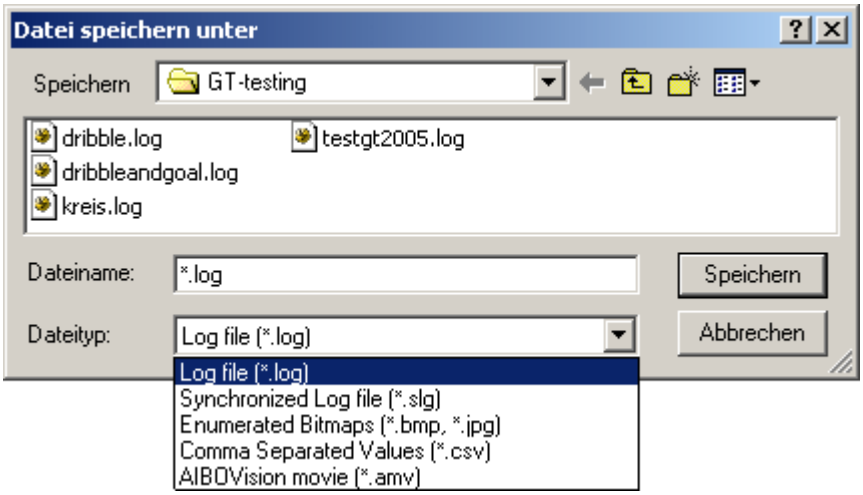

Abbildung C.3: Speicherdialog des Logplayers mit verschiedenen Formaten zur Auswahl

Der Speicherdialog des Logplayers (Abb. C.3) stellt verschieden Dateiformate zur Auswahl. Nach Beendigung einer Aufzeichnung sollte man die Logdatei auf jeden Fall zuerst im Standardformat (Endung .log) speichern. Dadurch bleiben alle aufgezeichneten Daten erhalten. Um Logdateien, die Daten von Roboter und Deckenkamerasystem enthalten, synchron abspielen zu können, muss man diese als Synchronized Log (Endung .slg) speichern. Dabei werden die Datensätze in die richtige Reihenfolge gebracht und die Zeitstempel angepasst.

Um Daten mit anderen Programmen visualisieren oder auswerten zu können, eignet sich die Speicherung als Textdatei mit durch Kommata getrennten Werten (comma separated values, Endung .csv). Jeder Datensatz wird dabei in eine Zeile geschrieben und die verschiedenen Daten wie z.B. Roboter- und Ballposition durch Kommas getrennt. Dieses Format kann z.B. von einem Tabellenkalkulationsprogramm eingelesen und ausgewertet werden.

C.3 Analyse einer synchronisierten Logdatei

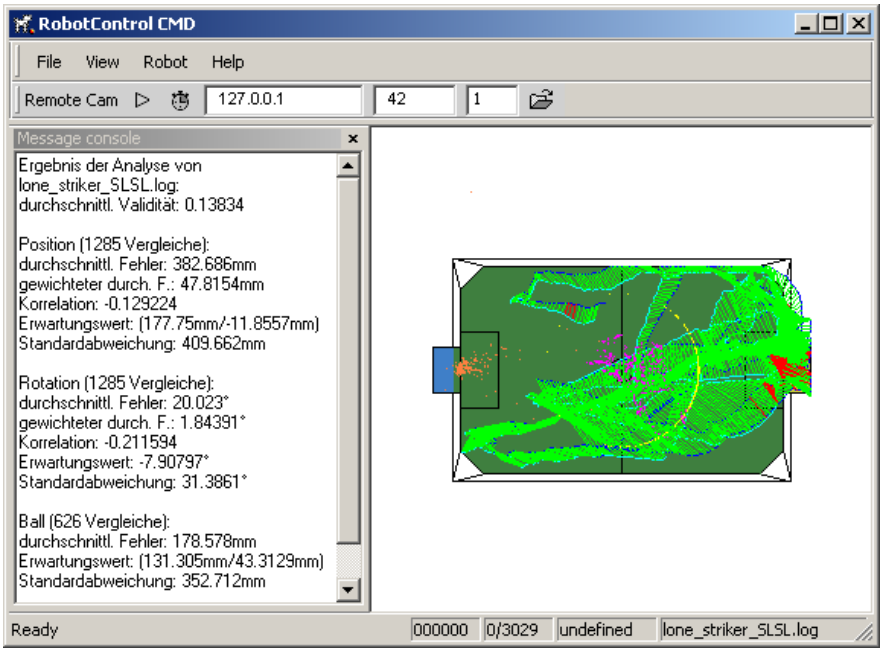

Abbildung C.4: Ausgabe der Loganalyse

Als Beispielanwendung wurde ein einfaches Analyseprogramm implementiert (s. Kapitel 7), das den durchschnittlichen Fehler und die Streuung des Selbstlokators und des Ballerkenners berechnet. Außerdem wird für den Selbstlokator der Zusammenhang mit den übermittelten Validitäten bestimmt und in Form von gewichtetem durchschnittlichen Fehler und Korrelation ausgegeben. Dazu muss man eine Logdatei auf obige Art und Weise erzeugen. Sie muss also neben den Daten von Roboter und Deckenkamerasystem entweder zusätzlich die Synchronisationszeitstempel der beiden Systeme enthalten oder schon als synchronisierte Logdatei vorliegen.

Um eine solche Datei zu analysieren, öffnet man sie mit dem Öffnen-Knopf der RemoteCam-Toolbar. Dadurch wird sofort eine Analyse angestoßen und die Ergebnisse in der Message Console ausgegeben. Zusätzlich wird der Weg des Roboters in der Feldansicht dargestellt und die Verteilung der Positions-, Orientierungs- und Ballfehler mithilfe von Punktverteilungen visualisiert (Abb. C.4). Diese Visualisierungen können im Kontextmenü der Feldansicht mithilfe der Punkte *logfile analyzer lines* und *logfile analyzer dots* aktiviert werden.

C.4 Überblick über die Implementierung

Der Empfang und die Weiterverarbeitung der Daten des Deckenkamerasystems auf Clientseite wurden vollständig in *RobotControl* integriert. Dadurch stehen die Daten bei Bedarf jedem anderen Teil von *RobotControl* zur Verfügung.

C.4.1 RemoteCam

RemoteCam bildet die Schnittstelle zum Deckenkamersystem. Hier findet die Kommunikation mit dem Deckenkamerasystem statt. Außerdem werden die Daten in die Datentypen des *GermanTeam* übersetzt. Jeder andere Teil von *RobotControl* muss nur ein Objekt vom Typ *RemoteCam* mit den richtigen Verbindungsdaten des Kamerarechners anlegen, um damit alle Daten des Deckenkamerasystems abfragen zu können.

Die wichtigsten Methoden von *RemoteCam* sind:

`setConnectionInfos:` Setzt IP und Port des Deckenkamerasystems.

`setIP:` Setzt nur IP des Deckenkamerasystems.

`setPort:` Setzt nur Port des Deckenkamerasystems.

`sendDataRequest:` Sendet Datenanfrage an das Deckenkamerasystem.

`receiveData:` Empfängt die Daten des Deckenkamerasystems.

`getWorldstate:` Liefert komplettes Weltbild in *GermanTeam*-kompatibler Form.

`getPlayerPoseCollection:` Liefert nur die PlayerPoseCollection.

`getBallModel:` Liefert nur das BallModel.

`getRobotPose:` Liefert nur die RobotPose des angegebenen Roboters.

`switchToSyncMode:` Schaltet das Deckenkamerasystem in den Synchronisationsmodus.

C.4.2 RemoteCamToolBar

Die *RemoteCamToolBar* dient als grafische Oberfläche der Schnittstelle zum Deckenkamerasystem. Hiermit können die Verbindungsdaten zum Kamerarechner eingegeben werden. Nach dem Start der Datenanforderung speist diese Toolbar die Daten in die MessageQueues von *RobotControl* ein und diese können dann unter anderem in der Feldansicht angezeigt werden. Außerdem können mit der *RemoteCamToolBar* entsprechend erstellte Logdateien analysiert werden. Die Details zur Bedienung werden in Abschnitt C.1 beschrieben.

Die wichtigsten Methoden von *RemoteCamToolBar* sind:

`OnIdle:` Abfrage der Daten vom Deckenkamerasystem und Weiterleitung in die richtigen MessageQueues.

`analyzeLog:` Analyse einer geeignet aufgezeicheten Logdatei.

`getInterpolatedRobotPose:` Interpoliert eine Roboterposition und -ausrichtung für einen gegebenen Zeitpunkt zwischen zwei gegebenen Roboterpositionen und -ausrichtungen.

`getInterpolatedBallPos:` Interpoliert eine Ballposition für einen gegebenen Zeitpunkt zwischen zwei gegebenen Ballpositionen.

C.4.3 LogPlayer

Dem *LogPlayer*, der zum Aufzeichnen und Abspielen von Logdateien mit *RobotControl* dient, wurde die Methode `saveSynchronized` hinzugefügt. Damit können bei einer Logdatei mit entsprechenden Synchronisierungszeitstempeln die Zeitstempel der Weltbilder angeglichen werden.

C.4.4 GT2004BehaviorControl

In *GT2004BehaviorControl*, der Verhaltenssteuerung des *GermanTeam*, wurde die Methode `postExecute` erweitert. Durch gleichzeitiges Drücken des Kinn- und vorderen Rückentasters wird der Roboter in des Synchronisationsmodus versetzt und sendet beim Erkennen des Synchronisationssignals den Synchronisationszeitstempel. Die Details zur Bedienung des Synchronisationsmodus werden in Abschnitt C.2.1 beschrieben.

C.4.5 ImageBrightnessEstimator

Das *SpecialVision*-Modul *ImageBrightnessEstimator* besitzt nur die Methode `execute`. Diese schätzt die durchschnittliche Helligkeit in der Mitte des Kamerabildes des Roboters ab und dient zur Zeitsynchronisierung mit dem Deckenkamerasystem.

Literaturverzeichnis

[1] Allied Vision Technologies. http://www.alliedvisiontec.com.

[2] J.-Y. Bouguet. Camera calibration toolbox for matlab. Only availaible online: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[3] J. Bunting, S. Chalup, M. Freeston, W. McMahan, R. Middleton, C. Murch, M. Quinlan, C. Seysener, and G. Shanks. Return of the nubots! - the 2003 nubots team report. 2003.

[4] P. Costa, A. Sousa, P. Marques, P. Costa, S. Gaio, and A. Moreira. 5dpo robotic soccer team for year 2003. In *Team Descrptions of the RoboCup 2003 Games and Conferences, Padova, Italy*, 2003.

[5] J. Heikkilä and O. Silvén. A four-step camera calibration procedure with implicit image correction, 1997. http://www.vision.caltech.edu/bouguetj/calib_doc/papers/heikkila97.pdf.

[6] Intek Hard- & Softwaretechnik. http://www.intek-darmstadt.de.

[7] Laws of the F180 League 2004, 2004. Only available online: http://www.itee.uq.edu.au/˜wyeth/F180%20Rules/f180rules300a.pdf.

[8] Th. Röfer, H.-D. Burkhard, U. Düffert, J. Hoffmann, D. Göhring, M. Jüngel, M. Lötzsch, O. v. Stryk, R. Brunn, M. Kallnik, M. Kunz, S. Petters, M. Risler, M. Stelzer, I. Dahm, M. Wachter, K. Engel, A. Osterhues, C. Schumann, and J. Ziegler. GermanTeam RoboCup 2003. Technical report, 2003. Available online: http://www.sim.informatik.tu-darmstadt.de/publ/download/2003-GermanTeamReport.pdf.

[9] Th. Röfer, T. Laue, H.-D. Burkhard, U. Düffert, J. Hoffmann, D. Göhring, M. Jüngel, M. Lötzsch, M. Spranger, B. Altmeyer, V. Goetzkeand O. v. Stryk, R. Brunn, M. Dassler, M. Kunz, M. Risler, M. Stelzer, D. Thomas, S. Uhrig, U. Schwiegelshohn, I. Dahm, M. Hebbel, W. Nisticó, C. Schumann, and M. Wachter. GermanTeam RoboCup 2004. Technical report, 2004. Available online: http://www.sim.informatik.tu-darmstadt.de/publ/download/2004-GermanTeamReport.pdf.

[10] M. Simon, S. Behnke, and R. Rojas. Robust real time color tracking. In *RoboCup 2000: Robot Soccer World Cup IV, P. Stone, T. Balch, and G. Kraetszchmar (Eds.)*, number 2019 in Lecture Notes in Artificial Intelligence, pages 239–248. Springer, 2001.

[11] G.F. Wyeth and B. Brown. Robust adaptive vision for robot soccer. In *Mechatronics and Machine Vision in Practice, J. Billingsley (Ed.)*, pages 41–48. Research Studies Press, 2000.