Roboter-Fußball: Zuverlässige Ballerkennung und Positionsschätzung

Diplomarbeit an der Universität Freiburg

Fakultät für Angewandte Wissenschaften

vorgelegt von:
Maximilian Thiel

Betreuer: Prof. Dr. B. Nebel

Freiburg, den 15. Dezember 1999

Zusammenfassung

Roboter-Fußball stellt eine Herausforderung für Robotik, Bildverarbeitung und Künstliche Intelligenz gleichermaßen dar. In Wettkampfatmosphäre werden auf sozusagen *spielerische* Art und Weise Lösungsansätze für das interdisziplinäre Gebiet der autonomen Robotik vorgestellt und deren Leistungsfähigkeit bewertet.

Diese Arbeit beschreibt die Methoden der Ballerkennung und Ballpositionsschätzung des CS-Freiburg-RoboCup-Teams. Es wird gezeigt, wie auch unter schwierigen Lichtbedingungen eine weitgehend robuste Ballerkennung und somit eine zuverlässige Schätzung der Ballposition realisiert werden kann.

Das verwendete Bildverarbeitungssystem besitzt eine hardwareimplentierte Farbsegmentierung. Diese wird so parametrisiert, daß sie Objekte in den Kamerabildern, deren Farbe mit der des Balles ähnlich ist, segmentiert. Aus der Menge dieser Objekte wird anhand verschiedener Plausibilitätstests das den Ball repräsentierende Objekt ermittelt. Schließlich wird aufgrund von zuvor erstellten Kalibrierungsdaten die Position des Balles auf dem Spielfeld bestimmt.

Das Konzept des Bildverarbeitungssystems des CS-Freiburg-RoboCup-Teams hat sich bewährt, was auch durch den Erfolg bei den Deutschen Meisterschaften 1999 im Roboter-Fußball seine Bestätigung fand.

Danksagung

Ich möchte mich bei all den Menschen bedanken, die mich bei meiner Arbeit unterstützt und betreut haben. Besonderer Dank geht an Bernhard Nebel, Steffen Gutmann und Wolfgang Hatzack, die trotz Robotik-Umfeldes noch sehr menschliche Regungen und Herzlichkeit gezeigt haben. Für die äußerst kompetente und fachliche Unterstützung möchte ich Stefan Rahmann danken. Weiterhin bedanke ich mich für das hervorragende Korrekturlesen bei Julia und Ralf Gotzel, Holger Keibel sowie Thilo Weigel. Herzlichen Dank gilt Christian Reetz, Burkhard Dümler, Boris Bauer und Andreas Hill für ihre moralische Unterstützung. Ebenso bedanke ich mich bei Markus Dietl für seine unermessliche Geduld. Für die technische Unterstützung sei Bruno Welsch gedankt. Bedanken möchte ich mich ebenso bei Augustinus Topor, der immer wieder für Spannung und Unterhaltung sorgte. Vielen Dank auch an Immanuel Hermann, dem „Farbklecks“ des CS-Freiburg-RoboCup-Teams. Und schließlich sei noch dem „Star “ unter den Spielern gedankt: Quasimo.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Zielsetzung 2

1.2 Aufbau der Arbeit 3

2 Ansätze der Bildverarbeitung **5**

2.1 Allgemeines 5

2.2 Die verwendeten Kamerasysteme 7

2.3 Die verwendeten Farbräume 9

2.4 Farbsegmentierung, Objekterkennung, Positionsbestimmung . 10

3 Hardware **18**

3.1 Allgemeines 18

3.2 Gesamtsystem 19

3.2.1 Notebook 19

3.2.2 Laserscanner 19

3.2.3 Bildverarbeitungssystem 19

3.2.4 Schußmechanismus 20

3.2.5 Zentralrechner 20

4 Ballerkennung im Überblick **22**

4.1 Farbsegmentierung auf dem Visionboard 23

4.2 Bildauswertung auf dem CPU-Board 24

4.3 Positionsschätzung auf dem Notebook 26

5 Das Bildverarbeitungssystem **28**

5.1 Das Cognachrome Vision-System 28

5.2 Das Vision-Board 29

5.2.1 Die Farbtabellen des Vision-Boards 30

5.2.2 Einschränkung durch das Farbtabellendesign 31

5.2.3 Initialisieren der Farbtabelle 32

5.2.4 Glanzlichter, Schatten und Geisterbälle 33

5.3 Das CPU-Board 34

6 Das PioneerVision-Programm **36**

6.1 Die Programm-Struktur 37

6.2 Auswertung der Binärbilder 37

6.2.1 Blob-Validierung 38

6.2.2 Blob-Auswahl 41

6.2.3 Das Dreiecksverfahren 42

6.3 Das Vision-Informationspaket 45

6.3.1 Erzeugen eines Vision-Informationspakets 45

6.3.2 Senden eines Vision-Informationspakets 46

7 Das Saphira-Programm **47**

7.1 Das Saphira-System 48

7.1.1 Saphira-System-Architektur 48

7.1.2 Rauschen des Kamerabildes 49

7.2 Mittelung 49

7.2.1 Mittelung des Mittelpunktes 50

7.2.2 Kontur-Mittelung 51

7.2.3 History-Tabelle 51

7.3 Mittelpunktsberechnung mittels Kreisfitting 52

7.4 Positionsberechnung 53

7.4.1 Positionstabelle 54

7.4.2 Verifikation 54

7.4.3 Ballposition bezüglich des Roboters 55
7.4.4 Bestimmung der globalen Ballposition 56
7.4.5 Geschwindigkeitsschätzung des Balles 56
7.5 Fusionierung der Ballpositionsdaten 57
7.5.1 Das globale Weltmodell 57

8 Kalibrierung **60**
8.1 Ermittlung der Kalibrierungsdaten 60
8.2 Auswahl der Meßpunkte 61
8.3 Interpolation 62
8.3.1 Der Operator $\otimes$ 63
8.3.2 Die Interpolationsformel 64

9 Kreisfitting **66**
9.1 Der Algorithmus 66
9.2 Der Abstandsfehler F_c 70

10 Diskussion **71**
10.1 Referenzpunkt 71
10.1.1 Mittelwert und Varianz 72
10.1.2 Mittelung 73
10.1.3 Ergebnisse der Messreihe 73
10.2 Validierung 74
10.3 Die Optimierung der Visionpakete 75
10.4 Spezielle Datenpakete 76

11 Zusammenfassung und Ausblick **78**
11.1 Zusammenfassung 78
11.2 Ausblick 79

A Glossar **82**

B Histogramme aus den Messreihen **83**

B.1 Die erste Messreihe 83
B.2 Die zweite Messreihe 87

C Die Struktur der Datenpakete **91**
C.1 Grundstruktur der Datenpakete 91
C.2 Das Vision-Informationspaket im *PioneerVision-98*-Programm 92
C.3 Das Vision-Informationspaket im *PioneerVision-99*-Programm 93
C.3.1 Der normale Blob-Modus 93
C.3.2 Der Blob-Outline-Modus 95
C.3.3 Der *Debug*-Blob-Modus 99
C.4 Das Vision-Steuerungspaket 99
C.5 Kommunikation mit dem Motorcontroller 100
C.5.1 Empfangen und Senden von Motor-Informationspaketen 100
C.5.2 Empfangen und Senden von Motor-Steuerpaketen . . 101

D Softwarebeschreibung **102**
D.1 Das Debug-Fenster 102
D.2 Parametereinstellungen 105
D.2.1 Ballerkennung und Kalibrierung 105
D.2.2 Kamerabild-Kalibrierung und Debug-Einstellung . . . 106
D.2.3 Parameter für die Blob-Validierung 107
D.2.4 Obere Schranke 108
D.2.5 Mittelung 108
D.2.6 Kreisfitting 109
D.2.7 Training der Farbtabelle 110
D.3 Die einzelnen Schritte der Kalibrierung 112

E Resultate 1999 **115**
E.1 Technical Challenge 115
E.2 Resultate 116
E.2.1 Weltmeisterschaft RoboCup 1999 116
E.2.2 Deutsche Meisterschaft RoboCup 1999 117

Kapitel 1

Einleitung

Vom 27. Juli bis zum 6. August 1999 fand im Rahmen der *IJCAI 99* (International Joint Conference on Artificial Intelligence) zum dritten Male die Roboter-Fußball-Weltmeisterschaft statt. Auch wenn der Wettbewerbs-Charakter sehr stark in den Vordergrund gerückt wurde, basiert die ursprüngliche Idee des Roboter-Fußballs auf der Erforschung und Lösung einer Vielzahl von Standard-Problemen in den Bereichen der Künstlichen Intelligenz, der Robotik, der Multiagentensysteme und der Bildverarbeitung innerhalb eines komplexen Gesamtsystemes.

Um Schwerpunkte auf unterschiedliche Forschungsgebiete legen zu können, wurden verschiedene RoboCup-Ligen geschaffen. Derzeit existieren vier Klassen: eine Simulations-Liga, die Klasse der Roboter kleiner Größe, die Klasse der Roboter mittlerer Größe und die Sony-Roboter-Liga. Das CS-Freiburg-RoboCup-Team spielt in der Klasse der Roboter mittlerer Größe (siehe Abbildung 1.1), bei der pro Mannschaft vier Roboter mit einem maximalen Durchmesser von 50 cm auf einem Spielfeld der Größe von etwa 9m x 5m gegeneinander antreten. Die Roboter agieren autonom, d.h. sie sind auf die Daten ihrer eigenen Sensorik (Sonare, Kamera, Laserscanner, Infrarotsensoren etc.) angewiesen. Zusätzlich ist es aber auch möglich, beliebige Informationen über Funk auszutauschen.

In den Spielen hat sich bisher gezeigt, daß gerade die Sensorik einen entscheidenden Beitrag zur Überlegenheit einer Mannschaft leistet. Nicht zuletzt dank der genauen Selbstlokalisation mittels Laser-Scanner ist es dem CS-Freiburg-RoboCup-Team gelungen, im Roboter-Fußball national sowie international einen Spitzenplatz einzunehmen.

Während viele andere Mannschaften der „Middle-Size-League“ die Selbstlo-

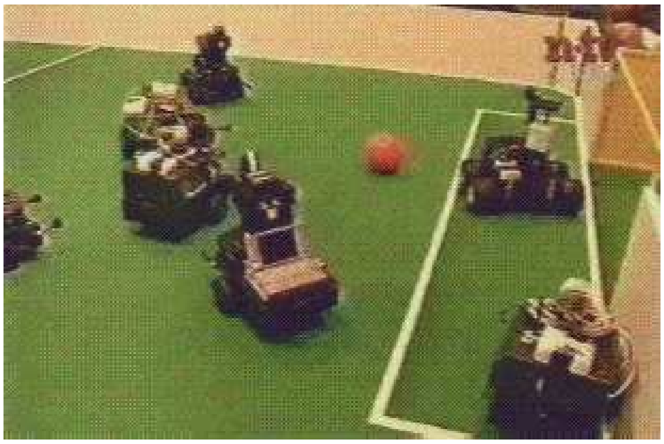

Abbildung 1.1: Ein Spiel des CS-Freiburg-RoboCup-Teams

kalisation, das Erkennen eigener wie auch gegnerischer Feldspieler, die Positionsschätzung des Balles sowie die Ortung der Tore über ihr Bildverarbeitungssystem realisieren, benutzt das CS-Freiburg-RoboCup-Team das Bildverarbeitungssystem ausschließlich zur Ballerkennung und Ballpositions-Schätzung – die anderen Aufgaben werden durch den verwendeten Laser-Scanner verwirklicht [Weigel, 1999]. Das heißt aber keinesfalls, daß die Anforderung an das Bildverarbeitungssystem zu einem trivialen Problem geworden ist. Gerade die ständig wechselnden Lichtbedingungen bei den Spielen wie auch unerwünschte Nebeneffekte wie Glanzlichter und Schatten stellen für das Bildverarbeitungssystem eine erhebliche Herausforderung dar. Dies wurde spätestens allen Beteiligten des CS-Freiburg-RoboCup-Teams klar, als sie – in der regulären Spielzeit zwar ungeschlagen – in Stockholm im Halbfinalspiel u.a. wegen Probleme des Bildverarbeitungssystems ausschieden und den Weltmeistertitel an den neuen Herausforderer aus dem Iran abgeben mußten.[1]

1.1 Zielsetzung

Ende 1997 wurde in der Abteilung „Grundlagen der Künstlichen Intelligenz“ an der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-

[1]Der Titel des Deutschen Meisters konnte in Stuttgart bei der *Vision-99* erfolgreich verteidigt werden

Universität Freiburg die Entscheidung getroffen, bei der Roboter-Fußball-Weltmeisterschaft RoboCup 98 in Paris teilzunehmen, wo die Mannschaft auch gleich den Weltmeister-Titel errang. Dennoch zeigte sich bei Tests und den Spielen, daß *die Ballerkennung und die Positionsbestimmung aber bei weitem noch nicht stabil genug* war [Topor, 1998]. Da die mit dem Bildverarbeitungssystem mitgelieferte Software unverändert eingesetzt wurde, waren Beschränkungen auferlegt, die nur mit einer Modifizierung der Software zu umgehen waren.

Daher wurde als Ziel gesetzt, die Software des Bildverarbeitungssystems den Ansprüchen des CS-Freiburg-RoboCup-Teams anzupassen. Einerseits sollte der Durchsatz des Datentransfers zwischen dem Bildverarbeitungssystem und anderen Hardwarekomponenten erhöht werden, andererseits sollte zusätzlich eine Konturenerkennung realisiert werden. Auch eine Synchronisation der Sensordaten mittels Zeitstempel sollte zu einer Verbesserung der Ballpositionsschätzung führen. Die meisten der Forderungen wurden implementiert und werden in den folgenden Kapiteln dargestellt.

1.2 Aufbau der Arbeit

Diese Arbeit beschreibt, wie unter Einsatz des Bildverarbeitungssystems des CS-Freiburg-RoboCup-Teams eine robuste Ballerkennung und Positonsschätzung realisiert werden kann.

Zunächst wird ein Überblick über die Bildverarbeitungssysteme gegeben, wie sie von bestimmten Mannschaften der „Middle-Size-League“ eingesetzt wurden. Auf die verschiedenen Farbsysteme, die verwendet wurden, wird gesondert eingegangen.

Kapitel 3 gibt eine kurze Übersicht über die verwendeten Hardwarekomponenten des CS-Freiburg-RoboCup-Teams.

In Kapitel 4 wird anhand eines Beispieles das Verfahren der Ballerkennung und Ballpositionsschätzung skizziert.

Auf das benutzte Bildverarbeitungssystem wird in Kapitel 5 eingegangen, das aus Kamera, einem Framegrabber und einem CPU-Board besteht. Neben dem Aufbau des Systems werden auch die implementierte Farbsegmentierung und die damit verbundenen Schwierigkeiten erläutert. Es wird beschrieben, wie mittels dieses Bildverarbeitungssystems Objekte im Kamerabild erkannt werden können.

Kapitel 6 beschäftigt sich mit dem Bildauswertungsprogramm auf dem CPU-

Board. Es werden spezielle Verfahren zur Überprüfung und Auswertung der im Kamerabild erkannten Objekte vorgestellt. Dabei wird ein zeiteffizienter Algorithmus besprochen, mit dessen Hilfe zu einem vorgegebenen Kreissegment der zugehörige Kreis-Mittelpunkt berechnet werden kann.

Das Kapitel 7 behandelt die Ballpositionsschätzung mit Hilfe der Daten, die vom CPU-Board bereitgestellt werden. Es werden zunächst verschiedene Verfahren besprochen, die zu einer robusten Ballpositionsschätzung beitragen. Anschließend wird ein weiterer Algorithmus zur Mittelpunktsberechnung beschrieben, der zu ausgewählten Randpunkten eines Objektes einen Kreis optimal schätzt. Als nächstes folgt eine Beschreibung, wie die Ballposition auf dem Spielfeld mittels einer Positionstabelle bestimmt wird. Abschließend wird dargestellt, wie die Ballposition eines Spielers mit den ermittelten Ballpositonen der anderen Spieler abgeglichen wird.

In Kapitel 8 wird das Verfahren erläutert, wie die für die Positionstabelle notwendigen Daten generiert werden. Die bei dem Verfahren verwendete Interpolationsmethode wird dabei in diesem Kapitel ausführlich behandelt.

Kapitel 9 liefert den mathematischen Hintergrund des in Kapitel 7 vorgestellten Kreisfittingverfahrens.

In Kapitel 10 werden die Verfahren der Ballerkennung und Ballpositionsschätzung des RoboCup '99-Programms diskutiert und mit den Verfahren des RoboCup '98-Programms verglichen.

Kapitel 11 stellt eine Zusammenfassung und einen Ausblick dar. Dieses Kapitel beschäftigt sich u.a. mit der Frage, wie und inwieweit eine weitere Verbesserung des Bildverarbeitungssystems möglich ist.

Die Arbeit endet mit einer Softwarebeschreibung, in der die Anwendung des im Rahmen dieser Diplomarbeit erstellten Programmes kurz beschrieben wird.

Kapitel 2

Ansätze der Bildverarbeitung

Dieses Kapitel stellt die Bildverarbeitungssysteme anderer Teams vor, die ebenso bei der Roboter-Fußball-Weltmeisterschaft RoboCup '99 in der „Middle-Size-League“ teilgenommen haben. Zunächst wird ein allgemeiner Überblick über die verwendeten Systeme gegeben. Dann werden die bildverarbeitungsspezifischen Hardware-Komponenten betrachtet, die von den verschiedenen Teams eingesetzt wurden. Anschließend werden die Eigenschaften der verwendeten Farbräume besprochen. Zum Schluß wird anhand von Beispielen gezeigt, wie die Erkennung und Positionsbestimmung verschiedener Objekte auf dem Spielfeld mit Hilfe der von den Systemen bereitgestellten Bilddaten realisiert wurde.

2.1 Allgemeines

Alle Teams der „Middle-Size-League“ verwendeten PAL- oder NTSC-Farb-CCD-Kameras. Einige der Teams montierten die Kameras fix auf den Robotern. Andere wiederum setzten schwenkbare oder sogenannte omnidirektionale 360°-Kameras ein. Die Kameras wurden mit einem Framegrabber verbunden. Die dort eingehenden analogen NTSC- oder PAL-Kamerasignale wurden in digitale RGB-, HSI-, HSV- oder YUV-Signale umgewandelt (siehe Abschnitt 2.3). Auf dem daraus entstehenden Farbbild[1] wurde anschließend eine Farbsegmentierung auf Basis eines zuvor erstellten Farbklassifizierers

[1] In dieser Arbeit wird dieses Farbbild auch als *Kamerabild* bezeichnet.

durchgeführt, um die Objekte im Kamerabild identifizieren zu können. Die Objekterkennung über die Farbe ist deshalb möglich, da bei RoboCup die Farben des Balles (orange), der Tore (gelb und blau), des Bodens (grün) sowie der Bande (weiß) und der Spieler (schwarz) vor dem Spiel feststehen. Einige Teams führten darüberhinaus eine Konturenerkennung durch, um genauere Informationen aus den Bilddaten zu erhalten. Das traf insbesondere für Teams zu, die ihre Selbstlokalisation mit Hilfe ihres Bildverarbeitungssystems realisierten.

Für die im Kamerabild erkannten Objekte wurde anschließend die relative Position zum Roboter errechnet. Ist die Position des Spielers im Spielfeld bekannt, wurde aus dieser die Position des Objektes im Spielfeld bestimmt. Bei einigen Teams wurde die so ermittelte Position mit den errechneten Positionen anderer Spieler bezüglich des gleichen Objektes mittels eines Zentralrechners abgeglichen. Sollte die Geschwindigkeit und Richtung von Objekten bestimmt werden, geschah dies anhand der Daten aus aufeinanderfolgenden Bildern.

Abbildung 2.1: Spieler des „GMD“-Teams mit schwenkbarer Kamera

2.2 Die verwendeten Kamerasysteme

Viele Teams montierten ihre Kamera fix auf die Spieler. Das Problem hierbei war, daß sich der gesamte Roboter drehen mußte, sobald die Blickrichtung geändert werden sollte. Um dies zu vermeiden, montierten einige Teams wie die „Ulm Sparrows", die „GMD-Robots" und das „Matto"-Team [Demura *et al.*, 1999] schwenkbare Kameras auf ihre Roboter. So waren die Roboter in der Lage, den Ball zu verfolgen ohne sich selbst bewegen zu müssen. Besonders bemerkenswert war die Realisierung der Kameraführung bei den Robotern des „GMD"-Teams, die es ermöglichte, selbst Bälle mit großer Geschwindigkeit präzise verfolgen zu können, ohne sie aus dem Blickfeld zu verlieren (siehe Abbildung 2.1).

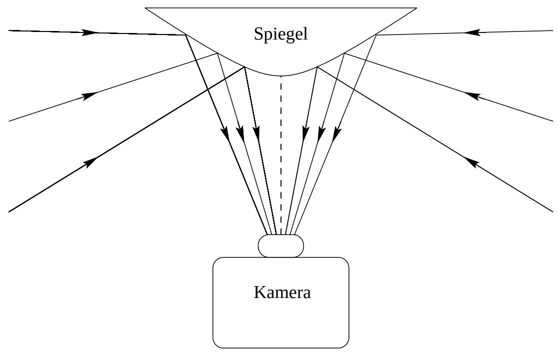

Abbildung 2.2: Omnidirektionale Kamera

Andere Teams, wie das „ISocRob"-Team [Ventura *et al.*, 1999], das „Patriarcas"-Team [Sampaio *et al.*, 1999] und das „Trackies 99"- Team, verwendeten sogenannte omnidirektionale Kameras, d.h. Kameras mit einem Sichtfeld von 360° (siehe Abbildung 2.3). Bei diesem System wurde ein rotationssymmetrischer konvexer Spiegel so auf den Roboter montiert, daß sich in ihm die gesamte Umgebung des Roboters innerhalb eines bestimmten Radius' spiegelte. Der Radius des Sichtfeldes war dabei abhängig von der Montage und Form des Spiegels. Unterhalb des Spiegels war eine Kamera so angebracht, daß sie die gesamte Fläche des Spiegels und somit die Umgebung des Roboters aufnehmen konnte (siehe Abbildung 2.2). Der Vorteil dieses Systems war, daß die Roboter das gesamte Spielfeld stets im Blickfeld hatten, ohne sich oder die Kamera bewegen zu müssen. Aufgrund der größeren Informationsdichte im Kamerabild lieferte die Auswertung der Daten, insbesondere die von etwas weiter entfernt liegenden Objekten, oftmals ungenaue

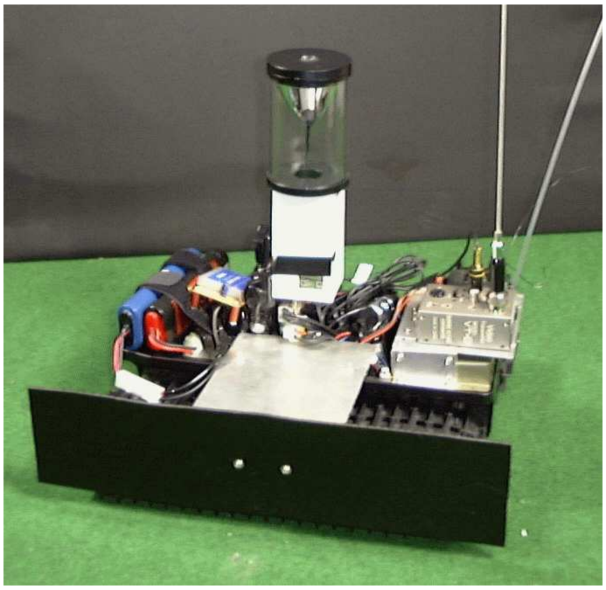

Abbildung 2.3: Torwart des „Trackies 99“- Teams mit omnidirektionaler Kamera

Ergebnisse.

Eine Möglichkeit, dieses Problem zu umgehen, wurde von dem „Attempto“- und dem „NAIST“- RoboCup Team realisiert. Diese montierten zusätzlich zur omnidirektionalen Kamera eine zweite auf die Vorderseite der Roboter. Das „Attempto“-Team [Plagge *et al.*, 1999] installierte eine fixe und das Team von „NAIST“ [Nakamura *et al.*, 1999] eine schwenkbare Kamera. Dies ermöglichte ihnen, die vor dem Roboter liegenden Objekte inklusive ihrer Position präziser zu bestimmen und gleichzeitig einen Überblick über das gesamte Spielfeld zu besitzen.

Das „CoPS“-Team aus Stuttgart wie auch das Team „Azzuro“ aus Italien befestigten mehrere Kameras auf ihren Torwart. Zur besseren Kontrolle des Umfeldes montierten die Italiener zwei und die Stuttgarter drei Kameras und hatten damit ein horizontales Sichtfeld von nahezu 200° bzw. 300°, wobei sich jeweils etwa 20° des Sichtfeldes der Kameras überschnitten [Levi *et al.*, 1999, Piaggio *et al.*, 1999].

Das italienische Team besaß darüberhinaus einen zweiten Torwart, auf dem es eine omnidirektionale Kamera befestigte, mit der der Roboter einerseits den Ball verfolgen und andererseits die Torposition berechnen konnte [Cassinis *et al.*, 1999]. Die anderen Spieler des italienischen Teams wurden sowohl mit omnidirektionalen, fixen, als auch mit schwenkbaren Kameras ausgestattet.

2.3 Die verwendeten Farbräume

Die von den Kameras gelieferten analogen PAL- oder NTSC-Signale wurden von den Bildverarbeitungssystemen in eine für den Computer lesbare digitale Form umgewandelt. Die Speicherung der Farbdaten hing von dem zugrundeliegenden Farbsystem ab.

Jeder Farbwert eines Pixels im Kamerabild läßt sich aus den drei Primärfarben Rot, Grün und Blau durch additive Farbmischung darstellen. Diese Eigenschaft macht man sich in dem RGB[2]-Farbraum zu Nutze. Soll die Intensität von jedem der Farbkomponenten Rot, Grün und Blau in je einem Byte gespeichert werden, so können pro Farbkomponente 256 verschiedene Intensitäten dargestellt werden. Da die gewählten Primärfarben linear unabhängig sind, lassen sich mit Hilfe von drei Bytes $256^3 = 16.777.216$ verschiedene Farben darstellen.

Eine der großen Schwierigkeiten bezüglich der Bildauswertung während der RoboCup-Spiele waren die sich ständig ändernden Beleuchtungsverhältnisse. Je nach Beleuchtung wurden Objekte wie z.B. der orange Ball im Kamerabild mit unterschiedlichen Farbwerten dargestellt.

Soll nun ein Objekt im Kamerabild unter all den vielfältigen Beleuchtungssituation erkannt werden, gibt es die Möglichkeit, eine dreidimensionale Farbtabelle zu erzeugen, die mit den RGB-Werten indiziert ist. Zu jedem RGB-Wert kann in der Tabelle gespeichert werden, ob der entprechende Farbwert das Objekt repräsentiert oder nicht. Wird das Kamerasignal als drei Byte großer RGB-Wert gespeichert, so hat die Tabelle $256^3 = 16.777.216$ Einträge.

Anstatt einer Farbsegmentierung mittels Farbtabelle könnte man auch das gesamte mögliche Farbspektrum des Objektes im Kamerabild mathematisch formulieren, denn das Farbspektrum des Objektes bildet im RGB-Raum einen Unterraum. Doch dieser Unterraum des RGB-Raumes hat eine Struktur, die sich auch in ihrer groben Form mathematisch nicht einfach beschreiben läßt.

[2]RGB=Red,Green,Blue

Daher gingen viele Teams dazu über, den RGB-Farbraum in einen Farbraum zu transformieren, bei denen einer der drei Basisvektoren im wesentlichen die Helligkeit beschreibt und die anderen zwei Basisvektoren dagegen weitgehend beleuchtungsinvariant sind. Die Beschreibung des Farbspektrums des Objektes im Kamerabild kann damit auf einen zweidimensionalen Farbraum beschränkt werden. In diesem Farbraum lassen sich nun weitgehend unabhängig von der Beleuchtung die relevanten Objekte beschreiben. Die verwendeten Farbräume waren der HSI[3]-, HSV[4]- bzw. der YUV-Farbraum, die sich bezüglich der Beleuchtungsinvarianz im wesentlichen nicht unterscheiden. Die Abbildung 2.4 zeigt den HSV-Farbraum. Mehr Informationen über Farbräume können z.B. in der Arbeit von Siggelkow [Siggelkow, 1998] nachgelesen werden.

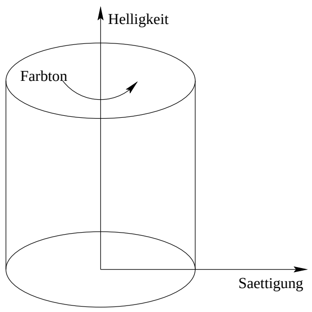

Abbildung 2.4: Der HSV-Farbraum (Hue-Farbton, Saturation-Sättigung, Value-Helligkeit)

2.4 Farbsegmentierung, Objekterkennung, Positionsbestimmung

Bezüglich der Bildauswertung waren die Anforderungen der Teams an das Bildverarbeitungssystem recht unterschiedlich. Grundsätzlich ließen sich die System-Anforderungen in folgende fünf Bereiche klassifizieren:

- Nur Ballerkennung (CS-Freiburg)
- Das Erkennen des Balles und der Tore (Alpha++)

[3] HSI = Hue – Saturation – Intensity
[4] HSV = Hue – Saturation – Value

- Das Erkennen des Balles, der Tore und der Spieler (CS-Sharif, Dreamteam, Patriarcas, Wisely)
- Das Erkennen des Balles, der Tore, der Mitspieler und der Gegner (Matto, NAIST, Attempto)
- Das Erkennen des Balles, der Tore, der Mitspieler und der Gegner, sowie eine aufwendige Selbstlokalisation auf Basis von Kantenextraktion (Agilo, Azzurra).

Im folgenden wird auf die unterschiedlichen Verfahren einzelner Teams zur Objekterkennung näher eingegangen:

Agilo

Das „Agilo"-Team aus München arbeitete mit sehr hochwertigen Bildverarbeitungskomponenten. Auch ihre Bildauswertung zeichnete sich durch einen hohen Standard aus. Das von der Kamera eintreffende PAL-Signal wurde hierbei zunächst in das YUV-Bildformat umgewandelt. Da die Verarbeitung des daraus resultierenden 768×572 Pixel-Bildes zu zeitaufwendig gewesen wäre, wurde einerseits die Auflösung des Kamerabildes halbiert und andererseits die oberen 40 Prozent des Bildes ignoriert. Das Ergebnis war ein zweidimensionales Bild mit 384×172 Pixeln Größe. Die Verarbeitungszeit pro Bild betrug nur noch ca. 100 ms. Pro Sekunde konnten bis zu 10 Bilder ausgewertet werden.

Die Farbsegmentierung erfolgte mit einem zuvor erstellten zweidimensionalen Farbklassifizierer (eine 256×256 große Tabelle), welcher über die UV-Werte des YUV-Signales indiziert war. Da aber ein zu hoher oder ein zu niedriger Y-Wert den Farbwert stark verändert, wurde aus Validitätsgründen in der Tabelle außer des Klassifikation noch zu jedem UV-Wert der minimal- und maximalgültige Helligkeitswert Y gespeichert. Befand sich der Y-Wert eines YUV-Signales innerhalb des gültigen Spektrums, wurde das YUV-Signal einer Farbklasse (farblos, schwarz, weiß, grün, blau, cyan, magenta, rot und gelb) eindeutig zugeordnet. Die Initialisierung der Tabelle geschah mit einem interaktiven Klassifizierungs-Tool, mit dem anhand verschiedener Bilder des aktuellen Spielfeldes die entsprechenden Farbklassen „erlernt" wurden.

Das aus der Farbsegmentierung resultierende Bild war gewöhnlich in verschiedene Farbregionen unterteilt. Mittels Kantenfilter und einer Datenbank, in der die relevanten Objekte mit Form und Farbe gespeichert waren, wurden die Farbsegmente den Objekten auf dem Spielfeld zugeordnet. Zur Positionsbestimmung der Objekte war die Kamera auf dem Roboter so befestigt, daß ihre optische Achse parallel zum Spielfeld verlief. Da die genaue Position und Orientierung der Kamera, sowie die internen Kameraparameter wie Pixelabstände, Brennweite und Bildhauptpunkt, bekannt waren, wurde anhand des untersten Punktes P eines Farbsegmentes im Kamerabild die maximale Entfernung des entsprechenden Objektes zum Roboter berechnet. Die minimale Entfernung wurde aus dem höchsten Punkt P' im Kamerabild berechnet, der unterhalb von P liegt und als Teil des Spielfeldes klassifiziert wurde. Über eine Funkverbindung mit einem Zentralcomputer wurden die errechneten Objektpositionen mit denjenigen anderer Spieler abgeglichen und damit die Position des Objektes auf dem Spielfeld bestimmt.

Das „Agilo"-Team verwendete das Bildverarbeitungssystem auch zur Selbstlokalisation. Mittels der relativen Position der Banden und – sofern sichtbar – der Tore, die durch Einpassen (Matchen) der extrahierten Banden und Torlinien in ein gespeichertes 3D-Spielfeld-Modell ermittelt wurde, wurde die eigene Position des Roboters im Spielfeld berechnet und mit den Odometriedaten fusioniert. Die Dauer für eine Selbstlokalisation wurde mit 5 bis 30 ms angegeben [Schmitt *et al.*, 1999, Bandlow *et al.*, 1999].

Azzurra

Das italienische Team wurde von Mitgliedern sieben verschiedener Universitäten gebildet. Jede der Universitäten hatte ein eigenes Entwicklungsteam und stellte genau einen Spieler zur Verfügung. Da die Entwicklungsteams verschiedene Forschungsschwerpunkte hatten, waren die Spieler unterschiedlich aufgebaut. Das traf insbesondere für den Bereich der Bildverabeitung zu. Im folgenden werden die Bildverarbeitungssysteme von drei der sieben Universitäten vorgestellt.

Palermo

Das Team aus Palermo ging in der Objekterkennung einen eigenen Weg. Im Gegensatz zu allen anderen italienischen Teams, die mit einem manuell generierten Farbklassifizierer arbeiteten, wurde die Farbklassifizierung mittels „Feed-Forward"-Neuronaler Netze realisiert. Die Motivation für die Benutzung des neuronalen Netzes lag in dem Umstand, daß eine exakte statistische Aufteilung des Farbraums bezüglich der zu erkennenden Objekte, aufgrund der sich ständig wechselnden Lichtverhältnisse nicht realisierbar war. Vielmehr war das neuronale Netz in der Lage, mit ungenauen Daten, die zum Beispiel durch ein Rauschen im Kamerabild verursacht wurden, umzugehen. Mit Hilfe des neuronalen Netzes sollte dieses Bildverarbeitungssystem in der Lage sein, den Farbbereich relevanter Objekte je nach Beleuchtungs-Situation zu erlernen.

Bemerkenswert ist, daß die Objekterkennung anstatt im drei-, bzw. zweidimensionalen nur in einem eindimensionalen Farbraum stattfand. Das Kamerasignal wurde dabei zunächst in den HSI-Farbraum transformiert. Anschließend wurde für das Erlernen bzw. für die Auswertung des neuronalen Netzes lediglich der Hue-Wert des HSI-Signales verwendet [Amoroso *et al.*, 1999].

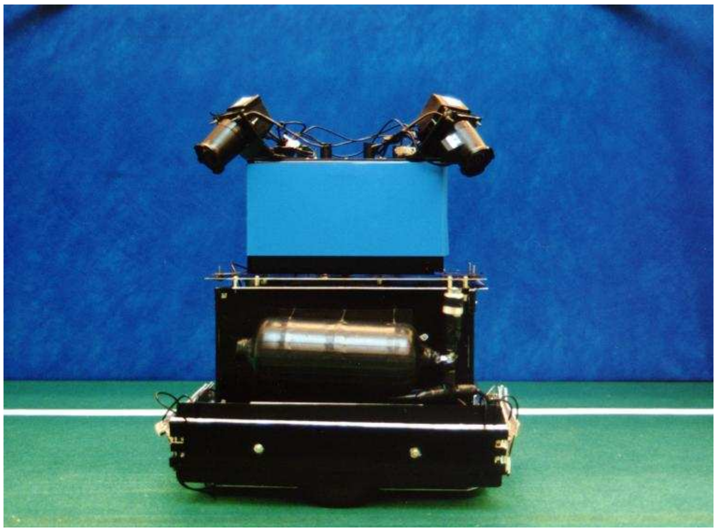

Abbildung 2.5: Italienischer Torwart mit zwei Kameras

Parma

Das Team aus Parma stellte den Torwart, auf dem zwei Kameras montiert waren. Es wurde ein spezieller Switch-Algorithmus implementiert, der von dem Framegrabber einer Kamera zu dem Framegrabber der anderen schaltete, wenn der Ball von keiner der beiden Kameras „gesehen“ wurde oder wenn der Ball aus dem Sichtfeld der einen Kamera in das Sichtfeld der anderen Kamera rollte. Neben der Ballverfolgung (Tracking) wurde auch die Selbstlokalisierung des Torwartes mit Hilfe des Bildverarbeitungssystems realisiert. Diese basierte auf der Lokalisierung der Linien im Strafraum, deren Positionen vor dem Spiel fest vorgegeben waren [Piaggio *et al.*, 1999].

Mailand

Das Team aus Mailand arbeitete mit einer omnidirektionalen Kamera, mit der 25 Bilder pro Sekunde bei einer Auflösung von 512×582 Pixel ausgewertet wurden. Die Farbsegmentierung geschah mittels des aus den Kamerasignalen erzeugten HSV-Raumes. Dabei wurde das 512×582 große Kamerabild gleichmäßig in 3×3 Pixel große „Rezeptoren" unterteilt. Dies ermöglichte eine schnellere Auswertung der Bilddaten. Als Farbwert des Rezeptors wurde der durchschnittliche Wert seiner neun Pixel verwendet. Die Farbsegmentierung erfolgte anhand der Farbklassifizierungsdaten, die auf dem durchschnittlichen Helligkeitswert des Kamerabildes basierten. Da das Bild einer omnidirektionalen Kamera ein großen Teil des Spielfeldes wiedergibt, blieb die durchschnittliche Helligkeit des Bildes relativ stabil. Dies ermöglichte eine robuste Segmentierung des Bildes unabhängig von der aktuellen Roboterposition [Bonarini, 1999].

Alpha++

Das Alpha++-Team aus Singapur verwendete das Cognachrome Vision System von Newton Labs, das aus einer Kamera, einem Framegrabber und einem CPU-Board besteht. Auf dem CPU-Board des Bildverarbeitungssystems fand neben der Bildauswertung gleichzeitig die Roboter-Steuerung statt. Der Framegrabber ist in der Lage, drei verschiedenfarbige Objekte gleichzeitig zu lokalisieren und zu verfolgen. Als die drei Objekte wurden der orangene Ball sowie das gelbe und das blaue Tor ausgewählt. Mit- und Gegenspieler konnten somit nicht identifiziert werden. Auch eine Selbstlokalisierung war nur bedingt möglich. Hierzu wurden von Zeit zu Zeit mit Hilfe der extrahierten Torlinien der Abstand und die Richtung zum Tor berechnet und mit den Odometriedaten abgeglichen [Yong *et al.*, 1999].

CS-Sharif

Der von dem iranischen „CS-Sharif"-Team verwendete Framegrabber wandelte die von der Kamera stammenden PAL-Signale zunächst in RGB- und anschließend in HSI-Signale mit einer Auflösung von 704×510 Pixel um. Da Farben als Teilwürfel des HSI-Farbraumes darstellbar sind und jedem relevanten Objekt (Ball, Tore, Spieler) auf dem Spielfeld eine unterschiedliche

Farbe zugeordnet ist, reichte es für die Klassifizierung aus, die entsprechende Position und Größe des Würfels im HSI-Farbraum festzulegen und darauf zu achten, daß sich die jeweiligen Würfel der entsprechenden Objekte nicht überschneiden. (Bereiche des Farbraumes, die in keinem der Teilwürfel lagen, waren keinem Objekt zugeordnet.) Wurde ein Pixel klassifiziert, mußte von dessen Farbwert lediglich die Position im HSI-Farbraum und somit der ihn umgebende Teilwürfel bestimmt werden. Der Vorteil dieser Art der Farbklassifizierung bestand in dem gegenüber Farbtabellen weitaus geringerem Speicherbedarf.

Um eine Auswertung der Bilder mit einer Rate von 30 Bildern pro Sekunde zu realisieren, wurde nicht jedes der 704×510 Pixel im Kamerabild untersucht. Zunächst wurde eine Mindestpixelbreite min_{width} und Mindestpixelhöhe min_{height} für alle diejenigen Objekte im Bild definiert, die erkannt werden sollen. Dann wurde das Kamerabild in Teilbilder aufgeteilt, deren Breite und Höhe dem Wert min_{width} bzw. min_{height} entsprach.

Von jedem Teilbild wurde zunächst nur das Pixel im Zentrum des Teilbildes untersucht. Besaß ein Objekt im Kamerabild die Mindestpixelbreite und Mindestpixelhöhe so mußte das Zentrumspixel mindestens eines der Teilbilder Element des Objektes im Bild sein. Hatte eines dieser Pixel einen Farbwert, der innerhalb eines relevanten Teilwürfels des HSI-Farbraumes lag, wurden alle benachbarten Pixel, deren Farbwerte in demselben Teilwürfel lagen, zu einem Segment zusammengefaßt. Anhand des Teilwürfels wurde der Typ des Objektes identifiziert. Anschließend wurde die relative Position des Objektes zum Roboter berechnet [Jamzad, 1999].

Dreamteam

Bei dem Bildverarbeitungssystem des amerikanischen „Dreamteam" wurde das Kamerasignal ebenfalls in HSV-Signale umgewandelt. Die Auflösung betrug 658×496 Pixel. Anders als bei vielen anderen Teams wurde das Kamerabild nicht segmentiert, sondern gezielt nach einem Objekt (Ball, Tor oder Spieler) durchsucht, dessen Farbbereich im HSV-Raum vorher festgelegt wurde. Anstatt das gesamte Kamerabild nach dem entsprechenden Objekt abzusuchen, wurde das gewünschte Objekt zunächst nur in dem Bereich gesucht, in dem es sich am wahrscheinlichsten aufhielt. Dieser geschätzte Aufenthaltsort des Objektes wurde anhand der ausgewerteten Daten vergangener Kamerabilder bestimmt. War die Suche erfolglos, wurde der Such-Bereich solange ringförmig vergrößert, bis das Objekt gefunden wurde oder der Bereich das gesamte Kamerabild einnahm.

Aus Geschwindigkeitsgründen wurde auch hier nicht jedes Pixel untersucht, sondern mehrere Pixel zu „Rezeptoren“ zusammengefasst, wobei die Größe eines „Rezeptors“ abhängig von dem zu suchenden Objekt gesetzt worden war. Der Farbwert eines „Rezeptors“ wurde aber nicht über den durchschnittlichen Farbwert der darin enthaltenen Pixel bestimmt, sondern gemäß eines Mehrheitsprinzips dem „Rezeptor“ derjenige Farbwert zugeordnet, der von den Pixeln im „Rezeptor“ am häufigsten vertreten war. Die Auswertung der Kamerabilder geschah mit einer Rate von 6 Bildern pro Sekunde [Adobbati *et al.*, 1999].

RMIT

Gemäß der Philosophie des „RMIT“-Teams aus Australien stellte die genaue Selbstlokalisation des Spielers im Spielfeld keine notwendige Vorbedingung für ein effizientes Fußballspiel dar. Wie ein menschlicher Fußballspieler, der nur über ein grobes Wissen über seine Position auf dem Fußballfeld verfügt, sollten auch für die Spieler des „RMIT“-Teams ihre ungefähren Positionsdaten ausreichend sein. Das gleiche galt für die Positionsbestimmung anderer Objekte auf dem Spielfeld. Denn obwohl die Kamera des RMIT-Teams nur eine maximale Auflösung von 320×240 Pixel besaß, wurde diese Auflösung nochmals auf 160×120 Pixel in der Höhe und Breite halbiert. Die Auswertung der Kamerabilder erfolgte mit einer Rate von maximal 16 Bildern pro Sekunde.

Auch das „RMIT“-Team arbeitete aus Speicherplatzgründen nicht mit einer Farbtabelle zur Bildsegmentierung, sondern mit einem Entscheidungsbaum auf Basis des RGB-Farbraumes. Dieser Entscheidungsbaum wurde mittels eines sogenannten C4.5-Lern-Algorithmus’ erstellt [Brusey *et al.*, 1999, Padgham *et al.*, 1999].

Attempto

Das Tübinger Team „Attempto“ hatte auf jedem Spieler eine omnidirektionale und eine fixe Kamera installiert. Die fixe Kamera wurde in erster Linie für die Ballerkennung verwendet. Aus der Ballposition vorangegangener Bilddaten wurde die Position des Balles im aktuellen Bild geschätzt. Von dieser Position ausgehend wurde der Ball im Kamerabild gesucht.

Das Bildverarbeitungssystem von „Attempto“ wandelte wie das Münchner Team (siehe Abschnitt 2.4) die von der Kamera stammenden PAL-Signale in YUV-Signale um. Die Auswertung der Kamerabilder geschah mit 25 Bildern pro Sekunde. Um Bilder bei dieser Bildwiederholungsfrequenz auswerten zu können, wurde die Auflösung von 768×576 Pixel sowohl in der horizontalen wie auch in der vertikalen Richtung auf 384×288 Pixel halbiert [Plagge *et al.*, 1999].

Kapitel 3

Hardware

Dieses Kapitel skizziert die Hardwarekomponenten der Roboter des CS-Freiburg-RoboCup-Teams, die während der Turniere bei RoboCup-98 und RoboCup-99 verwendet wurden.

3.1 Allgemeines

Die hier vorgestellten Hardwareteile wurden – bis auf die Schußvorrichtung – nicht selbst entwickelt und hergestellt, sondern über verschiedene Quellen bezogen und zusammengebaut. Die benutzten Hardwarekomponenten während des RoboCup-99-Turnieres unterschieden sich kaum von der Hardware, die beim RoboCup-98 verwendet wurde. Lediglich eine neuere Version der benutzten Laserscanner sowie schnellere Notebooks kamen zum Einsatz.

Die Roboter des CS-Freiburg-RoboCup-Teams basieren auf den von *ActivMedia* vertriebenen *Pioneer 1*-Robotern. Die Roboter besitzen je zwei Vorderräder und eine Laufrolle am hinteren Teil. Die Vorderräder sind mit je einem Motor gekoppelt, der mittels Batterie betrieben wird. Die Motoren werden über einen Motorcontroller gesteuert. Weiterhin ist der Roboter standardmäßig mit sieben Ultraschallsensoren ausgerüstet, die ebenfalls mit dem Pioneer-Motorcontroller verbunden sind [ActiveMedia, 1996].

3.2 Gesamtsystem

3.2.1 Notebook

Auf jedem Roboter wurde ein Notebook Libretto 70CT bzw. Libretto 110CT der Firma Toshiba montiert.[1] Die eingesetzten Notebooks besitzen einen Intel Pentium MMX-Prozessor mit einer Taktfrequenz von 120 MHz (Libretto 70CT) bzw. 233 MHz (Libretto 110CT). Weiterhin sind sie mit 32 MB Arbeitsspeicher und einer $1,6$ GB (Libretto 70 CT) bzw. $4,0$ GB (Libretto 100CT) Festplatte ausgestattet [Toshiba, 1998]. Auf den Notebooks wurde für RoboCup-99 als Betriebssystem *RTLinux*[2] Version 2.0.36.RTL1.1 installiert. Zur direkten Roboter-Steuerung wurde bei den Turnieren '98 und '99 jeweils ein von dem CS-Freiburg-RoboCup-Team entwickeltes Saphira-Programm benutzt, welches mit Hilfe der von Kurt Konolige geschriebenen Saphira-Entwicklungsumgebung [Saphira, 1997] erstellt wurde.

3.2.2 Laserscanner

Weiterhin wurde für das RoboCup-99-Turnier auf allen Robotern ein LMS2XX-Laserscanner der Firma *Sick* montiert. Mit dessen Hilfe können Gegenstände bis zu 30 Metern Entfernung mit einer Genauigkeit von bis zu 1 cm lokalisiert werden. Dieser Scanner hat einen maximalen Winkelbereich von 180°. Die Winkelauflösung kann zwischen 0.25° und 1° variiert werden. Dank einer Übertragungsrate von 500 Kilobaud können über eine serielle Schnittstelle bis zu 37 Scans pro Sekunde an das Toshiba-Notebook übermittelt werden.[3] [SICK, 1997] Diese Scannerdaten werden zur Selbstlokalisation des Spielers,[4] sowie für die Schätzung der Position anderer Spieler benutzt.

3.2.3 Bildverarbeitungssystem

Aufgrund der Höhe des Scanners kann der Ball auf dem Spielfeld mit Hilfe des Laserscanners nicht erfaßt werden. Daher wurde speziell für die Baller-

[1] Während des RoboCup-98-Turnier waren alle Roboter mit Libretti 70 CT ausgerüstet, die bis auf den des Torwartes durch Libretti 110CT ausgewechselt wurden.

[2] RTLinux = Real-Time-Linux

[3] Bei dem während des RoboCup-98-Turnier eingesetzten Scanner war lediglich eine Übertragungsrate von 38.400 Baud möglich, mit der nur bis zu 8 Scans pro Sekunde übermittelt werden konnten.

[4] Als Spieler werden in dieser Arbeit die Fußball-Roboter bezeichnet.

Abbildung 3.1: Torwart und zwei Feldspieler des CS-Freiburg-RoboCup-Teams

kennung jeweils eine CCD-Kamera der Firma Newton Labs auf den Laserscannern befestigt, die mit dem im Kapitel 5 näher beschriebenen Framegrabber des Bildverarbeitungssystemes *Cognachrome Vision-System* verbunden ist.

3.2.4 Schußmechanismus

Der Schußmechanismus der *Pioneer 1*-Roboter wurde mit Teilen des *Märklin Metallbaukastens* vom CS-Freiburg-RoboCup-Team selbst entwickelt. Er wird mittels Magnetspulen in Gang gesetzt [Herrman, 1998].

3.2.5 Zentralrechner

Als Zentralrechner (Server) wurde u.a. ein 133 MHz -*Twinhead*-Notebook verwendet. Über ein *WaveLan* Radio-Ethernet[5] können die Spieler mit einer Datenübertragungsrate von bis zu 2 Mbit/s Informationen mit dem Zentral-

[5] Hergestellt von der Firma *Lucent Technologies*

rechner austauschen. Der Server ist in erster Linie zum Abgleich der von jedem Spieler erfaßten Ball- und Spieler-Positionsdaten (Fusion), sowie für die Übermittlung spieltaktischer Daten bestimmt.

Die Abbildung 3.1 zeigt Roboter des CS-Freiburg-RoboCup-Teams mit den in diesem Abschnitt beschriebenen Komponenten.

Kapitel 4

Ballerkennung im Überblick

Ab diesem Kapitel beginnt die Beschreibung des vom CS-Freiburg-RoboCup-Teams verwendeten Verfahrens zur Ballerkennung und zur Ballpositionsschätzung. Zum besseren Verständnis wird dieses Verfahren anhand eines Beispiels skizziert.

Das verwendete Bildverarbeitungssystem besteht aus einer Kamera, einem Visionboard und einem CPU-Board. Die von der Kamera gelieferten Signale werden zunächst auf dem Visionboard digitalisiert und entsprechend des Farbwertes segmentiert. Anschließend findet auf dem CPU-Board eine Auswahl desjenigen Objektes statt, das von all den Objekten im Bild den Ball am wahrscheinlichsten repräsentiert. Die Daten des ausgewählten Objektes werden an das Saphira-Programm auf dem Notebook gesendet, das damit die Position des Balles auf dem Spielfeld schätzt.

4.1 Farbsegmentierung auf dem Visionboard

Auf dem Visionboard werden die Kamerasignale zunächst digitalisiert und anschließend mittels Farbtabelle segmentiert.

Digitalisieren der Kamerasignale

Die von der Kamera gelieferten Signale werden auf dem Visionboard zunächst digitalisiert. Für das Beispiel wurde eine Situation nachgestellt, in der eine Person mit rotem Hemd in das Spielfeld hineingreift und eine weitere Person mit einem roten T-Shirt hinter dem gegenüberliegenden Tor an der Bande steht (siehe Abbildung 4.1). Desweiteren befinden sich zwei Bälle auf dem Spielfeld, wobei der im Kamerabild vorderste Ball dasjenige Objekt darstellt, welches bisher „verfolgt“ worden ist.

Abbildung 4.1: Digitalisiertes Bild

Farbsegmentierung

Das Kamerabild wird als nächstes mit Hilfe einer zuvor initialisierten Farbtabelle segmentiert. In dieser sind diejenigen Farbwerte besonders gekennzeichnet, die für die Objekterkennung von Interesse sind. Es wurde somit derjenige Farbbereich in der Farbtabelle abgebildet, der das Farbspektrum des Balles im Kamerabild bestmöglich wiedergibt. Bei der anschließenden Farbsegmentierung wird dann jedem Pixel, dessen Farbwert innerhalb des entsprechenden Farbspektrums liegt, der Wert „1“ und den anderen Pixeln der Wert „0“ zugeordnet. Das Ergebnis der Segmentierung ist ein Binärbild, welches verschiedene Objekte enthalten kann (siehe Abbildung 4.2). Die Aufgabe ist nun zu bestimmen, ob eines dieser Objekte den Ball repräsentiert (siehe Abschnitt 5.2).

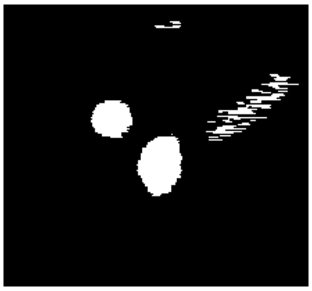

Abbildung 4.2: Segmentiertes Kamerabild

4.2 Bildauswertung auf dem CPU-Board

Auf dem CPU-Board werden die einzelnen Objekte erfaßt. Anschließend erfolgen verschiedene Plausibilitätstests. Unter den verbleibenden Objekten wird dann mittels verschiedener Kriterien ein Objekt ausgewählt, auf dem eine Mittelpunktsberechnung durchgeführt wird.

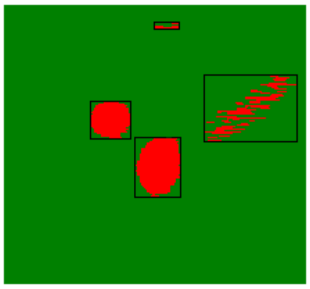

Abbildung 4.3: Verschiedene Objekte

Objektbestimmung

In diesem Schritt werden zusammenhängende Pixelflächen zu Objekten zusammengefaßt. Für jedes Objekt wird das umschreibende Rechteck berechnet. Hierunter ist das minimale Rechteck, welches das Objekt vollständig einschließt, zu verstehen (siehe Abbildung 4.3). Von den Rechtecken wird jeweils der Zentrumspunkt Z sowie die maximale und minimale Länge der Seiten bestimmt (siehe Abschnitt 6.2.1).

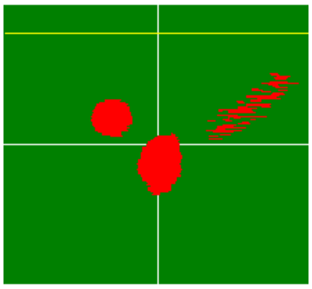

Abbildung 4.4: Obere Schranke (gelb)

Logische Filter

Als nächstes werden Objekte verworfen, die mit hoher Wahrscheinlichkeit nicht den Ball auf dem Spielfeld repräsentieren. Verworfen werden Objekte, die entweder ein zu kleines Ausmaß haben, die in ihrer Form zu schmal sind oder aber sich nur für eine kurze Zeit im Binärbild befinden. Da darüberhinaus davon ausgegangen werden kann, daß der Ball sich stets auf dem Boden des Spielfeldes bewegt, werden Objekte, die im Bild oberhalb einer gewissen Schranke (in Abbildung 4.4 gelb dargestellt) liegen, ebenfalls verworfen (siehe Abschnitt 6.2.1).

Objektauswahl

Mit Hilfe der zwei vorangehenden Kamerabilder wird der augenblickliche Aufenthaltsort geschätzt. Zu jedem der übriggebliebenen Objekten wird die Entfernung gemessen. Von den Objekten, die nicht zu weit entfernt liegen, wird die Kompaktheit berechnet. Diese ist definiert als $\frac{Umfang(Objekt)}{Fläche(Objekt)}$. Da eine Kreisscheibe die kompakteste Form darstellt, wird das Objekt mit der kleinsten Kompaktheit ausgewählt (siehe Abschnitt 6.2.2).

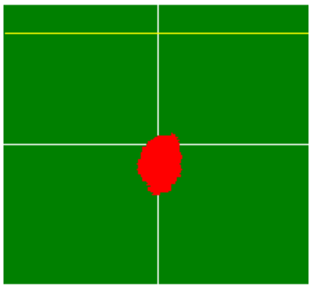

Abbildung 4.5: Ausgewähltes Objekt

Mittelpunktsschätzung

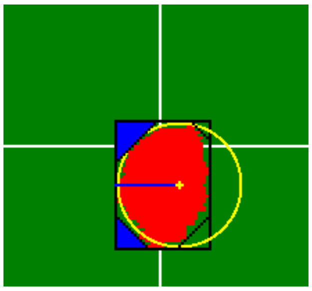

Abbildung 4.6: Geschätzter Mittelpunkt

Oftmals stellt das ausgewählte Objekt im Binärbild keine Kreisscheibe dar. In diesem Falle wird das Objekt als *Teil* einer Kreisscheibe S interpretiert. Da über den Mittelpunkt des Balles im Bild die Position des Balles auf dem Spielfeld bestimmt wird, schätzt man den Mittelpunkt von S mittels eines Dreiecksverfahrens ab. Dabei wird für jeden Eckpunkt des umschreibenden Rechteckes ein gleichschenkliges, rechtwinkliges Dreieck ermittelt, sodaß der Dreiecks-Eckpunkt, der gegenüber der Hypothenuse liegt, mit dem Eckpunkt des umschreibenden Rechteckes zusammenfällt und die anderen Eck-Punkte so auf den Rechtecksseiten zu liegen kommen, daß die Dreiecksfläche maximal ist und dabei nicht das Objekt schneidet (siehe Abbildung 4.6). Mit Hilfe der zwei größten Dreiecke (im Bild blau hervorgehoben) wird der Mittelpunkt des gesuchten Kreises (gelb) abgeschätzt. Es wird davon ausgegangen, daß der Radius des Kreises etwa das $1,7$-fache der Kathetenlänge des größten Dreieckes beträgt (siehe Abschnitt 6.2.3).

4.3 Positionsschätzung auf dem Notebook

Die vom CPU-Board bereitgestellten Konturdaten werden zunächst gemittelt. Anschließend erfolgt eine erneute Mittelpunktsberechnung mittels eines Kreisfitting-Verfahrens. Der daraus resultierende Mittelpunkt dient dann zur Positionsbestimmung des Balles auf dem Spielfeld.

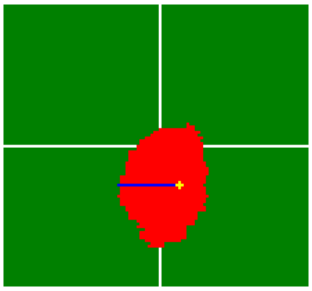

Abbildung 4.7: Gemitteltes Objekt

Objektmittelung

Über die Kontur der ausgewählten Objekte aus den vier vorausgehenden Kamerabildern wird gemittelt. Da die Mittelung nur dann durchgeführt werden soll, wenn sich der Ball auf dem Spielfeld *nicht* bewegt, werden lediglich Objekte, deren Zentrumspunkte Z nicht mehr als drei Pixel von dem Zentrumspunkt Z des aktuellen Objektes entfernt liegen, zur Mittelung einbezogen (siehe Abschnitt 7.2).

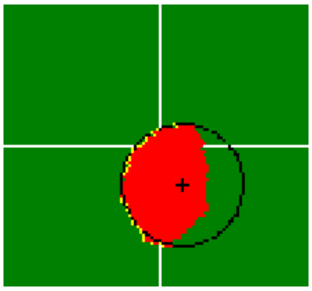

Abbildung 4.8: Kreisfittingverfahren

Das Kreisfittingverfahren

Auf der gemittelten Kontur wird das Kreisfitting-Verfahren angewendet. Dabei wird die Kontur in vertikaler und horizontaler Ebene längs des geschätzten Mittelpunktes geteilt. Für jedes der so entstehenden vier Kontursegmente wird derjenige Kreis gesucht, bei dem die Summe der Abstände zu ausgewählten Punkten des Kontursegmentes minimal ist. Diejenigen Kontursegmente, für die der Kreisfitting-Algorithmus eine kleine Abstandssumme liefert, werden zusammengefasst. Anschließend erfolgt eine erneute Anwendung des Kreisfitting-Algorithmus, um den für die eigentliche Positionsschätzung benötigten Mittelpunkt und Radius zu ermitteln (siehe Abschnitt 7.3).

Plausibilitätstest

Sind Radius und Mittelpunkt bestimmt, wird anhand einer zuvor generierten Tabelle überprüft, ob der Radius mit dem bei einem Kalibrierungsvorgang ermittelten Radius bezüglich der Mittelpunkt-Position übereinstimmend ist.

Bei dem Kalibrierungsvorgang werden zu verschiedenen Ballpositionen die zugehörigen Mittelpunkte und Radien des Balles im Bild errechnet. Anhand der Kalibrierungsdaten wird eine Positionstabelle generiert, die über die Punkte des Kamerabildes indiziert ist. Zu jedem Bildpunkt wird der bei der Kalibrierung ermittelte Radius und die entsprechende Ballposition auf dem Spielfeld gespeichert (siehe Abschnitt 7.4.2).

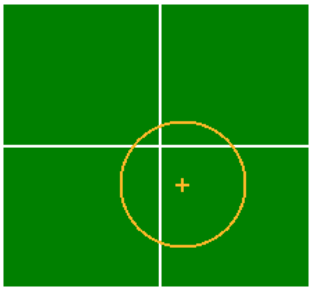

Abbildung 4.9: Mittelpunkt und Radius

Positionsbestimmung

Ist der ermittelte Radius bezüglich des zu erwartenden Radius passend, wird die Position des Balles im Spielfeld anhand der Positionstabelle bestimmt (siehe Abbildung 4.10). Mit Hilfe der Ballposition des letzten Kamerabildes wird die Geschwindigkeit in Richtung des Balles geschätzt.

Die ermittelte Ballposition eines Spielers wird abschließend mit den ermittelten Ballpositionen anderer Spieler abgeglichen. Dies geschieht mit Hilfe eines Zentralrechners, der mit den Spielern kommuniziert. Der Zentralrechner mittelt dabei die Balldaten und stellt die gemittelten Ballpositionen den einzelnen Spieler wieder zur Verfügung (siehe Abschnitt 7.4.3).

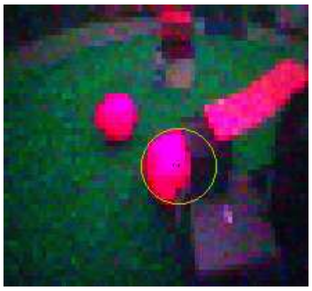

Abbildung 4.10: Ermittelte Ballposition

Kapitel 5

Das Bildverarbeitungssystem

Dieses Kapitel beschäftigt sich mit dem verwendeten Bildverarbeitungssystem. Im ersten Abschnitt wird die Struktur des Bildverarbeitungssystems beschrieben, das aus einer Kamera, einem Visionboard und einem CPU-Board besteht.

Der erste Abschnitt beschäftigt sich mit der Digitalisierung der Kamerasignale und der Farbsegmentierung auf dem Visionboard. Dabei wird beschrieben, wie die zur Farbsegmentierung benötigten Farbtabellen aufgebaut sind und welche Einschränkungen bezüglich der Objekterkennung aufgrund des Designs der Farbtabelle gegeben sind. Außerdem wird die Art und Weise besprochen, wie die Farbtabellen initialisiert werden.

Der letzte Abschnitt befaßt sich mit dem Aufbau des CPU-Boards, auf das speziell angepaßte Programme zur Bildauswertung heruntergeladen werden können.

5.1 Das Cognachrome Vision-System

Das vom CS-Freiburg-RoboCup-Team eingesetzte Bildverarbeitungssystem ist das *Cognachrome 2000 Color Vision System* der Firma *Newton Research Labs*. Es besteht aus einer CCD-Kamera mit Fischaugenobjektiv[1], einem Visionboard zur Digitalisierung und anschließender Farbsegmentierung der analogen Video-Signale der Kamera sowie einem CPU-Board, mit dessen Hilfe eine (Teil-)Auswertung der Kamerabilder erfolgen kann. Die

[1] Das Fischaugenobjektiv verzerrt zwar das Kamerabild, ermöglicht aber ein Sichtfeld von ca. 90°.

CCD-Kamera sendet ihre Signale im amerikanischen NTSC[2]-Format. Die beiden Boards sind übereinandergesteckt und haben zusammen die Maße $2,5'' \times 6,25'' \times 1,25''$. Über eine serielle Schnittstelle ist das Bildverarbeitungssystem sowohl mit dem Notebook als auch mit dem Pioneer-Motorcontroller verbunden (siehe Abbildung 5.1).

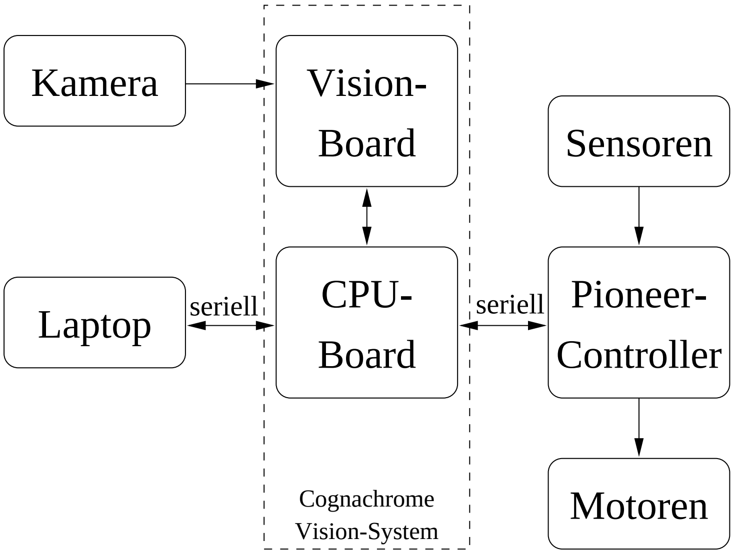

Abbildung 5.1: Architektur mit Cognachrome-Bildverarbeitungssystem

5.2 Das Vision-Board

Das Visionboard digitalisiert das von der Kamera gesendete analoge NTSC-Signal in 24 Bit RGB-Signale, bei der jede Farbe mit acht Bit kodiert wird. Die Digitalisierung der NTSC-Signale erfolgt mit einer Auflösung von ca. 223×254 Pixeln und einer Abtastrate von 60 Hz. Das daraus entstehende Kamerabild wird anschließend farbsegmentiert.

Zur Farbsegmentierung existieren drei Farbkanäle mit je einer Farbtabelle, mittels derer die RGB-Signale in Binärsignale umgewandelt werden. So können bis zu drei verschiedenfarbige Objekte gleichzeitig erkannt bzw. verfolgt werden. Welche Objekte aufgrund ihrer Farbe erkannt werden sollen,

[2]NTSC = National Television Systems Committee

kann durch eine farbspezifische Zuordnung der jeweiligen Farbtabellenfelder bestimmt werden.

Die Farbsegmentierung ist auf dem Visionboard hardwaremäßig implementiert worden. Dadurch konnte eine Segmentierung des Kamerabildes mit einer Frequenz von 60 Hz realisiert werden. Möchte man darüberhinaus das Kamerabild mittels eines Programms noch zusätzlich analysieren, ist dies nur eingeschränkt möglich. Anstatt in der Auflösung von 223×254 Pixeln wird einem Softwareprogramm das Kamerabild lediglich in der Auflösung von 64×48 Bildpunkten zur Verfügung gestellt. Der bei den RoboCup-Turnieren verwendete Ball ist z.B. in einer Entfernung von 3,5 Metern in solch einem Bild nur noch ca. drei Pixel breit. Eine robuste Ballerkennung ist in diesem Falle nicht mehr durchführbar.

5.2.1 Die Farbtabellen des Vision-Boards

Um Objekte wie einen Ball mit Hilfe des Bildverarbeitungssystems erkennen und verfolgen zu können, existieren wie oben beschrieben auf dem Visionboard drei Farbtabellen zur Farbsegmentierung. Da das CS-Freiburg-RoboCup-Team das Bildverarbeitungssystem lediglich zur Ballerkennung einsetzt, wird nur eine der Farbtabellen verwendet. Mit Hilfe dieser Farbtabelle wird jedes Pixel des Kamerabildes gemäß seiner Farbe klassifiziert. Dabei werden die Bildpunkte unterteilt in solche Punkte, die Teil des zu verfolgenden Objektes im Bild sind, und solche Punkte, die aller Wahrscheinlichkeit nach nicht Teil des Objektes sind. Die Farbtabelle ist über die RGB-Werte indiziert. Für jeden RGB-Wert liefert die Tabelle ein einzelnes Bit: „1“ wenn das jeweilige Pixel interessant, und „0“ falls es uninteressant ist. Das so entstehende Schwarz-Weiß-Bild wird im folgenden als Binärbild bezeichnet.

Da ein 24-Bit-RGB-Wert $2^{24} = 16.777.216$ verschiedene Farbwerte annehmen kann, müßte die Farbtabelle 16 Megabit groß sein, weil sie für jeden möglichen Farbwert ein Bit speichern muß. Da das Cognachrome-Vision-System die Möglichkeit bietet, drei verschiedenfarbige Objekte gleichzeitig zu verfolgen, würden die drei entsprechenden Farbtabellen zusammen einen Speicherplatz von 6 Megabyte benötigen. Zum Zeitpunkt des Entwurfs des Visionboards war eine Realisierung mit einer solchen Speicherplatzgröße nicht durchführbar. Daher beschlossen die Entwickler des Visionboards die Farbtabellen in zwei Untertabellen zu splitten. Dabei wird das RGB-Signal in ein 16 bit RG[3]-Signal und in ein 16 bit BG[4]-Signal unterteilt. Die erste Untertabelle ist dann über alle möglichen RG-Werte und die zweite Tabelle über alle möglichen BG-Werte indiziert. Damit ein RGB-Pixel im Binärbild die Zahl „1“ zugewiesen bekommt, müssen die Untertabellen sowohl für den RG-Wert wie auch für den BG-Wert die Zahl „1“ liefern. Durch diese Vorgehensweise sind für die dreimal 2 Untertabellen nur noch $3 * 2 * 2^{16}$ gleich 393.216 Bit, bzw. 48 Kilobyte nötig.

5.2.2 Einschränkung durch das Farbtabellendesign

Die Reduzierung der Farbtabellengröße führt jedoch zu einem unerwünschten Nebeneffekt. Durch sie wird die Wahrscheinlichkeit erhöht, daß manche Bildpunkte als „falsch positiv“ erkannt werden und damit erhebliche Probleme bei der Bildauswertung zur Folge haben können.

Betrachten wir dazu ein Beispiel und nehmen an, daß das Objekt, das erkannt werden soll, hellgrün sei und den RGB-Wert „80FF80h“[5] besitzt. Nehmen wir weiterhin den durchaus realistischen Fall an, daß aufgrund von Beleuchtungsverhältnissen das Objekt im Kamerabild an manchen Stellen heller und an anderen Stellen dunkler erscheint, also an manchen Stellen den RGB-Wert „48FF48h“ (Lindgrün) und an anderen Stellen den RGB-Wert „B8FFB8h“ (Seegrün) annimmt. Soll das ganze Objekt im Binärbild abgebildet werden, so muß u.a. in der Farbtabelle sowohl für die RG-Werte „B8FFh“ (Seegrün) und „48FFh“ (Lindgrün) wie auch für die BG-Werte „B8FFh“ (Seegrün) und „48FFh“ (Lindgrün) eine „1“ eingetragen werden. Das Problem ist nun, daß mit den Farben Seegrün und Lindgrün auch die Farben Türkis mit dem RGB-Wert „48FFB8h“ und Olivgrün mit dem RGB-Wert „B8FF48h“ als interessant klassiziert werden, da ihre RG- und BG-Werte ebenfalls „B8FFh“ bzw. „48FFh“ betragen (siehe Abbildung 5.2).

Das bedeutet, daß das Cognachrome-Vision-System in diesem Fall je nach Lichtverhältnissen ein hellgrünes Objekt nicht von einem türkisen Hintergrund unterscheiden kann. Für eine zuverlässige Objektverfolgung sollten daher die Farbwerte des gesamten Farbraumes des zu verfolgenden Objektes sich mit den Farbwerten des Farbraumes des Hintergrundes oder anderer Objekte nicht gleichzeitig in der RG- und BG-Ebene überschneiden. In der Praxis ist dies jedoch nur sehr schwer realisierbar.

[3]RG = Red Green
[4]BG = Blue Green
[5]Hexadezimalzahlen werden in dieser Arbeit dadurch gekennzeichnet, daß unmittelbar hinter der Zahl der Buchstabe „h“ folgt.

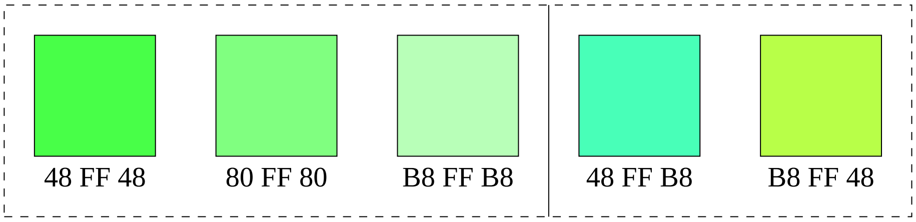

Abbildung 5.2: Verschiedene Grünwerte mit ihren RGB-Werten in hexadezimaler Schreibweise

5.2.3 Initialisieren der Farbtabelle

Je nachdem, welches Objekt erkannt werden soll, können die Farbtabellen entsprechend dem Farbspektrum des Objektes im Kamerabild belegt werden. Dies geschieht durch das sogenannte *Trainieren* des Cognachrome-Vision-Systems. Dabei wird das entsprechende Objekt, wie beispielsweise der Ball, in die Mitte des Sichtfeldes der Kamera gelegt. Eine durch ein Programm aufgerufene Bibliotheksfunktion initialisiert zunächst alle Felder der entsprechenden Tabelle mit dem Wert „0“ . Dann wird der Farbwert des Pixels im Zentrum des aktuellen Kamerabildes ausgelesen. Von diesem Bildpunkt ausgehend wird versucht, den Umriß des Objektes im Bild zu finden. Dabei werden alle umliegenden Pixel im Bild untersucht, wobei jeweils die Distanz des Farbwertes des Pixels im RGB-Raum zu dem Farbwert des Zentrumspixels berechnet wird. Übersteigt diese Distanz einen bestimmten parametrisierbaren Schwellenwert, so wird das entsprechende Pixel nicht mehr als zu dem Objekt gehörig klassifiziert. Im anderen Fall wird es als Teil des Objektes definiert. Um das Zentrumspixel herum entsteht so eine mehr oder weniger große zusammenhängende Fläche von solchen Pixeln, die als Teil des Objektes bestimmt wurden. Diese Fläche ist im Normalfall nicht „glatt“, sondern besitzt einige „Löcher“. Der Umriß dieser Fläche bestimmt dann die Kontur des Objektes.

Steht die Kontur des Objekts fest, wird für die Farbwerte von all den Pixeln, die innerhalb der Kontur liegen, in der entsprechenden Tabelle, bzw. in den entsprechenden zweidimensionalen Untertabellen, der Wert „1“ eingetragen. Die dadurch entstandene Belegung wird im Normalfall in den Tabellen keine zusammenhängende und glatte Struktur von denjenigen Feldern ergeben, die als „interessant“ markiert worden sind. Um eine solche Struktur zu erhalten, wird für jede Untertabelle die minimale zusammenhängende „Fläche“, d.h.

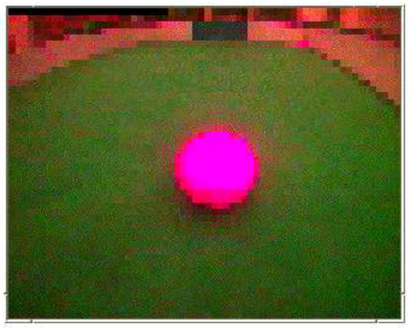

Abbildung 5.3: Kamerabild mit einer Auflösung von 64 × 48 Pixeln

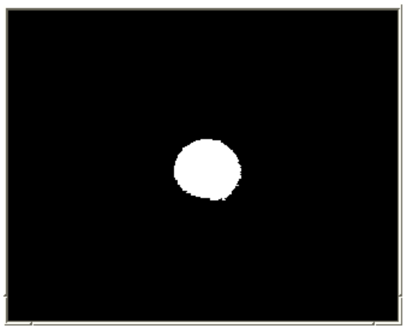

Abbildung 5.4: das entsprechende Binärbild

die konvexe Hülle in der jeweiligen Tabellenebene bestimmt, die alle als „interessant" markierten Felder einschließt. Anschließend wird jedes Feld der Tabelle, welches innerhalb dieser „Fläche" liegt, mit dem Wert „1" belegt.

Es ist leicht nachzuvollziehen, daß eine exakte Wiedergabe des tatsächlichen Farbspektrums des Objektes im Bild durch diese Art des „Trainierens" kaum möglich ist (vergleiche Abbildungen 5.3 und 5.4). Selbst die Möglichkeit der interaktiven Anpassung des obengenannten Schwellenwertes führt meist nicht zu dem gewünschten Ergebnis.

5.2.4 Glanzlichter, Schatten und Geisterbälle

Bei den RoboCup-Spielen sind die Beleuchtungsverhältnisse derart, daß im oberen Drittel des orangenen Balles Glanzlichter[6] auftreten, die den Ball an diesen Stellen im Kamerabild nahezu weiß erscheinen lassen. Im unteren Drittel des Balles dagegen erscheint der Ball aufgrund von Schatten im Kamerabild an manchen Stellen nahezu schwarz. Erschwerend kommt hinzu, daß auf dem Ball ein schwarzer Schriftzug aufgedruckt ist (siehe Abbildung 5.5). Je mehr Farbwerte von dem möglichen Farbspektrum des Balles im Bild durch das Trainieren in der Farbtabelle als „interessant" markiert werden, desto besser ist ein entfernt liegender Ball im Binärbild identifizierbar. Das Problem hierbei ist jedoch, daß aufgrund der teilweise sehr intensiven Beleuchtung auch auf anderen (andersfarbigen) Objekten im Spielfeld

[6]Die glänzendreflektierende Oberfläche des Balles spiegelt die Lichtquelle wieder. Das Glanzlicht besitzt somit die Farbe dieser Lichtquelle (weiß).

Abbildung 5.5: Glanzlichter und Schatten des Balles im Kamerabild

Glanzlichter und Schatten auftreten, deren Farbwerte im Kamerabild sich dann mit den Farbwerten des Balles im Bild überschneiden. D.h. je mehr Farbwerte von dem Farbspektrum des Balles als „interessant“ markiert werden, desto größer ist die Wahrscheinlichkeit, daß andere Objekte als der Ball im Binärbild wiedergegeben und fälschlicherweise als Ball erkannt werden. Dieses Problem wird noch verstärkt, da auch die beschriebene Reduzierung der Farbtabellen (siehe Abschnitt 5.2.2) zu „falsch positiv“ erkannten Bildpunkte führen kann. Das „Trainieren“ stellt somit eine schwierige Abwägung dar zwischen der qualitativen Repräsentation des Balles im Bild und der Häufigkeit des Auftretens von Objekten im Binärbild, die nicht den Ball repräsentieren. Diese letztgenannten Objekte werden in dieser Arbeit als *Geisterbälle* bezeichnet.

5.3 Das CPU-Board

Das Cognachrome-Vision-System enthält neben dem Visionboard noch eine zweite Platine: ein CPU-Board. Auf dieser Platine befindet sich ein kompletter Rechner mit einem 16 MHz-*Motorola- 68332*-Prozessor. Als Arbeitsspeicher stehen 256 Kilobyte RAM[7] und 256 Kilobyte EEPROM[8] zur Verfügung. Auf dem EEPROM befindet sich ein „Loader“ zum Laden des Betriebs-

[7]RAM = Random Access Memory

[8]EEPROM = Electrically Erasable Programmable Read Only Memory

systemkernels in den Arbeitsspeicher sowie das von der Firma *Newton Lab* geschriebene Betriebssystem. Auf dem CPU-Board wurde während des RoboCup-Turnieres '98 das von *Newton Lab* mitgelieferte *PioneerVision-98*-Programm[9] und während des Turniers '99 das selbst entwickelte *PioneerVision-99*-Programm verwendet.

Über zwei serielle RS-232-Schnittstellen ist das CPU-Board einerseits mit einem Laptop und andererseits mit dem Controller für die Roboter-Motorsteuerung (Pioneer-Controller) verbunden (siehe Abbildung 5.1). Mit einer Übertragungsrate von bis zu 38400 Baud können über diese Schnittstellen Datenpakete gesendet werden, die ausgewertete Bildinformationen oder Motorsteuerungsbefehle enthalten.

Da das CPU-Board mit einem relativ langsamen Prozessor ausgestattet ist und überdies keine Gleitkommaarithmetik unterstützt, können in dem PioneerVision-Programm keine rechenintensiven Algorithmen implementiert werden. Letztere werden deshalb auf dem Notebook des Roboters ausgeführt (wie z.B. der in einem späteren Kapitel beschriebene Kreisfitting-Algorithmus).

[9] Die exakte Bezeichnung des Programmes lautete „pioneer-27_0". Zur besseren Übersicht jedoch werden in dieser Arbeit die 1998 und 1999 auf dem CPU-Board eingesetzten Programme als *PioneerVision-98*-Programm bzw. als *PioneerVision-99*-Programm bezeichnet.

Kapitel 6

Das PioneerVision-Programm

Dieses Kapitel beschreibt das PioneerVision-Programm, das bei RoboCup-99 auf dem CPU-Board des Cognachrome-Vision-System gelaufen ist. Während beim RoboCup-Turnier 1998 das mit dem Bildverarbeitungssystem mitgelieferte *PioneerVision-98*-Programm unverändert eingesetzt wurde, ist 1999 das speziell für die Zwecke des CS-Freiburg-RoboCup-Teams konzipierte *PioneerVision-99*-Programm verwendet worden.

Der erste Abschnitt befaßt sich mit dem Aufbau des *PioneerVision-99*-Programmes. Dabei werden die verschiedenen Aufgabenbereiche kurz skizziert.

Der nächste Abschnitt behandelt die Auswertung der vom Visionboard bereitgestellten Bilddaten. Neben der Beschreibung verschiedener effizienter Plausibilitätstests wird auch das zur Mittelpunktsberechnung verwendete Dreiecks-Verfahren behandelt.

Das Senden der Vision-Informationspakete ist Gegenstand des letzten Abschnittes.

6.1 Die Programm-Struktur

Auf dem CPU-Board des Cognachrome-Vision-System sind folgende Programme installiert:

- Ein Bootstrap zum Laden (Booten) des Betriebssystems.
- Das Betriebssystem (Kernel) des Cognachrome-Vision-Systems inklusive eines speziellen Fehlersuch- (Debug-) Programms sowie einer Bibliothek (Library) mit Funktionen, die von Benutzerprogrammen verwendet werden können.
- Das PioneerVision-Programm zur Kommunikation mit dem Saphira-Programm auf dem Notebook einerseits und dem Motorcontroller des Roboters andererseits.

Das PioneerVision-Programm gliedert sich in fünf verschiedene Aufgabenbereiche. Diese lassen sich folgendermaßen beschreiben:

1. Werte die vom Visionboard erzeugten Binärbilder aus und erzeuge jeweils ein Vision-Informationspaket.
2. Sende das jeweils aktuelle Vision-Informationspaket an das Notebook.
3. Empfange Vision-Steuerungspakete vom Notebook und werte diese aus.
4. Leite Motor-Informationspakete des Motorcontrollers an das Notebook weiter.
5. Leite Motor-Steuerungspakete des Notebooks an den Motorcontroller weiter.

6.2 Auswertung der Binärbilder

Mit einer Frequenz von bis zu 60 Hz werden die Binärbilder vom Visionboard zur Verfügung gestellt. Die Binärbilder enthalten Pixel entweder mit dem Wert „0“ oder dem Wert „1“ (siehe Abschnitt 5.2.1). Eine zusammenhängende Fläche von Pixeln mit dem Wert „1“ wird als Blob bezeichnet. Im günstigen Falle repräsentiert der größte Blob das zu „verfolgende“ Objekt im Binärbild.

Das an das CPU-Board übermittelte Binärbild kann unter Umständen mehrere Blobs enthalten. Der Grund können die im Abschnitt 5.2.4 beschriebenen *Geisterbälle* sein. Diese können z.B. durch Kleider, Arme, Beine oder Hände von Zuschauern, die sich am Rande des Spielfeldes befinden, im Binärbild erzeugt worden sein. Um herausfinden zu können, ob ein Blob im Binärbild den Ball auf dem Spielfeld repräsentiert, wrude für den RoboCup '99, zusätzlich zur Farberkennung eine Konturenerkennung implementiert. Hierzu mußte das *PioneerVision-98*-Programm neu konzipiert und um einige Punkte erweitert werden.

Die Bildauswertung des *PioneerVision-99*-Programms umfaßt im wesentlichen drei Schritte. Diese sind

1. Blob-Validierung
2. Blob-Auswahl
3. Mittelpunktschätzung

6.2.1 Blob-Validierung

In diesem Schritt werden alle Blobs herausgefiltert, die mit hoher Wahrscheinlichkeit nicht den Ball auf dem Spielfeld repräsentieren, sondern *Geisterbälle* im Binärbild darstellen. Von jedem Blob wird zunächst das umschreibende Rechteck berechnet.[1] Die minimale der beiden Seitenlängen dieses Rechteckes wird in diesem Kapitel mit min_{length} und die maximale entsprechend mit max_{length} bezeichnet. Das Zentrum des Rechteckes nennen wir Z.

Geisterbälle können u.a. aufgrund ihrer Größe, ihrer Form, ihrer Position oder über ihre Verweildauer im Binärbild erkannt werden. Als *Geisterbälle* verworfen werden Blobs,

1. bei denen eine der Seiten des umschreibende Rechteck zu klein ist,
2. die in ihrer Form zu schmal sind,
3. deren oberster Punkt eine vordefinierte Schranke im Binärbild überschreitet,
4. die im Kamerabild nur kurz auftreten.

[1] Durch das Cognachrome-Vision-System bedingt sind alle vom Visionboard gelieferten Binärbilder in der Horizontalen im Verhältnis zur Vertikalen um etwa 1/4 gestaucht. Alle nun folgenden Angaben und Berechnungen beziehen sich auf das ungestauchte Bild.

Blob-Mindestgröße

Aufgrund von Störungen im Binärbild war es angebracht, für Blobs eine Mindestgröße $min_{blobsize}$ festzulegen. Ein Blob, dessen Rechteckseitenlänge min_{length} kleiner als $min_{blobsize}$ Pixel ist, wird somit als *Geisterball* interpretiert und verworfen. Bei der Bildauswertung ist es erfahrungsmäßig von Vorteil, eher *richtige* Blobs fälschlicherweise als *Geisterbälle* zu verwerfen, als *falsche* Blobs irrtümlich als Ball zu interpretieren.

Der Parameter $min_{blobsize}$ ist einstellbar und wurde während der RoboCup-Spiele auf den Wert 5 gesetzt. Ein Ball, der im Binärbild durch einen Blob mit der Höhe von 5 Pixeln repräsentiert wird, ist dabei ca. $4-5$ Meter vom Roboter entfernt.

Seitenverhältnis des umschreibenden Rechteckes

Ebenso werden diejenigen Blobs als *Geisterbälle* betrachet, die im Verhältnis zu lang oder zu breit sind. Da eine sinnvolle Konturenauswertung eines Blobs, der einen zu mehr als 2/3 verdeckten Ball repräsentiert, bei der vorhandenen Auflösung des Kamerabildes nicht durchführbar ist, werden alle Blobs verworfen, deren Rechteckseitenlänge max_{length} mehr als dreimal so lang ist wie die Rechteckseitenlänge min_{length}.

Obere Schranke im Binärbild

Betrachtet man den Ball im Kamerabild bei verschiedenen Positionen auf dem Spielfeld, so fällt auf, daß es einen oberen Abschnitt im Kamerabild gibt, in dem sich der Ball nie befindet. Die unterste Kante dieses Abschnittes verwenden wir als obere Schranke. Da davon ausgegangen werden kann, daß sich der Ball während des Spieles normalerweise auf dem Boden befindet, kann keiner der Blobs, die auch nur teilweise oberhalb dieser Schranke liegen, den Ball auf dem Spielfeld repräsentieren. Deshalb werden diese Art von Blobs ebenfalls verworfen.[2]

Mindestlänge einer Blobsequenz

Wie bereits im Kapitel 5 beschrieben, ist die Anzahl und die Art des Erscheinens von *Geisterbällen* im Binärbild insbesondere von den jeweiligen

[2]Die Blobs, die im Binärbild oberhalb der oben genannten Schranke liegen, werden durch Kleidungsstücke oder Hände von Zuschauern am Rande des Spielfeldes erzeugt.

Beleuchtungsverhältnissen abhängig. Da das Licht nicht gleichmäßig von den Objekten reflektiert wird, tritt das Phänomen auf, daß die *Geisterbälle* oftmals nur sehr kurz im Binärbild erscheinen, bzw. im Binärbild hin- und herspringen oder sprunghaft ihre Größe ändern. Insbesondere dann, wenn der Spieler oder die betreffenden Objekte sich bewegen.

Um *Geisterbälle* dieser Art zu erkennen, werden Blobs aus aufeinanderfolgenden Binärbildern analysiert. Dabei werden nur diejenigen Blobs betrachtet, die in den $min_{sequencelen}$ vorangegangenen Binärbildern als Ball oder als *möglicher Ball* (siehe weiter unten) interpretiert wurden. Die Zahl $min_{sequencelen}$ ist ein einstellbarer Parameter.

Existiert nicht in jedem der $min_{sequencelen}$ vorangegangenen Binärbilder ein den (möglichen) Ball repräsentierender Blob, werden alle verbleibenden Blobs des aktuellen Binärbildes zunächst als *Geisterbälle* markiert.

Existieren dagegen $min_{sequencelen}$ solcher Blobs, wird der Aufenthaltsort und die Größe des Balles im aktuellen Binärbild anhand linearer Extrapolation geschätzt. Diese wird mit Hilfe der umschreibenden Rechtecke der zwei vorangehenden als Ball oder als *möglicher Ball* interpretierten Blobs, die im folgenden als $blob_1$ und $blob_2$ bezeichnet werden, durchgeführt.

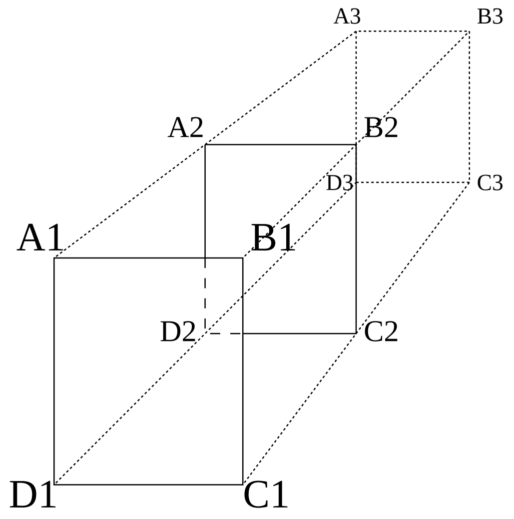

Abbildung 6.1: Die umschreibenden Rechtecke $\vec{A}_1\vec{B}_1\vec{C}_1\vec{D}_1$, $\vec{A}_2\vec{B}_2\vec{C}_2\vec{D}_2$ und $\vec{A}_3\vec{B}_3\vec{C}_3\vec{D}_3$

Sei $blob_1$ der zeitlich vorangehende Blob zu $blob_2$. Seien weiterhin

$\vec{A}_1, \vec{B}_1, \vec{C}_1, \vec{D}_1$ die Punkte des umschreibenden Rechteckes von $blob_1$ und $\vec{A}_2, \vec{B}_2, \vec{C}_2, \vec{D}_2$ die Punkte des umschreibenden Rechteckes von $blob_2$. Die geschätzten Werte für die Punkte $\vec{A}_3, \vec{B}_3, \vec{C}_3, \vec{D}_3$ des umschreibenden Rechteckes des Blobs, das den Ball im aktuellen Binärbild repräsentiert, ergeben sich aus folgender Berechnung (siehe auch Abbildung 6.1):

$$\begin{aligned} \vec{A}_3 &= \vec{A}_1 + 2(\vec{A}_2 - \vec{A}_1); \vec{B}_3 = \vec{B}_1 + 2(\vec{B}_2 - \vec{B}_1); \\ \vec{C}_3 &= \vec{C}_1 + 2(\vec{C}_2 - \vec{C}_1); \vec{D}_3 = \vec{D}_1 + 2(\vec{D}_2 - \vec{D}_1); \end{aligned} \tag{6.1}$$

Sei im folgenden A die konvexe Hülle der Punkte $\vec{A}_1, \vec{B}_1, \vec{C}_1, \vec{D}_1, \vec{A}_3, \vec{B}_3, \vec{C}_3, \vec{D}_3$. Existiert mindestens ein Blob im Binärbild, der einen Schnittbereich mit der Fläche A besitzt, werden alle Blobs die keinen Schnittbereich mit A besitzen als *Geisterbälle* interpretiert und verworfen. Im anderen Falle werden die bisher als gültig interpretierten Blobs des aktuellen Bildes zunächst nur als *Geisterbälle markiert.*

6.2.2 Blob-Auswahl

Im nächsten Schritt wird von den verbleibenden Blobs derjenige Blob ausgewählt, der am *kompaktesten* ist. Die Kompaktheit K ist hierbei folgendermaßen definiert, wobei max_{length} die größere der beiden Seitenlängen des umschreibenden Rechteckes bezeichnet:

$$K(Blob) = \frac{Umfang(Blob) \times \frac{max_{length}}{4}}{Fläche(Blob)} \tag{6.2}$$

Je kleiner K ist, desto *kompakter* ist der entsprechende Blob. Stellt der Blob eine ideale Kreisfläche dar, so ist dessen Kompaktheit gleich $\frac{2r\pi \times \frac{r}{2}}{r^2\pi} = 1$, mit $r = \frac{max_{length}}{2}$. Die Kreisfläche ist das *kompakteste* Gebilde im zweidimensionalen Raum. Da der Ball im Binärbild im Idealfall eine Kreisfläche darstellt, wird der Ball am wahrscheinlichsten durch den Blob mit der kleinsten Kompaktheit repräsentiert, und somit ausgewählt. Die anderen Blobs werden verworfen.

Wurden die Blobs des aktuellen Binärbildes bei der Validierung zuvor als *Geisterbälle* markiert (siehe oben), so wird der ausgewählte Blob als *möglicher Ball* markiert. Für die weiteren Schritte der Ballerkennung wird dieser Blob dann nicht weiter verwendet. D.h. es wird so verfahren, als ob im aktuellen Binärbild kein Ball erkannt worden ist. Dieser Blob spielt dann nur noch für die Blob-Validierung der nächsten $min_{sequencelen}$ Binärbilder eine Rolle.[3]

Eine Schwierigkeit bezüglich der als *möglicher Ball* markierten Blobs ist nun folgende: Wenn der Ball in das Sichtfeld der Kamera kommt, so werden die ausgewählten Blobs der ersten $min_{sequencelen}$ Binärbilder als *möglicher Ball* markiert. Das bedeutet, daß der Ball mindestens für die Dauer von $min_{sequencelen} + 1$ bearbeiteten Binärbildern im Sichtfeld der Kamera sein muß, um von dem Bildverarbeitungssystem erkannt zu werden. Die Wahl von $min_{sequencelen}$ stellt somit eine schwierige Abwägung dar zwischen dem qualitativen Erkennen von im Binärbild kurz auftretenden *Geisterbällen* und der möglichen Reaktionszeit auf Bälle, die in das Sichtfeld der Kamera kommen. Insbesondere für die Situation, bei der sich der Spieler um die eigene Achse dreht, weil er den Ball sucht, gibt es keine einfache Lösung. Ist $min_{sequencelen}$ zu groß, kann es sein, daß er den Ball überhaupt nicht sieht, weil der Ball entweder nach $min_{sequencelen}$ Binärbildern bereits wieder aus dem Sichtfeld verschwunden ist oder aber – wegen der Entfernung des Balles vom Roboter sowie wegen der Drehbewegung – der Ball nicht konstant für die Dauer von $min_{sequencelen} + 1$ Binärbildern in den Binärbildern erscheint. Ist $min_{sequencelen}$ zu klein, so wird der Ball zwar relativ schnell erkannt, jedoch steigt die Wahrscheinlichkeit, daß *Geisterbälle* als Ball interpretiert werden.

Während der RoboCup-Spiele wurde der Wert für $min_{sequencelen}$ auf 4 gesetzt.

6.2.3 Das Dreiecksverfahren

Mittels der Position des Balles im Binärbild wird die relative Position des Balles zum Spieler bestimmt. Dazu muß ein Referenzpunkt im Bild existieren, mit Hilfe dessen die Position des Balles auf dem Spielfeld ermittelt werden kann. In der 98-er Version des PioneerVision-Programmes wird als Referenzpunkt der Schwerpunkt des Blobs gewählt. Solange der Ball im Binärbild als (ideale) Kreisscheibe abgebildet wird, ist die Abbildung vom

[3] Anmerkung zur Validierung mittels Extrapolation der umschreibenden Rechtecke: Bei einem ruhenden Ball sind im günstigen Falle die umschreibenden Rechtecke $\vec{A}_1\vec{B}_1\vec{C}_1\vec{D}_1$ und $\vec{A}_2\vec{B}_2\vec{C}_2\vec{D}_2$ identisch. Somit sind diese auch identisch mit dem geschätzten umschreibende Rechteck $\vec{A}_3\vec{B}_3\vec{C}_3\vec{D}_3$. Bekommt ein Ball plötzlich einen so starken Impuls, daß der ihn repräsentierende Blob keinen gemeinsamen Schnittbereich mit dem geschätzten umschreibende Rechteck hat, wird der Ball im aktuellen Kamerabild als *Geisterball* markiert und nur als *möglicher Ball* interpretiert. Beim nächsten ausgewerteten Kamerabild jedoch wird der entsprechende Blob mit großer Wahrscheinlichkeit wieder als Ball erkannt.

Binärbild auf die reale Welt eindeutig. Eine relativ zum Roboter bestimmte Ballposition auf dem Spielfeld wird im Binärbild stets den gleichen Schwerpunkt liefern. Anders sieht es aus, wenn ein ruhender Ball aus der Sicht der Kamera nach und nach verdeckt wird. Dann verschiebt sich der Schwerpunkt des Blobs im Binärbild, weil der Ball nicht mehr als vollständige Kreisscheibe, sondern nur noch als *Teil* einer Kreisscheibe abgebildet wird. Aber auch bestimmte Lichtverhältnisse oder Schatten führen zu solch einer deformierten Abbildung des Balles im Binärbild. Der Schwerpunkt als Referenzpunkt ist in solchen Fällen nicht mehr korrekt.

Daher wurde im *PioneerVision-99*-Programm ein Verfahren implementiert, das zu einem gegebenen Blob, der als Teil einer Kreisscheibe interpretiert wird, den Mittelpunkt $\tilde{M}$ des entsprechenden Kreises errechnet. Da das CPU-Board des Cognachrome-Vision-Systems lediglich mit einem 16 MHz-Prozessor ausgestattet ist und keine Gleitkommaarithmetik ermöglicht (siehe Abschnitt 5.3), mußte ein schnelles und effizientes Verfahren entwickelt werden.

Bei diesem Verfahren wird für jeden Eckpunkt des umschreibenden Rechteckes des Blobs das eindeutige gleichschenklige und rechtwinklige Dreieck „ABC“ ermittelt, so daß

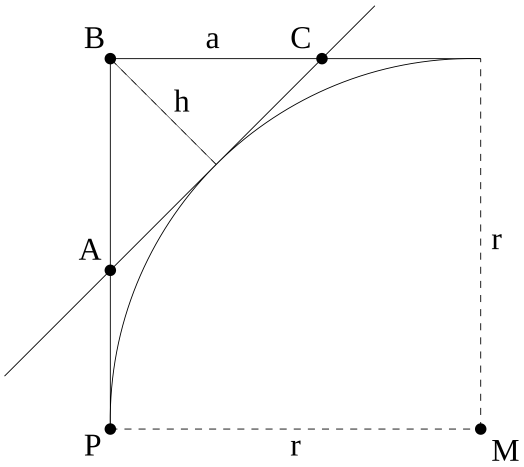

Abbildung 6.2: Das gleichschenklige Dreieck ABC

- der Dreiecks-Eckpunkt B, der gegenüber der Hypothenuse liegt, mit dem jeweiligen Eckpunkt des umschreibenden Rechteckes zusammenfällt

- die Dreieckspunkte A und C so auf den Seiten des umschreibenden Rechteckes liegen, daß die Dreiecksfläche maximal ist und das Dreieck keine Elemente des Blobs enthält.

Dieses Verfahren wird in dieser Arbeit als *Dreiecksverfahren* bezeichnet.

Wird der Ball im Binärbild als ideale Kreisscheibe abgebildet, so sind alle vier so konstruierten Dreiecke deckungsgleich und die jeweiligen Hypothenusen stellen Tangenten an den entsprechenden Kreis dar (siehe Abbildung 6.2). Möchte man zu einer gegebenen Kathetenlänge $a = \|\vec{BC}\|$ des Dreieckes den Radius r des Kreises berechnen, so ergibt sich zunächst für die Höhe h des Dreiecks:

$$h = r(\sqrt{2} - 1)$$

Aus $h = a \sin(\frac{\Pi}{4})$ folgt dann für r:

$$r = \frac{a \sin(\frac{\Pi}{4})}{(\sqrt{2} - 1)} \approx 1,7a \tag{6.3}$$

Wird nun aber der Ball nur als Teil einer Kreisscheibe abgebildet, kann meist nicht mehr von jedem beliebigen der vier konstruierten Dreiecke auf die Größe der Kreisscheibe geschlossen werden. Vielmehr findet zur Berechnung des Radius' r eine Auswahl der Dreiecke statt.

Das hier vorgestellte *Dreiecks*-Verfahren setzt voraus, daß zumindest der größte Teil eines Viertels der Kreisscheibe abgebildet wird. Dabei ist als Viertel entweder der obere-linke, obere-rechte, untere-linke oder untere-rechte Teil der Kreisscheibe gemeint. Konstruieren wir an jedem Eckpunkt des umschreibenden Rechteckes ein nach obiger Art beschriebenes gleichschenkligrechtwinkliges Dreieck, so wird davon ausgegangen, daß eines der zwei größten Dreiecke dasjenige Dreieck ist, welches an dem (fast) vollständig abgebildeten Kreisscheibenviertel liegt, und mit welchem sich nach obiger Formel der Radius $\tilde{r}$ und somit der Mittelpunkt $\tilde{M}$ am besten abschätzen läßt.

Wir bezeichnen das größte der Dreiecke mit $A_1B_1C_1$ und das zweitgrößte mit $A_2B_2C_2$. Die jeweiligen Kathetenlängen seien a_1 und a_2. Gemäß der Formel 6.3 wird für a_1 und a_2 der entsprechende Wert für r_1 und r_2 bestimmt. Da sowohl r_1 wie auch r_2 den tatsächlichen Radius $\tilde{r}$ der Kreisscheibe darstellen können, wird $\tilde{r}$ zunächst mit

$$\tilde{r} = \min(max_{length} - r_2, r_1)$$

abgeschätzt, wobei max_{length} die größere der beiden Seitenlängen des umschreibenden Rechteckes bezeichnet.

Es ist offensichtlich, daß $\tilde{r}$ mindestens $\frac{max_{length}}{2}$ betragen muß, denn sonst kann der Kreis mit Radius $\tilde{r}$ nicht den entsprechenden Blob umfassen. Damit gilt weiterhin:

$$\tilde{r} = \max(\frac{max_{length}}{2}, \tilde{r})$$

Mit Hilfe von $\tilde{r}$ wird nun – ausgehend von dem Eckpunkt B_1 des größten Dreiecks $A_1B_1C_1$ – mit Hilfe der Vektorrechnung der Mittelpunkt $\tilde{M}$ ermittelt:

$$\tilde{M} = B_1 + \tilde{r}\frac{\vec{B_1A_1}}{\|B_1A_1\|} + \tilde{r}\frac{\vec{B_1C_1}}{\|B_1C_1\|}$$

Anzumerken bleibt, daß durch diese Art der Referenzpunktbestimmung der Referenzpunkt außerhalb des Kamerabildes liegen kann. Dies kann dann der Fall sein, wenn der Ball am Rand des Sichtfeldes der Kamera liegt.

Die Berechnung des Mittelpunktes $\tilde{M}$ mittels des *Dreiecksverfahrens* ist nur sinnvoll, wenn der Blob groß genug und nicht zu schmal ist. Ist der Blob zu klein, wird das Zentrum Z des umschreibenden Rechteckes als Kreis-Mittelpunkt $\tilde{M}$ definiert. Blobs, die zu schmal sind, werden bereits bei der Blob-Validierung als ungültig verworfen.

6.3 Das Vision-Informationspaket

6.3.1 Erzeugen eines Vision-Informationspakets

Für die Übertragung der Mittelpunkts- und Radiusdaten an das Notebook wird ein spezielles Vision-Informationspaket erzeugt. Außer den Daten für Mittelpunkt und Radius werden je nach Einstellung (Modus) des *Pioneer-Vision-99*-Programms zusätzlich noch die Konturdaten des ausgewählten Blobs für eine weitere Analyse auf dem Notebook in das Paket hinzugefügt.

Für eine ausführlichere Beschreibung der Datenpakete sei auf das Kapitel C im Anhang verwiesen.

6.3.2 Senden eines Vision-Informationspakets

Die erzeugten Vision-Informationspakete werden in einem zwei-elementigen Ringpuffer gespeichert. Das jeweils aktuelle Vision-Informationspaket wird an das Notebook gesendet. Sind in den Paketen jeweils nur die Mittelpunkt- und Radius-Daten gespeichert, können aufgrund der eingestellten Baudrate von 38400 Baud alle 60 pro Sekunde erzeugten Vision-Pakete verschickt werden. Enthält dagegen das Visionpaket noch Konturdaten, so ist die Anzahl der pro Sekunde verschickten Pakete von der Größe des jeweiligen Blobs abhängig.

Kapitel 7

Das Saphira-Programm

Dieses Kapitel gibt zunächst einen kurzen Überblick über das Programm, welches auf den Notebooks beim RoboCup-Turnier 1999 lief. Besonderes Augenmerk wird dabei auf diejenigen Bereiche gelegt, die zur Positionsschätzung des Balles auf dem Spielfeld eine Rolle spielen.

Anschließend werden verschiedene Mittelungs-Verfahren besprochen, die die durch das Rauschen des Kamerabildes verursachte Dateninstabilität abfangen können. Dabei wird sowohl die Mittelung der vom CPU-Board übermittelten Mittelpunktwerte wie auch die der übermittelten Kontur beschrieben.

Im nächsten Abschnitt wird ein weiterer Algorithmus zur Mittelpunktsberechnung vorgestellt. Dieser Algorithmus basiert auf einem Kreisfitting-Verfahren, der zu ausgewählten Randpunkten eines Objektes einen Kreis liefert, so daß die Summe seiner Abstände zu diesen Randpunkten minimal ist.

Darauf folgt ein Abschnitt, in dem gezeigt wird, wie anhand einer Positionstabelle und eines errechneten Mittelpunktwerts die geschätzte Ballposition auf dem Spielfeld bestimmt und auf Gültigkeit überprüft wird. Ebenso wird die Abschätzung von der Geschwindigkeit des Balles beschrieben.

Der Schluß des Kapitels behandelt die Fusionierung der Ballpositionsdaten, die die einzelnen Spieler unabhängig voneinander bestimmt haben. Es wird gezeigt, wie ein solcher Abgleich der Daten zu einer robusteren Ballpositionsschätzung beiträgt. Aber auch ein Nachteil des Verfahrens wird erläutert, der in der Praxis zu mancher Irritation auf dem Spielfeld beigetragen hat.

7.1 Das Saphira-System

7.1.1 Saphira-System-Architektur

Die Saphira-System-Architektur stellt das Basissystem zur Roboter-Steuerung dar. Das entsprechende Software-Programm wurde am SRI[1] International entwickelt und ist im Lieferumfang der Pioneer-Roboter enthalten. Das Saphira-System ist eine Entwicklungsumgebung, in der Programme geschrieben werden können, die die eigentliche Roboter-Steuerung übernehmen.

Das mit Hilfe dieses Systems entwickelte *Saphira-99*-Programm,[2] ist in verschiedene Aufgabenbereiche unterteilt:

- Bestimmung der Ballposition auf dem Spielfeld mit Hilfe der vom Cognachrome-Vision-System bereitgestellten Bilddaten,
- Positionsbestimmung des Spielers auf dem Spielfeld (Selbstlokalisation) mittels Odometrie- und Laserscannerdaten,
- Positionsbestimmung anderer Spieler auf dem Spielfeld anhand von Laserscannerdaten,
- Erzeugung und Verwaltung eines lokalen Weltmodelles, das die wahrgenommenen Objekte eines Spielers wiedergibt,
- optionale Kommunikation mit einem Zentralrechner, auf dem ein Serverprogramm läuft, das Daten von den einzelnen Spielern empfängt (z.B. die *lokalen* Weltmodelldaten) wie auch Daten an die einzelnen Spieler versendet (z.B. die aus den *lokalen* Weltmodell erstellten *globalen* Weltmodelldaten) und
- Pfadplanung auf der Basis des lokalen, bzw. – soweit vorhanden – des globalen Weltmodelles zur Berechnung eines kollisionsfreien Pfades von der momentanen Spielerposition zu einem bestimmten Zielpunkt.

In den folgenden Abschnitten wird lediglich auf die Bereiche eingangen, die relevant für die Ballpositions-Schätzung sind. Alle anderen Bereiche werden

[1] Standford Research Institute

[2] Der Einfachheit halber werden in dieser Arbeit die auf dem Notebook laufenden Saphira-Programme als *Saphira-98*-Programm (RoboCup '98) bzw. *Saphira-99*-Programm (RoboCup '99) bezeichnet.

ausführlich in den Arbeiten von Weigel [Weigel, 1999], Topor [Topor, 1999] und Reetz [Reetz, 1999] behandelt.

7.1.2 Rauschen des Kamerabildes

Aufgrund der Lichtverhältnisse und der Konzeption des Cognachrome-Vision-Systems ist das Kamerabild erheblich verrauscht. Dies führt dazu, daß bei einem ruhenden Ball der ihn repräsentierende Blob im Binärbild stets seine Form ändert. In den Abbildungen 7.1, 7.2 und 7.3 werden die Binärbilder von einem sich nicht bewegenden Ball aus drei hintereinanderfolgenden Kamerabildern gezeigt.

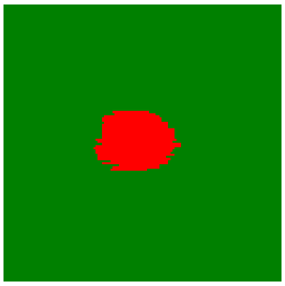

Abbildung 7.1: Blob1

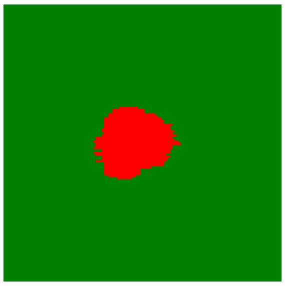

Abbildung 7.2: Blob2

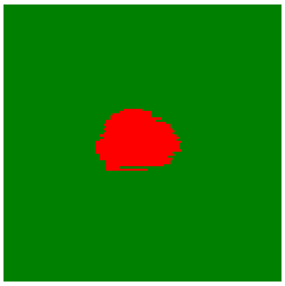

Abbildung 7.3: Blob3

Diese in den Bildern deutlich sichtbaren Formänderungen können einen wechselnden Mittelpunkt und somit einen sich stets ändernden Wert für die Ballposition auf dem Spielfeld zur Folge haben. Da dieser Wert für die Abschätzung der Ballgeschwindigkeit und damit für die Abschätzung von nachfolgenden Ballpositionen herangezogen wird, kann man hierbei irreführende Resultate erhalten. Daher wurde im *Saphira-99*-Programm eine Mittelung der Ballposition über die Zeit eingeführt.

7.2 Mittelung

Im *Saphira-99*-Programm hängt die Art und Weise der Mittelung von einem auf dem CPU-Board eingestellten Modus ab. Der Modus gibt dabei an, ob die Konturdaten des Blobs im Binärbild zur weiteren Analyse übermittelt werden oder nicht.

7.2.1 Mittelung des Mittelpunktes

Werden die Konturdaten nicht übermittelt, so werden die Koordinaten des vom CPU-Board gesendeten Mittelpunktes M wie auch der Radius R mit den Koordinaten der Mittelpunkte und Radien der maximal $num_{avg} - 1$ vielen vorangegangenen Vision-Informationspakete arithmetisch gemittelt. Dabei ist der Wert num_{avg} ein einstellbarer Parameter und muß größer gleich 1 sein.

Es zeigt sich aber, daß der Wert für num_{avg} nicht ungerade sein sollte. Durch die im Binärbild gemachte Beobachtung konnte man feststellen, daß bei einem ruhenden Ball der übermittelte Mittelpunkt häufig zwischen zwei (meist benachbarten) Punkten p_1 und p_2 hin- und herspringt. Mittelt man über die letzten num_{avg} Mittelpunkte und ist num_{avg} ungerade, so alterniert auch das Ergebnis der arithmetischen Mittelung zwischen zwei Punkten, wie die folgende Rechnung zeigt:

Sei x_1 die X-Koordinate von p_1 und x_2 die von p_2. O.B.d.A. sei ferner $x_1 \neq x_2$ und $d = x_2 - x_1, d \in Z$. Ist n die Anzahl des Vorkommens des Mittelpunktes p_1 und m die entsprechende Anzahl des Mittelpunktes p_2 unter den letzten num_{avg} vielen übermittelten Paketen, so gilt: $num_{avg} = n+m, n, m \in \mathbf{N}$. Das arithmetische Mittel x für die X-Koordinate errechnet sich demnach folgendermaßen:

$$\begin{aligned} x &= \frac{1}{num_{avg}}(nx_1 + mx_2) = \frac{1}{num_{avg}}(nx_1 + m(x_1 + d)) \\ &= \frac{1}{num_{avg}}((n+m)x_1 + md) = x_1 + d\frac{m}{num_{avg}} \end{aligned} \tag{7.1}$$

Wegen des Sprungverhaltens des Mittelpunktes ist m einmal um 1 kleiner und dann wiederum um 1 größer als n. Da num_{avg} ungerade ist, springt der Quotient $\frac{m}{num_{avg}}$ zwischen einem Wert kleiner als $\frac{1}{2}$ und einem Wert größer als $\frac{1}{2}$ hin und her. Entsprechend alterniert x zwischen zwei X-Koordinaten.

Dennoch läßt sich dieser Sprung-Effekt allein durch arithmetische Mittelung nicht ganz vermeiden. Daher wird der aus der Mittelung errechnete Mittelpunkt $\tilde{M_{act}}$ mit dem Mittelpunkt $\tilde{M_{pre}}$ der vorangehenden Mittelung verglichen. Ist der Pixelabstand[3] zwischen $\tilde{M_{act}}$ und $\tilde{M_{pre}}$ kleiner als ein vorgegebener Parameter $nomove_{tolerance}$,[4] so wird angenommen, daß sich der

[3]In dieser Arbeit wird der Pixelabstand aus Effizienzgründen nicht über die euklidische, sondern immer über die Maximumsnorm ermittelt.

[4]Bei den RoboCup-Turnieren wurde für $nomove_{tolerance}$ der Wert 1 gewählt.

Ball nicht bewegt hat. In diesem Falle werden als Ergebnis der Mittelung die Werte der vorangegangenen Mittelung übernommen.

Die so durchgeführte Mittelung liefert für einen ruhenden Ball relativ stabile Mittelpunkt- und Radiuswerte. Für einen sich relativ zum Spieler bewegenden Ball ist der Effekt der Mittelung jedoch nicht erwünscht. In diesem Fall zieht die errechnete Ballposition der realen Ballposition nach. Daher wurde ein weiterer einstellbarer Parameter $max_{distance}$ eingeführt, der angibt, welchen Abstand von M die vorangegangenen Mittelpunkte höchstens haben dürfen, um in die Mittelung miteinbezogen zu werden. Dabei gibt $max_{distance}$ nicht die absolute, sondern die zu dem Radius R relative Distanz an. Dies geschieht, um die Entfernung des Balles zum Spieler mit zu berücksichtigen.

7.2.2 Kontur-Mittelung

Werden neben dem Mittelpunkt M und dem Radius R auch die Konturdaten in den Vision-Informationspaketen mitgesendet, so wird zunächst über die num_{avg} letzten vom *PioneerVision-99*-Programm bereitgestellten Konturdaten des Blobs gemittelt. Ebenso wie bei der Mittelpunkts-Mittelung werden nur diejenigen Konturdaten in die Mittelung mit einbezogen, bei denen die auf dem CPU-Board ermittelten Mittelpunkte zum aktuellen Mittelpunkt M eine Distanz von maximal $max_{distance}$ besitzen.

Das Verfahren der Kontur-Mittelung basiert nicht auf einer arithmetischen Mittelung, sondern auf einer punktweisen Auswertung der Kontur. Liegt ein Konturpunkt nicht auf, oder innerhalb mindestens der Hälfte der Konturen, über die gemittelt wird, so wird dieser als Rauschen interpretiert und verworfen. Die verbleibenden äußeren Konturpunkte bilden die gemittelte Kontur. Auf diese wird anschließend eine weiter unten beschriebene Mittelpunktsberechnung anhand des Kreisfittingverfahrens durchgeführt. Der daraus resultierende Mittelpunkt M_c wird dann anhand des im letzten Abschnitt 7.2.1 beschriebenen Verfahrens nochmals gemittelt.

7.2.3 History-Tabelle

Sowohl die gemittelten als auch die ungemittelten Mittelpunkt- und Radiusdaten der letzten 20 Vision-Informationspakete werden in einer sogenannten History-Tabelle gespeichert. Dabei werden die aktuellen Daten an den Anfang der Tabelle gesetzt und die restlichen Daten in der Tabelle dementsprechend verschoben. Diese History-Tabelle dient zur Mittelung der Mittelpunkte und Radien sowie zur Geschwindigkeitsschätzung des Balles auf dem Spielfeld.

7.3 Mittelpunktsberechnung mittels Kreisfitting

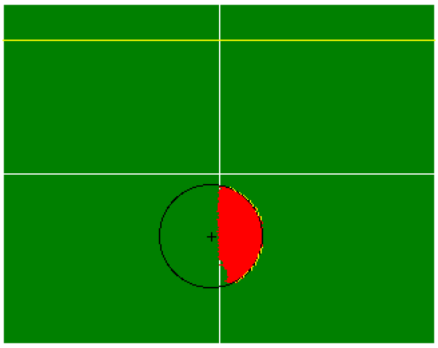

Abbildung 7.4: Das Kreisfittingverfahren

Im Zuge dieser Arbeit wurde neben der Mittelpunktsberechnung mittels Dreiecksverfahren eine weitere, weitaus rechenintensivere Variante der Mittelpunktsberechnung implementiert. Dabei wird mit Hilfe der übermittelten Konturdaten derjenige Kreis gesucht, der durch die Kontur am besten repräsentiert wird, d.h. es wird der Kreis gesucht, bei dem die Summe seiner Abstände zu den ausgewählten Konturpunkten minimal ist.

Betrachtet man die gemittelten Konturpunkte und den vom CPU-Board übermittelten Mittelpunkt M in einer Ebene, so läßt sich diese Kontur in der vertikalen und in der horizontalen Komponente längs des Mittelpunktes M teilen. Die so entstandenen vier Konturabschnitte seien als Kontursegmente bezeichnet. Die Punkte der Kontursegmente bilden jeweils eine Punktmenge $P_i, 1 \leq i \leq 4$. Eine Punktmenge kann durchaus auch leer sein. Dies ist dann der Fall, wenn der Mittelpunkt M außerhalb der Kontur liegt, wenn beispielsweise der Ball zu mehr als der Hälfte verdeckt wird.

Von jedem Punkt der Konturpunktmenge wird der Abstand zu M berechnet. Ist der Abstand kleiner als $\frac{5}{6}R$, wobei R den vom CPU-Board übermittelten Radius angibt, so wird der entsprechende Punkt aus der Punktmenge entfernt. Die Motivation hierfür liegt u.a. darin, daß der Blob im Binärbild, bedingt durch bestimmte Lichtverhältnisse und Schatten, eine Aussparung besitzen kann, deren Konturpunkte das Ergebnis des Kreisfitting-Algorithmus' verfälschen würden. Auch im Falle, daß der Mittelpunkt M außerhalb der Kontur liegt, dürfen die dem Punkt M naheliegenden Konturpunkte nicht in die Berechnung mit einbezogen werden.

Für jede Konturpunktmenge P_i wird anschließend der Kreisfitting-Algorithmus, der in Kapitel 9 ausführlich beschrieben wird, durchgeführt, sofern die Anzahl der verbleibenden Punkte in der Menge eine bestimmte Schwelle übersteigt. Ist die Anzahl zu klein, dann ist das Ergebnis des Algorithmus' zu ungenau. Bei den RoboCup-Spielen wurde eine Mindestanzahl $min_{contourset}$ von 10 Pixeln festgesetzt.

Der Kreisfitting-Algorithmus liefert zu einer übergebenen Punktmenge P_i den Mittelpunkt M_{c_i} und den Radius R_{c_i} eines Kreises C_i, sowie einen Abstandsfehler F_{c_i}. Der Abstandsfehler F_{c_i} ist dabei ein Maß für den durchschnittlichen Abstand der Punkte aus P_i zum ermittelten Kreis C_i. Je kleiner der Fehler F_{c_i}, desto besser werden die Punkte aus P_i durch den Kreis C_i repräsentiert.

Diejenigen Mengen P_i, auf die der Kreisfitting-Algorithmus angewendet worden ist und die dabei einen relativ kleinen Fehler F_{c_i} liefern, werden zu einer neuen Punktmenge P vereinigt. Ist diese Menge nicht leer, so wird auf ihr erneut der Kreisfitting-Algorithmus durchgeführt. Das Ergebnis ist dann der gesuchte Mittelpunkt M_c und der Radius R_c. Ist die Punktmenge P jedoch leer, so wird der gesuchte Mittelpunkt M_c mit dem übermittelten Mittelpunktswert M und der gesuchte Radius R_c entsprechend mit dem übermittelten Radius R gleichgesetzt.

7.4 Positionsberechnung

Das Cognachrome-Vision-System bildet die dreidimensionale reale Welt auf ein zweidimensionales Kamerabild ab, welches sich aus diskreten Bildpunkten zusammensetzt.[5] Im Kamerabild sind drei Größen meßbar: die X- und Y-Koordinate des Mittelpunktes und der Radius des Balles im Kamerabild. Da die Größe des Balles auf dem Spielfeld sowie die Kameraeinstellung[6] bekannt sind, kann man anhand der Bilddaten die Position des Balles in der dreidimensionalen Welt bestimmen.

Beim Roboter-Fußball befindet sich der Ball normalerweise auf dem Boden.[7]

[5] Es wird somit eine Abbildung vom R^3 in einen $X \times Y$-Bereich vollzogen, wobei X und Y einen Unterraum von N bilden.

[6] Position auf dem Roboter, Neigungswinkel und Brennweite

[7] Beim Halbfinalspiel *Azzurra Robot Team* (Italien) gegen das CS-Freiburg-RoboCup-

Somit reduziert sich die Positionsbestimmung des Balles in der realen Welt auf ein zweidimensionales Problem. Mit den drei oben erwähnten Größen (X- und Y-Koordinaten und Radius) ist das $2D$-Problem überbestimmt. Die Redundanz des Radius kann deshalb zur Verifikation des errechneten Ballmittelpunktes im Bild benutzt werden.

7.4.1 Positionstabelle

Zur Schätzung der Ballposition bedient man sich einer zuvor generierten Tabelle, die in dieser Arbeit als Positionstabelle bezeichnet wird. Die Positionstabelle ist eine durch eine Meßreihe generierte zweidimensionale Tabelle, die über die Bildpunkte des Kamerabildes indiziert ist. Ein Feld der Positionstabelle enthält dabei zu dem jeweiligen Bildpunkt eine relative Position des Balles im Spielfeld bezüglich der Kamera, und zur Validierung einen Erwartungswert für den Radius des Balles im Binärbild.[8] Die Daten hierfür werden im Zuge eines Kalibrierungsverfahrens gewonnen, das im Kapitel 8 ausführlich beschrieben wird.

Sind die Pixelkoordinate des Ballmittelpunktes im Bild gegeben, kann mit Hilfe der erstellten Positionstabelle auf die Position des Balles auf dem Spielfeld geschlossen werden.

7.4.2 Verifikation

Der ebenfalls in der Positionstabelle gespeicherte Erwartungswert für den Radius des Balles im Bild dient zur Verifikation der errechneten Daten. Hierbei wird überprüft, ob der ermittelte Radius $\tilde{R}$ mit dem erwarteten Radius bezüglich des errechneten Mittelpunktes $\tilde{M}$ übereinstimmt. Weicht der ermittelte vom erwarteten Radius um mehr als einen vorher festgelegten Prozentwert ab, wird davon ausgegangen, daß es sich bei dem Objekt im Binärbild nicht um den Ball handelt. Die aktuellen Mittelpunkt- und Radius-Daten werden dann in der History-Tabelle (siehe Abschnitt 7.2.3) als ungültig markiert und damit für die Mittelung sowie für die Positionsbestimmung des Balles auf dem Spielfeld nicht mehr herangezogen. Befindet sich der ermittelte Radius jedoch innerhalb der Toleranzgrenzen, so wird die für den Mittelpunkt $\tilde{M}$ in der Positionstabelle gespeicherte (relative) Ball-

Team während der RoboCup-Weltmeisterschaft '99 in der „Middle-Size-League“ verwandelten die Italiener einen Elf-Meter per Hochschuß.

[8] Im *Saphira-98*-Programm wird zur Validierung der Erwartungswert der Ballfläche in der Positionstabelle gespeichert.

position als korrekt angenommen. Mit Hilfe dieser relativen Position wird dann die absolute Position des Balles auf dem Spielfeld berechnet.

7.4.3 Ballposition bezüglich des Roboters

Die Positionsberechnung des Balles auf dem Spielfeld wurde bisher bezüglich der Kamera berechnet. Daher muß zunächst die Position des Balles relativ zum *Roboter* berechnet werden. Relativ zum Roboter heißt in diesem Falle relativ zum Ursprung des zweidimensionalen Roboterkoordinatensystems. Dabei ist der Radachsenmittelpunkt der Vorderräder als Ursprung definiert worden. Die Radachse selbst bildet die Y-Achse und die Vorwärtsfahrtrichtung des Roboters bestimmt die positive X-Achse des Roboters (siehe Abbildung 7.5). Ebenso wurde ein zweidimensionales Kamera-Koordinatensystem definiert. Dabei bildet die Kamera-Linse den Ursprung und die „Blickrichtung" der Kamera die positive X-Achse.

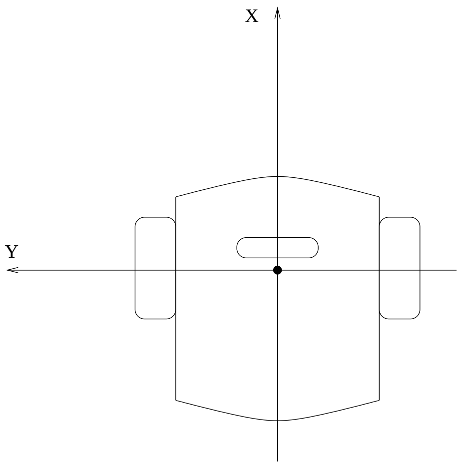

Abbildung 7.5: Das Roboter-Koordinatensystem

Da bei den Spielern die Kameras bezüglich der Radachsen versetzt montiert wurden, wird mit Hilfe einfacher Koordinatentransformation die Ballposition relativ zum Roboterursprung bestimmt. Eine detaillierte Beschreibung dieser Transformation findet sich bei Topor [Topor, 1998].

7.4.4 Bestimmung der globalen Ballposition

Ist die relative Position des Balles zum Roboter bestimmt, wird die absolute Position des Balles auf dem Spielfeld berechnet. Dies geschieht erneut anhand einer Koordinatentransformation, wobei die Positionsdaten des Balles im Roboter-Koordinatensystem in das Spielfeld-Koordinatensystem transformiert werden. Im Spielfeld-Koordinatensystem bildet die vom Torwart aus betrachtete linke Bande die positive Y-Achse und die Rückseite des Tores die X-Achse.

Für die o.g. Transformation sind die Daten der Position und Ausrichtung des Roboters bezüglich des Spielfeld-Koordinatensystems notwendig. Diese Daten werden mit dem Laserscanner bei der Selbstlokalisation des Spielers gewonnen. Da die Ermittlung der Positionsdaten des Spielers und des Balles nicht synchron verläuft, erhält sowohl die Messung der Ball- sowie die der Spielerposition einen Zeitstempel. Mit Hilfe dieses Zeitstempels wird zu einer vorgebenen Ballpositionszeit die Position und Ausrichtung des Roboters mittels Interpolation ermittelt [Dümler, 1999]. Sind die Roboterdaten auf diese Weise ermittelt worden, wird die globale Ballposition anhand einer Koordinatentransformation bestimmt.

7.4.5 Geschwindigkeitsschätzung des Balles

Mittels der aktuellen und der vorangegangenen globalen Ballposition wird anhand der Zeitdifferenz T und der dabei zurückgelegten Wegstrecke W die Geschwindigkeit V und die Richtung des Balles geschätzt. Die Richtung des Balles ergibt sich aus der Differenz der aktuellen zur vorangegangenen Ballposition. Die Geschwindigkeit V wird mit $V = \frac{W}{T}$ bestimmt.

Anhand von Geschwindigkeit und Richtung des Balles können die nächsten Ballpositionen im voraus abgeschätzt werden. Eine robuste Bestimmung der zukünftigen Ballposition kann insbesondere für das Verhalten des Torwartes spielentscheidend sein.

7.5 Fusionierung der Ballpositionsdaten

Die ermittelten Ballpositionsdaten werden in das vom Spieler verwaltete lokale Weltmodell übertragen. Dieses Weltmodell *gibt die Sicht der Welt aus der Perspektive eines Spielers wieder*[Weigel, 1999]. Sofern eine Anbindung zum Server besteht, werden die Daten des Weltmodelles an den Server übermittelt.

7.5.1 Das globale Weltmodell

Aufgrund unvollständiger Daten sowie fehlerhafter Interpretationen der Sensorinformationen (Laserscanner-, Kamera- und Ultraschalldaten) werden die Daten des *lokalen* Weltmodelles eines jeden Spielers auf dem Zentralrechner zu einem *globalen* Weltmodell fusioniert [Dietl, 1999]. Dieses Weltmodell wird dann wiederum allen Spielern zugänglich gemacht. Auf diese Weise ist es möglich, einem Spieler eine für ihn nicht erfaßbare Position des Balles mitzuteilen.

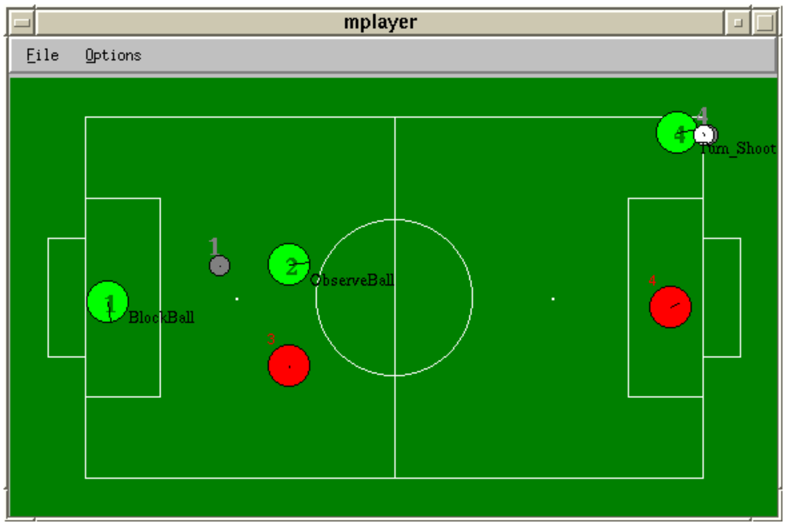

Abbildung 7.6: Der vom Torwart gesehene Geisterball (grau) wird durch die Fusionierung der Ballpositionsdaten als solcher erkannt und verworfen. Die vom Zentralrechner geschätzte Ballposition ist in der Abbildung weiß dargestellt.

Noch bedeutender ist der Informationsabgleich in den Fällen, in denen einzelne Spieler unterschiedliche Ballpositionen ermittelt haben (siehe Abbildung 7.6). Stimmen die Werte der anderen Spieler bezüglich der Ballposition weitgehend überein, so kann die fehlerhafte Ballpositionsschätzung verworfen und über die Ballpositionsdaten der anderen Spieler gemittelt werden. Die so ermittelte Ballposition wird dann allen Spielern gesendet, die ihrerseits damit die Position des Balles in ihrem *lokalen* Weltmodell aktualisieren. Die Fusionierung der Positionsdaten stellt somit ein weiteres wichtiges Faktum für die Robustheit und Zuverlässigkeit der Ballpositionsschätzung dar.

Unzulänglichkeiten der Weltmodellverwaltung

Mit jeder neuen Auswertung der Kameradaten wird die Ballposition im Weltmodell aktualisiert. Liegt zum aktuellen Zeitpunkt für einen bereits im Weltmodell vorhandenen Ball keine Sensorinformation mehr vor, wird die aktuelle Ballposition mit Hilfe vergangener Ballpositionen geschätzt und zunächst als „erinnert“ markiert. Erst nach einer vorgegebenen Verweildauer im Weltmodell ohne Sensoraktualisierung wird der Ball aus dem Weltmodell entfernt.

Der Zweck dieses Vorgehens liegt darin, daß der Spieler die Position des Balles auch dann noch „weiß“ und sein Spielverhalten dementsprechend ausrichten kann, wenn der Ball kurzfristig aus dem Sichtfeld der Kamera verschwunden ist. Die Zeitspanne, in der die Position in „Erinnerung“ bleibt, ist dabei gerade so gewählt, daß ein Spieler den Ball umfahren kann, ohne diesen dabei aus dem „Gedächtnis“ zu verlieren.

Das Verfahren des „Erinnerns“ hat aber einen Nachteil: Ist zu einem gewissen Zeitpunkt der Ball für keinen der Spieler mehr sichtbar – sei es, weil er von Gegenspielern verdeckt wird oder zu weit entfernt liegt – kann folgender Fall eintreten:

Nehmen wir an, daß ein Spieler für kurze Zeit fälschlicherweise den Ball an einer anderen Position sieht. Die Zeitspanne ist dabei gerade noch so lang, daß der Plausibilitätstest des *PioneerVision-99*-Programms den Blob im Binärbild als Ball erkennt (siehe Abschnitt 6.2.1).

Der Spieler, der den „falschen“ Ball sieht, trägt diesen in sein *lokales* Weltmodell ein und übermittelt die Informationen an den Zentralrechner. Der Zentralrechner fusioniert die Daten der *lokalen* Weltmodelle aller Spieler und aktualisiert das *globale* Weltmodell. Da die anderen Spieler keine Ballpositionsdaten übertragen haben, führt dies dazu, daß die Position des „falschen“ Balles als richtig erkannt und in das *globale* Weltmodell eingetragen wird.

Das *globale* Weltmodell wird an alle Spieler gesendet, die ihr *lokales* Weltmodell entsprechend aktualisieren und dabei folglich die Position des „falschen“ Balles übertragen.

Tritt nun der Fall ein, daß im nächsten Moment das Programm des betreffenden Spielers den *Geisterball* als solchen erkennt, und der „richtige“ Ball für die Spieler immer noch nicht sichtbar ist, so erhält der Zentralrechner von keinem der Spieler mehr eine Aktualisierung der Ballposition. Dies führt dazu, daß der als Ball gespeicherte *Geisterball* während der oben genannten Verweildauer in den jeweiligen Weltmodellen als „erinnert“ markiert wird. Während dieser Zeitspanne richten die Spieler ihr Spielverhalten gemäß dieser vermeintlich richtigen Ballposition aus. Das bedeutet, daß die Spieler beispielsweise dieser Position entgegenfahren, anstatt weiterhin nach der Position des „echten“ Balles zu suchen.

Umgehen ließe sich das Problem, indem die Zeitspanne des „Erinnerns“ in Abhängigkeit von der Zuverlässigkeit der letzten Ballposition formuliert werden würde. Eine Ballposition wird demzufolge als „recht sicher“ eingestuft, wenn der Ball über eine längere Zeitspanne dort „gesehen“ wird. Noch effizienter wäre es, auf das Vorgehen des „Erinnerns“ zu verzichten. Wenn die Roboter anstatt mit einer festmontierten mit einer schnell- beweglichen 360° Kamera ausgestattet werden, die den Ball permanent verfolgen kann.

Kapitel 8

Kalibrierung

In diesem Kapitel wird beschrieben, wie die zur Positionsschätzung des Balles benötigten Kalibrierungsdaten gewonnen werden. Insbesondere wird die verwendete Art der Interpolation ausführlich behandelt.

8.1 Ermittlung der Kalibrierungsdaten

Die Kalibrierungsdaten sind Meßdaten, die im Rahmen einer Meßreihe zu einer bestimmten Kameraeinstellung aufgenommen werden. Hierbei werden für verschiedene Ballpositionen auf dem Spielfeld die zugehörigen Mittelpunkte und Größen des Balles im Kamerabild gemessen. Anschließend wird eine zweidimensionale Positionstabelle generiert, die über die Bildpunkte des Kamerabildes indiziert ist. Ein Feld dieser Positionstabelle speichert dabei die relative Position des Balles im Spielfeld und den entsprechenden Radius des Balles im Bild.

Bei der Erstellung der Kalibrierungsdaten für die Positionstabelle wird nicht für jeden der über 50000 Pixel des Kamerabildes eine Ballposition im Spielfeld gemessen. Vielmehr wird eine bestimmte Anzahl von Meßpunkten ausgewählt. Je mehr Meßpunkte bei der Kalibrierung verwendet werden, desto genauer kann dann die Zuordnung einer Pixelposition zu einer Position auf dem Spielfeld erfolgen. Die Werte für diejenigen Pixelpositionen, für die keine Ballposition gemessen wird, werden durch lineare Interpolation ermittelt.

8.2 Auswahl der Meßpunkte

Bei der Auswahl der Meßpunkte wird zunächst die maximale Entfernung $maxdistance_{measure}$, in der ein Ball von dem Bildverarbeitungssystem noch zuverlässig erkannt werden kann, bestimmt. Zusammen mit dem Sichtfeld der Kamera beschreibt dieser Wert in der realen Welt bezüglich des Roboters eine Fläche, die mit Meßpunkten abgedeckt werden soll (siehe Abbildung 8.1).

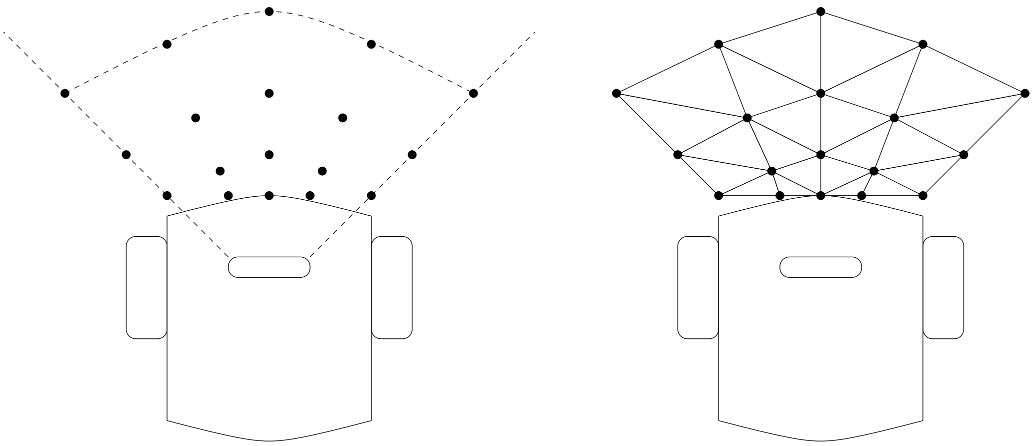

Abbildung 8.1: Links: Meßpunkte innerhalb einer vorgegebenen Entfernung und innerhalb des Sichtfeldes der Kamera - Rechts: Triangulation der Meßpunkte

Beim *Saphira-99*-Programm wird nicht wie im *Saphira-98*-Programm die zum Roboter relative Ballposition *manuell* verändert, sondern die zum Ball relative Roboterposition verändert sich *automatisch*: der Roboter wechselt während dem Kalibrierungsvorgang selbständig seine Position. Der Ball bleibt dabei unverändert liegen.

Beim Start der Kalibrierung wird der Ball zunächst direkt vor den Roboter gelegt. Da der Abstand der Robotervorderachse zum Ballmittelpunkt bekannt ist, kann aufgrund der genauen Positionsschätzung des Roboters durch den Laserscanner die aktuelle Position des Balles auf dem Spielfeld bestimmt werden. Anstatt den Ball nun zu bewegen, entfernt sich der Roboter „schrittweise“ vom Ball weg. Nachdem eine zuvor definierte Entfernung von $step_{measure}$ zurückgelegt worden ist, beginnt eine Meßreihe. Dabei dreht sich der Roboter zunächst solange, bis der Ball am Rand des Kamerabildes verschwindet. Dann dreht sich der Roboter in die entgegengesetzte Richtung. Bei dieser Drehung stoppt er nach jeweils einem vorgegebenen Winkel $anglestep_{measure}$, generiert anhand den Daten aus dem Binärbild die jeweiligen Kalibrierungsdaten und dreht sich dann wieder um den Winkel $anglestep_{measure}$. Diese Vorgang wird solange wiederholt, bis der Ball am anderen Rande des Kamerabildes verschwindet. Danach dreht sich der Roboter wieder zu der Position, bei der die Meßreihe begann. Nun bewegt sich der Roboter um die Distanz $step_{measure}$ vom Ball weg und eine neue Meßreihe beginnt. Der Kalibrierungsvorgang endet, wenn der Roboter vom Ball eine Entfernung von $maxdistance_{measure}$ zurückgelegt hat.

Bei jeder Position an der der Roboter stoppt, ermittelt das Kalibrierungsprogramm zu der relativen Ballposition auf dem Spielfeld den zugehörigen Mittelpunkt und Radius des Balles im Kamerabild mittels Kreisfitting-Algorithmus. Um den Fehler der durch das Rauschen des Kamerabildes entsteht so gering wie möglich zu halten, wird pro Stop mehrmals der entsprechende Mittelpunkt und Radius bestimmt und darüber gemittelt.

Die Koordinaten des gemittelten Mittelpunktes bilden den Zeiger auf ein Feld in der Positionstabelle, in welches die relative Ballposition zum Roboter, sowie der entsprechend gemittelte Ballradius im Kamerabild gespeichert werden. Die relative Ballposition kann aufgrund der genauen Selbstlokalisation des Roboters und der Kenntnis der globalen Position des Balles ermittelt werden.

8.3 Interpolation

Für diejenigen Pixelpositionen, für die keine Ballposition gemessen wurde, müssen die entsprechenden Werte durch Interpolation gewonnen werden. Bei der Interpolation wird zunächst eine Liste von Dreiecken generiert. Die Eckpunkte der Dreiecke sind dabei die bei der Kalibrierung aufgenommenen Meßpunkte im Kamerabild (siehe Abbildung 8.1). Soll die Position des Balles auf dem Spielfeld anhand des Ballmittelpunktes M im Kamerabild ermittelt werden, werden die drei Eckpunkte M_1, M_2 und M_3 desjenigen Dreieckes zur Berechnung verwendet, das den Punkt M enthält.

Die Generierung der Dreiecksliste geschieht mit Hilfe des *Delaunay*-Triangulationsverfahrens [Fortune, 1992], welches eine gleichmäßige Triangulation erzeugt, bei der keines der Dreiecke ein anderes Dreieck enthält. Das Auffinden des Dreieckes, das den Pixelpunkt M enthält, geschieht dann mittels einer sequenziellen Suche, die abbricht, sobald ein entsprechendes

Dreieck gefunden wurde. Der Test, ob der Punkt M in einem Dreieck mit den Dreieckspunkten M_1, M_2 und M_3 liegt, wird durch einfache Vektorrechnung realisiert. Eine ausführliche Beschreibung findet sich bei Topor[Topor, 1998].

Die Interpolation im *Saphira-99*-Programm basiert auf der Berechnung von Teildreiecksflächen. Im folgenden wird dieses Verfahren näher erläutert.

8.3.1 Der Operator $\otimes$

An dieser Stelle soll ein Operator $\otimes$ eingeführt werden. Seien $\vec{A} = (a_x\ a_y)^T$, $\vec{B} = (b_x\ b_y)^T$ und $\vec{C} = (c_x\ c_y)^T$ zweidimensionale Vektoren. Der Operator $\otimes$ wird nun definiert als

$$\vec{A} \otimes \vec{B} := (a_x b_y - a_y b_x),\ \mathbf{R^2} \times \mathbf{R^2} \to \mathbf{R} \tag{8.1}$$

$\vec{A} \otimes \vec{B}$ stellt die Fläche des von $\vec{A}$ und $\vec{B}$ aufgespannten Parallelogrammes dar. Folgende Regeln lassen sich herleiten:

$$\vec{A} \otimes \vec{A} = (a_x a_y - a_y a_x) \quad = \quad 0 \tag{8.2}$$

$$\begin{aligned} \vec{A} \otimes \vec{B} = (a_x b_y - a_y b_x) &= -((b_x a_y - b_y a_x)) \\ &= -\vec{B} \otimes \vec{A} \end{aligned} \tag{8.3}$$

$$\begin{aligned} ((\vec{A} - \vec{B})) \otimes ((\vec{C} - \vec{B})) &= (a_x - b_x)(c_y - b_y) - (a_y - b_y)(c_x - b_x) \\ &= (a_x c_y - a_y c_x) - (b_x c_y - b_y c_x) - (a_x b_y - a_y b_x) \\ &= \vec{A} \otimes \vec{C} - \vec{B} \otimes \vec{C} - \vec{A} \otimes \vec{B} \end{aligned} \tag{8.4}$$

$$\begin{aligned} (\vec{A} \otimes \vec{B})\vec{C} - (\vec{C} \otimes \vec{B})\vec{A} &= \begin{pmatrix} (a_x b_y - a_y b_x)c_x - (c_x b_y - c_y b_x)a_x \\ (a_x b_y - a_y b_x)c_y - (c_x b_y - c_y b_x)a_y \end{pmatrix} \\ &= \begin{pmatrix} (a_x c_y - a_y c_x)b_x \\ (a_x c_y - a_y c_x)b_y \end{pmatrix} \\ &= (\vec{A} \otimes \vec{C})\vec{B} \end{aligned} \tag{8.5}$$

Aus 8.5 und 8.3 folgt

$$(\vec{A} \otimes \vec{B})\vec{C} + (\vec{B} \otimes \vec{C})\vec{A} + (\vec{C} \otimes \vec{A})\vec{B} = \vec{0} \tag{8.6}$$

8.3.2 Die Interpolationsformel

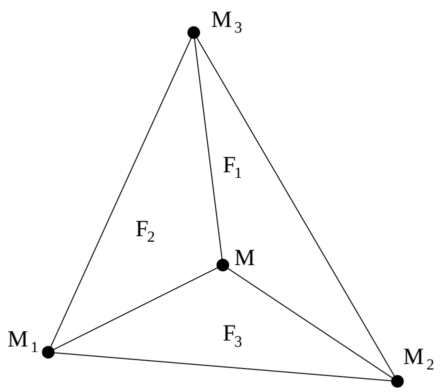

Abbildung 8.2: Interpolation mittels der Dreiecksflächen F_1, F_2 und F_3

Sei M der Ballmittelpunkt im Kamerabild und seien M_1, M_2 und M_3 die drei Eckpunkte desjenigen Dreieckes D der Dreiecksliste, das den Mittelpunkt M enthält. Verbindet man jeden Eckpunkt des Dreieckes D mit dem Mittelpunkt M, dann erhält man Teildreiecke, deren Flächeninhalt mit F_1, F_2 und F_3 bezeichnet seien. Dabei sei F_i die Fläche des dem Punkt M_i gegenüberliegenden Dreiecks (siehe Abbildung 8.2). Die Fläche des Dreiecks D sei mit F bezeichnet.

Seien $\vec{M_i}, 1 \leq i \leq 3$ die Eckpunkte des Dreieckes und $\vec{M}$ der Mittelpunkt in Vektorschreibweise. Die Darstellung des Punktes $\vec{M}$ als Linearkombination der Dreieckspunkte läßt sich folgendermaßen darstellen:

$$\vec{M} = \sum_{i=1}^{3} a_i \vec{M_i},\ mit\ a_i = \frac{F_i}{F} \tag{8.7}$$

Im folgenden wird die Behauptung 8.7 bewiesen. Für den Beweis sei $\vec{M_i} = \vec{M_{i+3}}, 1 \leq i \leq 3$. Damit gilt

$$\sum_{i=1}^{3} (\vec{M_i} \otimes \vec{M_j})\vec{M_k} = \sum_{i=1}^{3} (\vec{M_{i+1}} \otimes \vec{M_{j+1}})\vec{M_{k+1}},\ 1 \leq i,j,k \leq 3 \tag{8.8}$$

Für die Flächen F_i und F gilt gemäß 8.1:

$$\begin{aligned} F_i &= \frac{1}{2}((\vec{M_{i+1}} - \vec{M}) \otimes (\vec{M_{i+2}} - \vec{M})) \\ F &= \frac{1}{2}((\vec{M}_2 - \vec{M}_1) \otimes (\vec{M}_3 - \vec{M}_1)) \end{aligned} \tag{8.9}$$

Mittels Umformung gemäß 8.2 - 8.6 erhält man den Beweis:

$$\begin{aligned} \frac{\sum_{i=1}^{3} F_i \vec{M}_i}{F} &= \frac{\frac{1}{2}\sum_{i=1}^{3}((\vec{M_{i+1}} - \vec{M}) \otimes (\vec{M_{i+2}} - \vec{M}))\vec{M}_i}{\frac{1}{2}((\vec{M}_2 - \vec{M}_1) \otimes (\vec{M}_3 - \vec{M}_1))} \\ &\overset{(8.4)}{=} \frac{\sum_{i=1}^{3}(\vec{M_{i+1}} \otimes \vec{M_{i+2}})\vec{M}_i - \sum_{i=1}^{3}(\vec{M} \otimes \vec{M_{i+2}})\vec{M}_i}{(\vec{M}_2 \otimes \vec{M}_3) - (\vec{M}_1 \otimes \vec{M}_3) - (\vec{M}_2 \otimes \vec{M}_1)} \\ &\quad - \frac{\sum_{i=1}^{3}(\vec{M_{i+1}} \otimes \vec{M})\vec{M}_i}{(\vec{M}_2 \otimes \vec{M}_3) - (\vec{M}_1 \otimes \vec{M}_3) - (\vec{M}_2 \otimes \vec{M}_1)} \\ &\overset{(8.6)}{=} \frac{\sum_{i=1}^{3}(\vec{M_{i+2}} \otimes \vec{M})\vec{M}_i - \sum_{i=1}^{3}(\vec{M}_i \otimes \vec{M})\vec{M_{i+2}}}{\sum_{i=1}^{3}(\vec{M}_i \otimes \vec{M_{i+1}})} \\ &\overset{(8.5)}{=} \frac{\sum_{i=1}^{3}(\vec{M_{i+2}} \otimes \vec{M}_i)\vec{M}}{\sum_{i=1}^{3}(\vec{M_{i+2}} \otimes \vec{M}_i)} \\ &= \vec{M} \end{aligned} \tag{8.10}$$

Seien die zu den Eckpunkten M_1, M_2 und M_3 zugehörigen Meßpunkten mit S_1, S_2 und S_3 und die zu den Eckpunkten entsprechenden Ballradien im Kamerabild mit R_1, R_2 und R_3 bezeichnet. Damit ergibt sich für die Schätzung der Ballposition S auf dem Spielfeld

$$S = \sum_{i=1}^{3} a_i S_i \tag{8.11}$$

und für den entsprechend zu erwartenden Radius R des Balles im Kamerabild:

$$R = \sum_{i=1}^{3} a_i R_i \tag{8.12}$$

Kapitel 9

Kreisfitting

In diesem Kapitel wird der Algorithmus beschrieben, wie zu gegebenen Randpunkten eines Blobs, welcher im Binärbild einen Teil einer Kreisscheibe darstellt, der Mittelpunkt sowie der Radius des entsprechenden Kreises errechnet wird.

9.1 Der Algorithmus

Beim Kreisfitting-Algorithmus wird zu einer gegebenen Menge P von Punkten ein Kreis C gesucht, bei dem die Summe der Abstände zu den Punkten der Menge P minimal ist. Der Algorithmus liefert den Mittelpunkt M_c und den Radius des Kreises C sowie einen Abstandsfehler F_c. Dieser Abstandsfehler F_c ist dabei ein Maß für den durchschnittlichen Abstand der Punkte der Menge P zum Kreis C. Der Mittelpunkt M_c des gesuchten Kreises C sei (x_c, y_c). Der Radius des Kreises sei r_c.

Die Punktmenge P mit n Elementen sei mit $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ bezeichnet. Da geometrisch ein Kreis eindeutig durch 3 auf ihm liegende Punkte bestimmt wird, wird im folgenden davon ausgegangen, daß $n \geq 3$ ist, und nicht alle Punkte aus P auf einer Geraden liegen.

Liegen die Punkte $(x_i, y_i), 1 \leq i \leq n$ auf dem Kreis C, so gilt:

$$(x_i - x_c)^2 + (y_i - y_c)^2 = r_c^2 \tag{9.1}$$

Durch Umformung aus Gleichung 9.1 folgt:

$$(x_i^2 + y_i^2) - 2x_c x_i - 2y_c y_i + (x_c^2 + y_c^2 - r_c^2) = 0 \tag{9.2}$$

In Matrixschreibweise läßt sich das Gleichungssystem 9.2 auf folgende Weise darstellen:

$$M\vec{v} := \begin{pmatrix} (x_1^2 + y_1^2) & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ (x_n^2 + y_n^2) & x_n & y_n & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2x_c \\ -2y_c \\ (x_c^2 + y_c^2 - r_c^2) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \tag{9.3}$$

Sei im folgenden M die oben genannte $n \times 4$ - Matrix. Liegen die Punkte (x_i, y_i) nicht auf einem Kreis, so wird derjenige Kreis C mit Mittelpunkt (x_c, y_c) und Radius r_c gesucht, dessen Summe der Abstände zu den Punkten (x_i, y_i) minimal ist. Mit anderen Worten wird der Vektor $\vec{v} := (v_1\ v_2\ v_3\ v_4)^T$ gesucht, bei dem die Abstandsfehler-Funktion f

$$f(\vec{v}) := \|M\vec{v}\|^2 = \vec{v}^T M^T M \vec{v}, f : R^4 \to R_0^+ \tag{9.4}$$

ein Minimum annimmt.

Ist $\vec{v}$ der gesuchte Lösungsvektor, können die Werte x_c, y_c und r_c aus diesem berechnet werden. Damit ist aber auch ein Vielfaches des Lösungsvektors eine Lösung, und daher auch $\frac{\vec{v}}{\|\vec{v}\|}$. Führen wir somit noch die Funktion

$$g(\vec{v}) := \|\vec{v}\|^2 = \vec{v}^T \vec{v}, g : R^4 \to R_0^+ \tag{9.5}$$

und mit ihr die Nebenbedingung $g(\vec{v}) = 1$ ein, so müssen in dem gesuchten (lokalen) Minimum $\vec{v}$ die Gradienten von f und g parallel sein,[1] d.h. es muß folgende notwendige Bedingung gelten:

$$\nabla f(\vec{v}) = \lambda \nabla g(\vec{v}) \tag{9.6}$$

[1] An einem lokalen Minimum, berühren sich die Funktionen f und g auf der entsprechenden Niveauebenene tangential. Da die Gradienten normal auf der Niveauebene stehen, folgt die Parallelität der Gradienten.

wobei $\lambda \in R$ eine Proportionalitätskonstante ist. Aus den Funktionen f und g sowie einer Hilfsvariablen λ können wir die Lagrange-Funktion erzeugen:

$$L(\vec{v}, \lambda) = f(\vec{v}) + \lambda(1 - g(\vec{v})), L : R^4 \times R \to R \tag{9.7}$$

Die Hilfsvariable λ ist dabei der Lagrange-Multiplikator. Wenn die Nebenbedingung erfüllt ist, stimmen die Funktionen f und L miteinander überein.

Bestimmt man die Extrema der Lagrange-Funktion L, so geschieht dies durch Nullsetzen der ersten partiellen Ableitungen von L nach $\vec{v}$ bzw. nach λ:

$$\begin{aligned} \frac{\partial L}{\partial \vec{v}} &= \frac{\partial f}{\partial \vec{v}} - \lambda \frac{\partial g}{\partial \vec{v}} = 2M^T M\vec{v} - 2\lambda\vec{v} = 0 \\ \frac{\partial L}{\partial \lambda} &= 1 - g(\vec{v}) = 0 \end{aligned} \tag{9.8}$$

Die Extrema der Lagrange-Funktion erfüllen somit die Bedingung 9.6 sowie die Nebenbedingung $g(\vec{v}) = 1$. Aus 9.8 folgt:

$$M^T M\vec{v} = \lambda\vec{v} \tag{9.9}$$

Diese Gleichung ist ein Eigenwert-Problem und läßt sich mittels charakteristischem Polynoms lösen.[2] Die resultierenden Eigenvektoren müssen aufgrund obiger Nebenbedingung $\|\vec{v}\|^2 = 1$ bzw. $\|\vec{v}\| = 1$ ggf. noch normiert werden.

Da die Matrix $M^T M$ eine symetrische 4×4-Matrix[3] ist, sind alle Eigenwerte reell und seien als $\lambda_1, \lambda_2, \lambda_3$ *und* λ_4 bezeichnet. Die zugehörigen Eigenvektoren seien $\vec{v_1}, \vec{v_2}, \vec{v_3}$ *und* $\vec{v_4}$. Aus 9.4 folgt wegen 9.9 und $\|\vec{v_i}\| = 1$ für alle Eigenvektoren $\vec{v_i}$:

$$f(\vec{v_i}) = \vec{v_i}^T M^T M\vec{v_i} = \vec{v_i}^T \lambda_i \vec{v_i} = \lambda_i \|\vec{v_i}\| = \lambda_i \tag{9.10}$$

Wegen $\vec{v_i}^T M^T M\vec{v_i} = \|M\vec{v_i}\|^2 \geq 0$ folgt, daß auch alle Eigenwerte $\lambda_i \geq 0$ sind. Da die Funktion f den zu minimierenden Fehler bezüglich der Abstände der Punkte $(x_i, y_i), 0 < i \leq n$ zum Kreis C darstellt, ist der gesuchte Vektor der zum kleinsten Eigenwert zugehörige Eigenvektor.

[2] λ ist genau dann ein Eigenwert von $M^T M$, falls $det(M^T M - \lambda E)$ gleich Null ist (E stellt hierbei die Einheitsmatrix dar).

[3] Symmetrie ist wegen $(M^T M)^T = M^T M$ gegeben.

Sei o.B.d.A. λ_1 der kleinste Eigenwert. Der dazugehörige Eigenvektor sei $\vec{v_1} = (v_{1_1}\ v_{1_2}\ v_{1_3}\ v_{1_4})^T$. Ist $v_{1_1} \neq 0$ [4] so erhält man den Mittelpunkt (x_c, y_c) und den Radius r_c des gesuchten Kreises durch Auflösen des folgenden Gleichungssystems (siehe auch 9.3):

$$\frac{\vec{v_1}}{v_{1_1}} = \frac{1}{v_{1_1}} \begin{pmatrix} v_{1_1} \\ v_{1_2} \\ v_{1_3} \\ v_{1_4} \end{pmatrix} = \begin{pmatrix} 1 \\ -2x_c \\ -2y_c \\ (x_c^2 + y_c^2 - r_c^2) \end{pmatrix} \tag{9.11}$$

Gilt $v_{1_2}^2 + v_{1_3}^2 - 4v_{1_4}v_{1_1} \geq 0$ folgt aus 9.11:

$$x_c = \frac{-\ v_{1_2}}{2v_{1_1}} \tag{9.12}$$

$$y_c = \frac{-\ v_{1_3}}{2v_{1_1}} \tag{9.13}$$

$$r_c = \frac{\sqrt{v_{1_2}^2 + v_{1_3}^2 - 4v_{1_4}v_{1_1}}}{2v_{1_1}} \tag{9.14}$$

Der so errechnete Mittelpunkt (x_c, y_c) dient zur Positionsbestimmung des Balles auf dem Spielfeld. Die Mittelpunktkoordinaten dienen hierbei als Zeiger in eine Positionstabelle (siehe Abschnitt 7.4.1). Da die Zeiger ganzzahlig sind, werden die Ergebnisse des Kreisfitting-Algorithmus noch auf ganze Zahlen gerundet.

Anstatt über das charakteristische Polynom werden im Saphira-Programm die Eigenwerte mit Hilfe eines sogenannten SVD[5]-Programmes ermittelt, das eine zur Matrix M^TM ähnliche Diagonalmatrix D erzeugt, bei der auf der Hauptdiagonale die Eigenwerte der Matrix M^TM stehen.[6]

---

[4] Liegen die Punkte (x_i, y_i) auf einer Geraden h, so gibt es einen Vektor $\vec{v_1}$ mit $v_{1_1} = 0$, der die Gleichung $M\vec{v_1} = \vec{0}$ erfüllt: Mit $h : y = mx + c$ folgt $M\vec{v_1} = M(0\ \ -m\ 1\ c)^T = \vec{0}$.

[5] SVD = Single Value Decomposition

[6] Aufgrund der Symmetrieeigenschaft sind die vier Eigenvektoren $\vec{v_i}$ der Matrix M^TM zueinander orthogonal. Somit gibt es eine Orthonormalbasis $\{u_1, u_2, u_3, u_4\}$ aus den (normierten) Eigenvektoren. Mittels der Transformationsmatrix $U = (u_1, u_2, u_3, u_4)$ läßt sich die Matrix M^TM diagonalisieren, d.h. $D = U^{-1}M^TMU = U^TM^TMU$. In der Hauptdiagonale der Diagonalmatrix D stehen die Eigenwerte λ_i zu den entsprechenden Eigenvektoren $\vec{v_i}$ bzw. $\vec{u_i}, 1 \leq i \leq 4$.

9.2 Der Abstandsfehler F_c

Bei der Mittelpunktsbestimmung eines Objektes im Binärbild wird der Kreisfitting-Algorithmus auf verschiedene Segmente des Objektes angewandt und deren Ergebnisse verglichen. Um die Qualität der Ergebnisse vergleichen zu können, gibt der Algorithmus einen Abstandsfehler F_c zurück, bei dem die Größe des Segmentes mitberücksichtigt wird. Der Fehler eines Punktes sei dabei ein Maß für den Abstand des Punktes zum Kreis mit Mittelpunkt (x_c, y_c) und Radius r_c. Die Abstandsfehler-Funktion f gebe die Summe der Fehler der n Punkte (x_i, y_i), also den Gesamtfehler λ_1 an. Der Abstandsfehler F_c, der ein Maß für den *durchschnittlichen* Abstand der Punkte zum Kreis C darstellt, ist somit als $\frac{\lambda_1}{n}$ definiert.

Kapitel 10

Diskussion

In diesem Kapitel werden die in dieser Arbeit besprochenen Verfahren diskutiert. Dabei werden diese Verfahren jeweils mit denjenigen Verfahren verglichen, die bei RoboCup '98 eingesetzt worden sind. Anhand des Vergleichs soll dann die Effizienz der jeweiligen Verfahren bestimmt werden.

Zunächst werden die verschiedenen Methoden der Referenzpunktbestimmung analysiert und verglichen. Dann folgt ein Abschnitt über die Blob-Validierung. Anschließend werden die verschiedenen Vision-Informationspakete des *PioneerVision-98*-Programm und des *PioneerVision-99*-Programm besprochen. Das Kapitel endet mit einer Abhandlung spezieller Datenpakete, die im *PioneerVision-99*-Programm eingesetzt werden.

10.1 Referenzpunkt

Anhand des Referenzpunktes wird die relative Position des Balles zum Spieler bestimmt. Bei RoboCup '98 wurde als Referenzpunkt der Schwerpunkt des Blobs gewählt. Bei RoboCup '99 war der Referenzpunkt der Mittelpunkt des Balles im Kamerabild.

Der Schwerpunkt als Referenzpunkt hat aber – wie bereits im Abschnitt 6.2.3 erwähnt – den Nachteil, daß dieser sich bei einer kontinuierlichen Verdeckung des Balles verschiebt. Aber nicht nur Teilverdeckungen des Balles haben eine Verschiebung des Schwerpunktes zur Folge. Auch das Rauschen des Kamerabildes führt zu einer Verschiebung.

Es stellt sich nun die Frage, ob die im RoboCup '99 verwendeten Verfahren zur Mittelpunktsberechnung bei einem verrauschten Bild und einem ruhenden Ball, dessen Verdeckung sich nicht ändert, im Verhältnis zum Schwerpunkt stabilere Werte liefern.

Diese Frage kann nicht eindeutig beantwortet werden, denn die Qualität des jeweiligen Referenzpunktes ist von Position zu Position des Balles im Bild unterschiedlich. Je nach Position ist einmal der Schwerpunkt und das andere Mal der errechnete Mittelpunkt stabiler.

Um dies zu demonstrieren, werden zwei Messreihen exemplarisch vorgestellt. Eine Messreihe besteht dabei aus 2000 aufeinanderfolgenden Messungen. Jede Messung ermittelt pro Binärbild den

- von dem CPU-Board gelieferten Schwerpunkt
- den Mittelpunkt mit Hilfe des Dreiecks-Verfahren und
- den Mittelpunkt mittels Kreisfitting-Verfahren.

Um die Messreihen miteinander vergleichen zu können, werden zu den jeweiligen Punkten die Mittelwerte, die Varianz und die Standardabweichung berechnet. Mit Hilfe der Mittelwerte werden dann Histogramme erstellt, die die quantitive Abweichung der Messungen bezüglich der Mittelwerte veranschaulicht.

10.1.1 Mittelwert und Varianz

Für die Mittelwertberechnung werden die Mittelwerte der X- und Y-Koordinate der jeweiligen Punkte bestimmt. Die Mittelwerte $\tilde{x}$ und $\tilde{y}$ der Koordinaten werden folgendermaßen errechnet:

$$\tilde{x} = \frac{1}{2000} \sum_{i=1}^{2000} x_i \ ; \ \tilde{y} = \frac{1}{2000} \sum_{i=1}^{2000} y_i \tag{10.1}$$

Die x_i und y_i-Werte sind dabei die jeweiligen X- und Y-Koordinaten der einzelnen Meßpunkte.

Mit Hilfe der Mittelwerte $\tilde{x}$ und $\tilde{y}$ wird dann die Varianz V und die Standardabweichung S für jeden Punkt berechnet:

$$V = \frac{1}{1999} \sum_{i=1}^{2000} (\tilde{x} - x_i)^2 + (\tilde{y} - y_i)^2$$

$$S = \sqrt{V} \tag{10.2}$$

10.1.2 Mittelung

Um die Effektivität der Mittelung zu demonstrieren werden auch die entsprechenden Werte für die gemittelten Punkte vorgestellt. Bei der Mittelung wurde über die jeweils letzten 4 Punkte gemittelt. Als Wert für $nomove_{tolerance}$ wurde 1 gewählt. Wie in 7.2 bereits beschrieben, gibt dieser Wert an, bis zu welcher Pixeldistanz des vorangehenden Meßpunktes zum aktuellen Messpunkt die Werte von dem vorhergehenden Messpunkt übernommen werden.

10.1.3 Ergebnisse der Messreihe

Die Bilder 10.1 und 10.2 sind den einzelnen Messreihen entnommen. Die Tabelle 10.1.3 zeigt die entsprechenden Ergebnisse. Bei der ersten Messung sind die Werte für Standardabweichung und Varianz bei den Schwerpunktdaten besser als bei den Kreisfittingdaten. Bei der zweiten Messung ist es genau umgekehrt.

Wie aus der Tabelle entnommen werden kann, fallen bei der Mittelung die Werte für Standardabweichung um ca. $20 - 30$ Prozent geringer aus. Die Mittelung bringt somit eine deutliche Verbesserung für die Stabilität des Referenzpunktes.

Die einzelnen Histogrammdaten der Messreihen sind im Kapitel B des Anhangs aufgelistet.

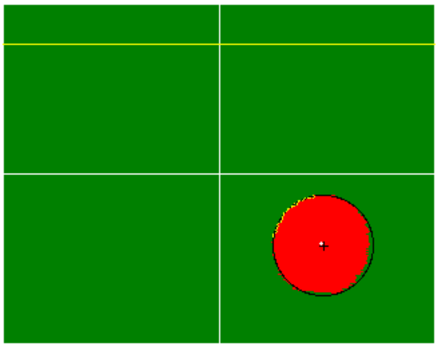

Abbildung 10.1: Das Binärbild zur 1. Messreihe

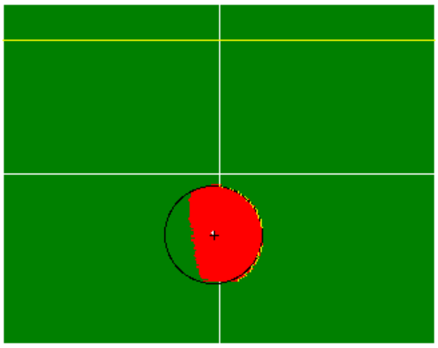

Abbildung 10.2: Das Binärbild zur 2. Messreihe

Messpunkt-Typ	**Standardabweichung**	**Varianz**
1. Messreihe	**(in Pixel)**	
Schwerpunkt	0.68	0.46
Schwerpunkt (gemittelt)	**0.46**	0.21
Dreiecksverfahren	1.38	1.79
Dreiecksverf.(gemittelt)	1.10	1.22
Kreisfitting	1.55	2.39
Kreisfitting (gemittelt)	**1.31**	1.71
2. Messreihe		
Schwerpunkt	0.72	0.52
Schwerpunkt (gemittelt)	**0.57**	0.32
Dreiecksverfahren	0.98	0.97
Dreiecksverf.(gemittelt)	0.73	0.53
Kreisfitting	0.60	0.36
Kreisfitting (gemittelt)	**0.00**	0.00

Tabelle 10.1: Standardabweichung und Varianz aus den Messreihen

10.2 Validierung

Eines der Probleme, die das Bildverarbeitungssystem bei RoboCup '98 dem CS-Freiburg-RoboCup-Team bereitete war die Eigenschaft des *Pioneer-Vision-98*-Programm den Schwerpunkt nur des jeweils *größten* Blobs eines Binärbildes bereitzustellen. Standen Personen an der Bande, deren Kleidung rot woar oder deren Hände auf der Bande lagen, so im Binärbild sehr häufig größere Blobs als der Ball erzeugt. Auch wenn durch verschiedene Plausibilitätstest dieser Blob vom *Saphira-98*-Programm als ungültig hätte erkannt werden können, so blieb der Ball dennoch für den Spieler unsichtbar. Aufgrund der implementierten Validierung mittels oberer Schranke im *Saphira-99*-Programm (siehe Abschnitt 6.2.1) tauchte dieses Problem während der Turniere '99 nicht mehr auf.

Aber auch aufgrund der Validierung nach Form, Größe und Aufenthaltsdauer des Blobs im Binärbild, sowie aufgrund der Auswahl der Blobs mittels dem Kompaktheitskriterium konnte die Anzahl des Vorkommens von *Geisterbällen* gegenüber '98 deutlich gesenkt werden.

10.3 Die Optimierung der Visionpakete

Eine signifikante Veränderung bezüglich der '98-iger Version gab es im Pakethandling. Nicht nur der Datendurchsatz auf der seriellen Verbindung konnte von 9600 Baud auf 38400 Baud erhöht werden, sondern auch ein großer Teil der Paketstrukturen wurde den individuellen Bedürfnissen des CS-Freiburg-RoboCup-Teams angepasst. In erster Linie sind Struktur und Handling des Vision-Informationspaketes vollständig neu konzipiert worden.

Während im *PioneerVision-98*-Programm das Senden des Vision-Informationspaketes mit dem Senden des Motor-Informationspaketes gekoppelt ist, laufen für das Versenden dieser beiden Pakettypen im *Pioneer-Vision-99*-Programm zwei voneinander unabhängige Unterprogramme. Im *Saphira-98*-Programm ist, da die Motor-Informationspakete lediglich mit einer Frequenz von 10 Hz übertragen werden, die Anzahl der empfangenen Pakete auf 10 Stück pro Sekunde beschränkt. Dagegen können im *Pioneer-Vision-99*-Programm bis zu 60 Pakete pro Sekunde gesendet werden.[1]

Da der Saphira-Zyklus ebenfalls mit 10 Hz getaktet ist, stehen im *Saphira-98*-Programm pro Zyklus maximal die Daten aus einem Binärbild zur Auswertung zur Verfügung. Im *Saphira-99*-Programm können dagegen pro Zyklus die Daten aus bis zu sechs Binärbildern ausgewertet werden. Dies bedeutet, daß die Daten eines Zyklus anhand verschiedener Plausibilitätstest gefiltert werden können. Darüberhinaus kann Dank der höheren Übertragungsrate über die Daten eines Zyklus zusätzlich noch gemittelt werden. All dies hat eine im Vergleich zur '98-Version robustere Ballpositionsschätzung zur Folge.

Der Grund, daß die Pakete im *Saphira-99*-Programm mit einer Frequenz bis zu 60 Hz gesendet werden können, liegt nicht nur an der viermal so hohen Baud-Rate, sondern auch daran, daß die Pakete speziell den Bedürfnissen des CS-Freiburg-RoboCup-Teams angepaßt und somit kompakter wurden.

Während das Vision-Informationspaket des *PioneerVision-98*-Programm viele Daten enthält, die für die Bildauswertung des CS-Freiburg-RoboCup-

[1]Werden keine Konturdaten übermittelt, werden alle 60 pro Sekunde generierten Vision-Informationspakete versendet. Im anderen Fall werden die Pakete noch mit einer durchschnittlichen Frequenz von 25-30 Hz übertragen, je nachdem wie groß der Blob im Binärbild ist.

Teams nicht relevant sind, werden im *PioneerVision-99*-Programm nur die für die Ballpositionsschätzung wichtigen Informationen übermittelt. So wird für die Berechnung der Ballposition im *Saphira-98*-Programm lediglich die Schwerpunkt- und Flächeninformation von einem einzigen Farbkanal benutzt. Anstatt den 47 Byte, die das Vision-Informationspaket im *Pioneer-Vision-98*-Programm als Größe besitzt, können im *PioneerVision-99*-Programm die gleichen relevanten Daten mit 12 Bytes versendet werden (siehe Anhang C).[2]

Die Vision-Informationspaketgröße von 47 Byte im *PioneerVision-98*-Programm ist im übrigen zusammen mit der maximalen Übertragungsrate von 9600 Baud ein anderer Grund, warum die Rate der gesendeten Pakete auf 10 Hz begrenzt ist. Infolge der Baudrate von 9600 Baud können – die zwei Steuerbits jeweils mitgerechnet – bei einer fehlerfreien Übertragung maximal nur 960 Bytes pro Sekunde gesendet werden. Da im *PioneerVision-98*-Programm die Größe des Vision- und Motor-Informationspaketes zusammen etwa 100 Bytes beträgt (siehe C.2 und C.5.1), ist dadurch ebenfalls eine obere Schranke für die maximale Anzahl der pro Sekunde übertragbaren Vision-Informationspakete von 10 gegeben.

10.4 Spezielle Datenpakete

Dank des veränderten Konzepts des Pakethandlings im *PioneerVision-99*-Programm und *Saphira-99*-Programm können große Datenmengen sicher zwischen Notebook und CPU-Board verschickt werden. Gehen Teile der Daten bei der Übertragung verloren, werden diese nochmals gesendet.[3]

Dadurch ist es möglich, die gesamte Farbtabelle des Visionboards als Datei auf die Festplatte zu speichern. Da gerade das „Trainieren“ des Bildverarbeitungssystems zu den besonders zeitaufwendigen Tätigkeiten vor einem Spiel gehört, kann man damit die einmal erfassten Trainings-Daten speichern, um sie gegebenenfalls bei nahezu gleichen Beleuchtungsverhältnissen wieder von der Festplatte in die Farbtabelle zu laden.

Noch entscheidender aber ist die Möglichkeit, mittels spezieller Datenpakete ein Abbild des Binärbildes des Visionboards an das *Saphira-99*-Programm auf dem Notebook zu übermitteln. Damit kann das in Anhang D beschriebene Debug-Fenster realisiert werden. Mit Hilfe des Debug-Fensters kann die obere Schranke für die Blob-Validierung den verschiedenen Situationen an-

[2] 6 Byte für Fläche und Schwerpunkt sowie 6 Byte für die Paketbasis.

[3] Die Kontrolle geschieht mit Hilfe sogenannter *Data-Confirmation*-Pakete.

gepasst, können System-Parameter verändert, das Visionboard trainiert und dabei die Ergebnisse der Mittelpunktsberechnung visuell verifiziert werden.

Kapitel 11

Zusammenfassung und Ausblick

11.1 Zusammenfassung

In dieser Arbeit wurde die Ballerkennung und die Ballpositionsschätzung des CS-Freiburg-RoboCup-Teams vorgestellt, die mit dem Cognachrome-Vision-System realisiert wurde.

Zunächst wurden die Bildverarbeitungssysteme einiger anderer Teams besprochen. Dabei wurden die verwendeten Kamerasysteme, die unterschiedlichen Methoden der Farbsegmentierung sowie die der Bildverarbeitung zugrunde liegenden Farbräume beschrieben. Besonders auf die verschiedenen Verfahren der Ballerkennung und Ballpositionsbestimmung wurde eingegangen.

Dann folgte eine Beschreibung der einzelnen Hardwarekomponenten. Insbesondere wurde das vom CS-Freiburg-RoboCup-Team eingesetzte Bildverarbeitungssystem beschrieben. Das System besteht u.a. aus einem Visionboard und einem CPU-Board, auf dem das selbstentwickelte *PioneerVision-99*-Programm läuft. Das Visionboard ist für die Farbsegmentierung des Kamerabildes bestimmt. Dabei kann die bei der Segmentierung verwendete Farbtabelle den individuellen Anforderungen entsprechend angepasst werden.

Die Segmentierung liefert ein Binärbild, das verschiedene Objekte enthalten kann. Anhand logischer Filter werden diejenigen Objekte im Binärbild verworfen, die mit hoher Wahrscheinlichkeit nicht den Ball auf dem Spielfeld repräsentieren. Von den verbliebenen Objekten wird dasjenige mit der besten

Kompaktheit ausgewählt, da der Kreis im zweidimensionalen die kompakteste Form darstellt.

Oftmals stellt das Objekt im Binärbild keine Kreisscheibe dar, dessen Mittelpunkt einfach zu berechnen wäre. In diesem Falle wird das Objekt als Teil einer Kreisscheibe interpretiert. Mittels eines zeiteffizienten Dreiecks-Verfahrens wird der Mittelpunkt des entsprechenden Kreises geschätzt.

Die vom CPU-Board übermittelten Konturdaten werden zunächst gemittelt. Auf der gemittelten Kontur wird anhand des *geschätzten* Mittelpunktes ein Kreis-Fitting-Verfahren angewendet, welches den für die eigentliche Positionsschätzung benötigten Mittelpunkt mit zugehörigem Radius liefert. Ist der Radius und der Mittelpunkt bestimmt, wird anhand einer zuvor generierten Positionstabelle überprüft, ob der ermittelte Radius mit dem erwartenden Radius bezüglich der Mittelpunkt-Position übereinstimmt. Ist dies der Fall, wird die Position des Balles im Spielfeld – ebenfalls mit Hilfe der Positionstabelle – bestimmt. Die Positionstabelle wird durch ein Kalibrierungverfahren initialisiert, bei der zu verschiedenen Ballpositionen die zugehörigen Mittelpunkte und Größen des Balles im Bild errechnet werden.

Die ermittelte Ballposition eines Spielers wird abschließend mit den ermittelten Ballpositionen anderer Spieler abgeglichen. Dies geschieht mit Hilfe eines Zentralrechners, der mit den Spielern kommuniziert. Der Zentralrechner mittelt die Balldaten aller Spieler und stellt diese den einzelnen Robotern wieder zur Verfügung.

11.2 Ausblick

Obwohl die Ballerkennung und Positionsschätzung im Vergleich zur Vorgängerversion in einigen Punkten verbessert werden konnte, gibt es immer noch genügend Bereiche des Bildverarbeitungssystems, die verbesserungswürdig sind. Speziell für den Torwart wäre es sinnvoll, mit zwei oder gar drei anstatt nur mit einer Kamera ausgestattet zu sein. Bisher muß sich der gesamte Roboter drehen, wenn der Ball aus seinem Sichtfeld verschwindet. Dieses Verhalten hat sich während der Spiele öfters als recht ungünstig erwiesen. Da schon mit zwei Kameras ein Sichtfeld von nahezu 200° gegeben ist, würde dies zu einer Verbesserung des Torwartverhaltens beitragen.

Auch die anderen Spieler könnten mit mehreren Kameras, bzw. einer schwenkbaren 360°-Kamera ausgestattet werden. Da die Spieler bei der Ballsuche sich bisher um ihre Achse drehen müssen, bedeuten mehrere Kameras, bzw. eine sich schnell drehende Kamera einen erheblichen Zeitgewinn bei der Ballverfolgung.

Ob nun eine schwenkbare Kamera oder mehrere Kameras eingesetzt werden, ist zunächst eine Preisfrage. Mehrere Kameras mitsamt den entsprechenden Framegrabbern sind in der Anschaffung teurer als eine schwenkbare. Der Nachteil der schwenkbaren Kamera dürfte aber in der Schwierigkeit einer exakten Steuerung der Kameraführung liegen, die aber entscheidend für eine genaue Ballpositionsschätzung ist.

Außer den Kameras sollten auch die übrigen Komponenten des Bildverarbeitungssystem ausgetauscht werden. Sinnvoll wären sogenannte PC-104-Boards mit Framegrabber. Damit wäre eine gezielte Anpassung der Farbsegmentierung möglich. Bisher verläuft das „Trainieren“ der Farben eher nach dem Zufallsprinzip. Ebenso könnte anstatt des RGB- der HSV-Raum als Basis für die Farbsegmentierung benutzt werden. Anhand des HSV-Raums besteht die Möglichkeit auf Basis von Farbnormierung eine gegenüber sich ändernden Beleuchtungsverhältnissen robustere Ballerkennung zu implementieren. Auch bietet ein neues Bildverarbeitungssystem die Möglichkeit, die Farbklassifzierung mittels neuronaler Netze durchzuführen. Somit wäre eine Implementierung einer adaptiven Korrektur des Farbklassifizierers an unterschiedliche Beleuchtungsverhältnisse auf Basis spezieller Lernalgorithmen denkbar.

Auch die mit einem neuen Framegrabber verbundene höhere Auflösung des Kamerabildes bedeutet für die Ballerkennung und Konturenauswertung einen erheblichen Gewinn. Der Ball könnte z.B. viel weiter „gesehen“ werden. Die Mittelpunktschätzung wäre sicherlich noch robuster.

Vielversprechend erscheint auch der Einsatz effizienterer Farbfilter. Bisher wurden mehrere kleine Plastikfolien vor die Linse der Kamera geklebt. Durch Farbspektrumanalyse mit Hilfe eines Spektrometers kann das Farbspektrum des Balles unter verschiedenen Lichtverhältnissen analysiert werden. Auf Basis dieser Analyse kann für jeden Spieler ein spezieller Farbfilter angeschafft werden, der lediglich den gewünschten Farbbereich passieren läßt.

Auch eine aktive Beleuchtung mit Infrarot könnte mit einer entsprechenden Farbspektrumanalyse zu einer effizienten Verbesserung der Ballerkennung führen.

Ein weiterer Punkt wäre die direkte Steuerung des Roboters auf dem CPU-Board. Wenn man aus den Sensordaten direkt geeignete Reaktionen für die Robotersteuerung generieren würde, könnte dies insbesondere für den Torwarts von Vorteil sein. Der „Umweg“ über das Notebook mit den dazugehörigen Rechen- und Latenzzeiten wäre dann überflüssig. Die Reaktionszeit könnte bis zu 0.4 Sekunden reduziert werden, was nicht unerheblich ist, da in dieser Zeit der Ball eine Distanz von ein bis zwei Meter zurückgelegt haben kann. [1]

[1]Bisher werden die Ballinformationen zunächst an das *Saphira-99*-Programm über eine serielle Leitung geschickt. Dieses empfängt und wertet die Information im $\frac{1}{10}$ Sekunden Takt aus und errechnet damit das Verhalten des Roboters. Dann wird der entsprechende Motorsteuer-Befehl über das CPU-Board an den Motorcontroller geschickt. Da die Taktzyklen des Motorcontrollers und des *Saphira-99*-Programms nicht synchron verlaufen, hat dies eine Verzögerung zur Folge. Von der Auswertung des Binärbildes bis zum Empfang des entsprechenden Motorpaketes durch den Motorcontroller können somit bis zu 0.4 Sekunden verstreichen.

Anhang A

Glossar

Bezeichnung	Beschreibung
Binärbild	Wird allen Pixeln eines Kamerabildes der Wert „0“ oder „1“ zugeordnet, spricht man von einem Binärbild. Dabei stellen die Pixel mit dem Wert „0“den Hintergrund und die anderen Pixel den Vordergrund, bzw. ein oder mehrere Objekt dar.
Blob	Blobs sind die in dem Binärbild zusammenhängenden Objekte.
umschreibendes Rechteck	Das umschreibende Rechteck ist das minimale Rechteck, welches einen Blob vollständig einschließt.
Geisterball	Als *Geisterbälle* werden in dieser Arbeit all diejenigen Blobs genannt, die im Binärbild nicht den Ball auf dem Spielfeld repräsentieren.

Anhang B

Histogramme aus den Messreihen

In diesem Kapitel werden die im Kapitel 10 vorgestellten Messreihen explizit in Form von Histogrammen wiedergegeben.

B.1 Die erste Messreihe

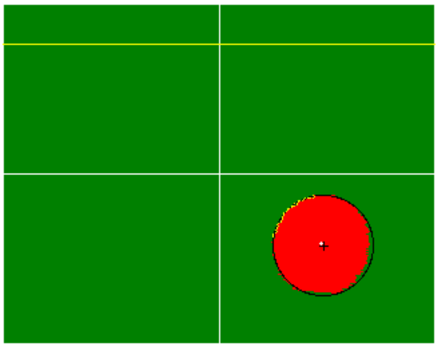

Abbildung B.1: Das Binärbild zur 1. Messreihe

Zunächst wird eine Messreihe gezeigt, bei der die Werte für den Schwerpunkt stabiler sind, als die für den Mittelpunkt aus dem Kreisfitting-Verfahren. Das Bild B.1 stammt aus der ersten Messreihe.

Die Schwerpunkt-Messdaten aus der ersten Messreihe

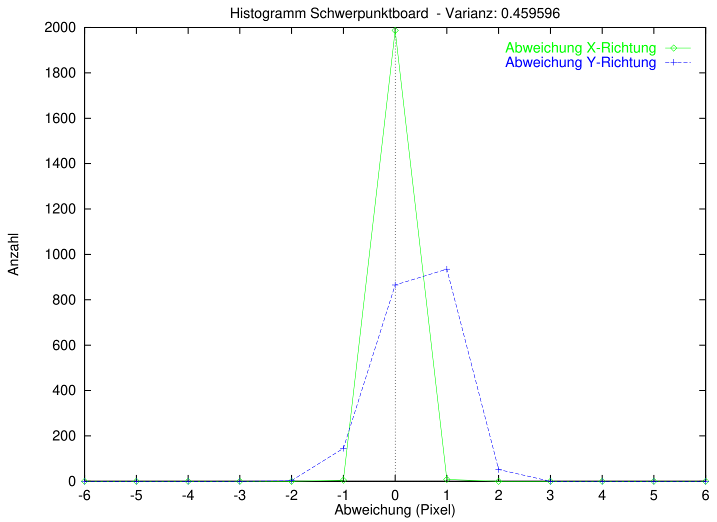

Abbildung B.2: Schwerpunkt-Daten aus der ersten Messreihe; die Standardabweichung beträgt 0.68.

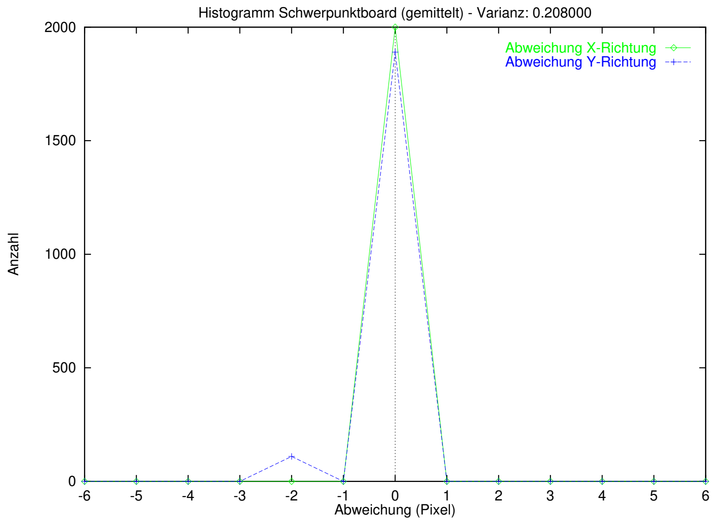

Abbildung B.3: Schwerpunkt-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 0.46.

Die Dreiecksverfahren-Messdaten aus der ersten Messreihe

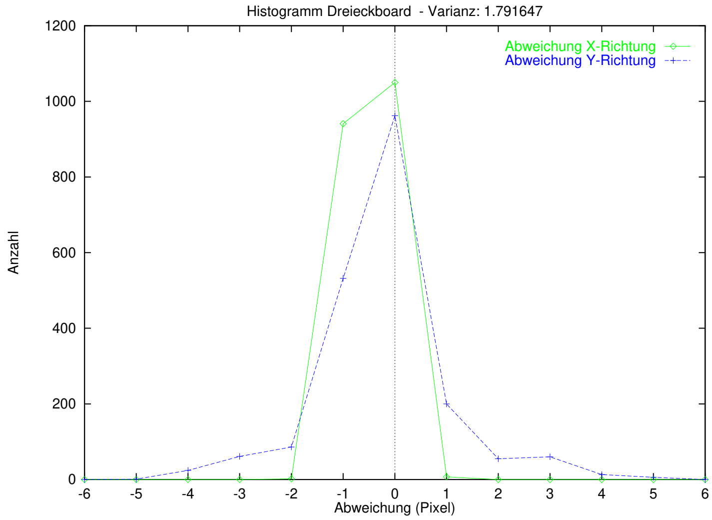

Abbildung B.4: Dreiecksverfahren-Daten aus der ersten Messreihe; die Standardabweichung beträgt 1.38.

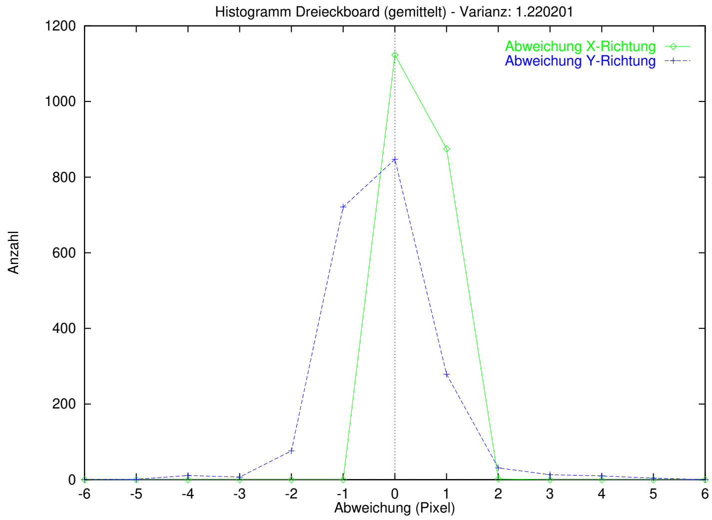

Abbildung B.5: Dreiecksverfahren-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 1.10.

Die Kreisfitting-Messdaten aus der ersten Messreihe

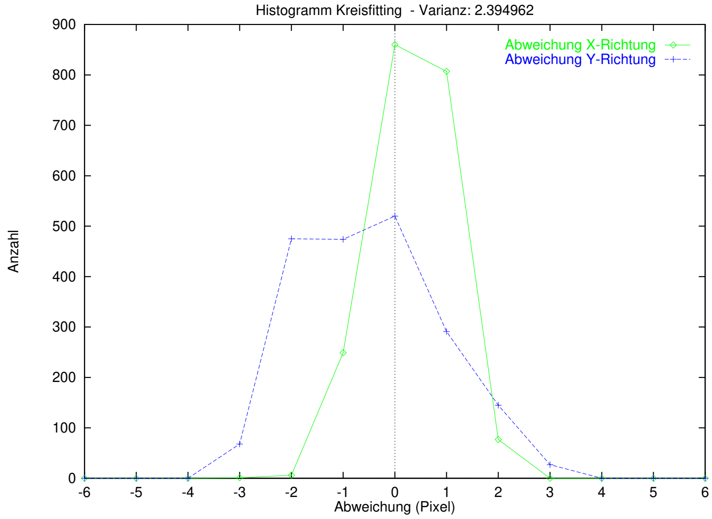

Abbildung B.6: Kreisfitting-Daten aus der ersten Messreihe; die Standardabweichung beträgt 1.55.

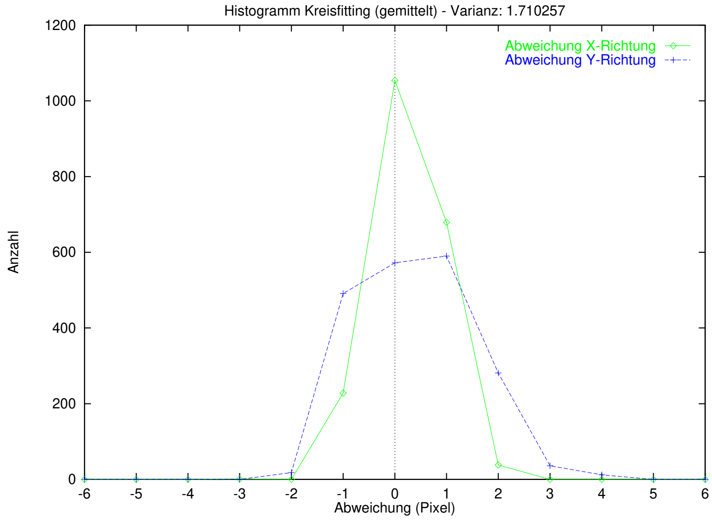

Abbildung B.7: Kreisfitting-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 1.31.

B.2 Die zweite Messreihe

Bei dieser Messreihe sind die Werte für den Mittelpunkt aus dem Kreisfitting-Verfahren stabiler als der Schwerpunkt. Das Bild B.8 stammt aus dieser zweiten Messreihe.

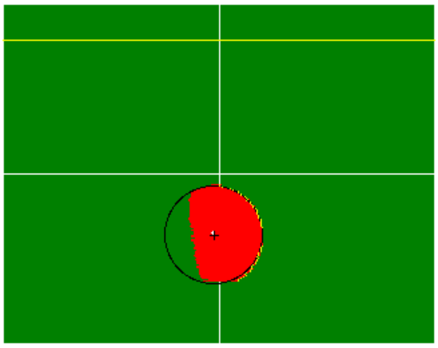

Abbildung B.8: Das Binärbild zur zweiten Messreihe

Die Schwerpunkt-Messdaten aus der zweiten Messreihe

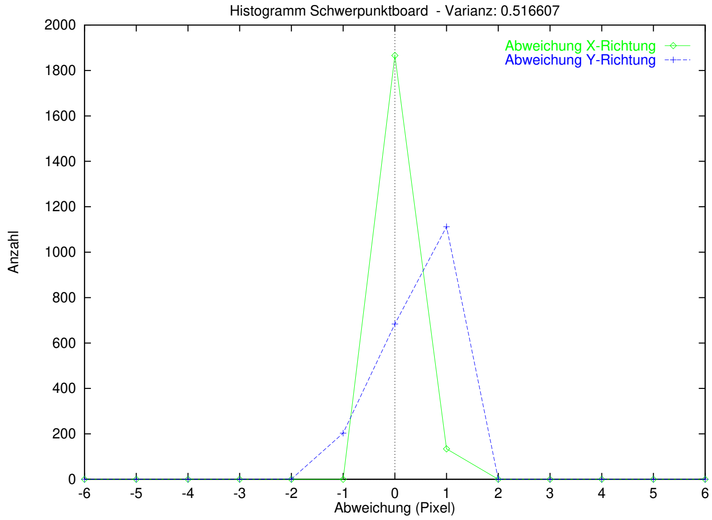

Abbildung B.9: Schwerpunkt-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.72.

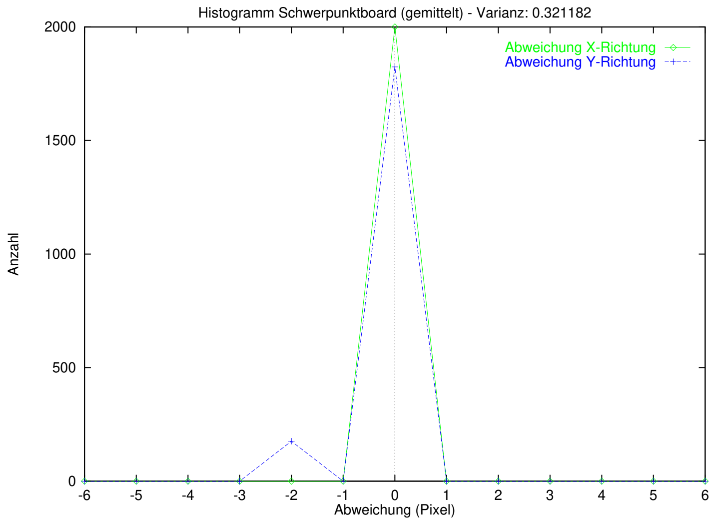

Abbildung B.10: Schwerpunkt-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.57.

Die Dreiecksverfahren-Messdaten aus der zweiten Messreihe

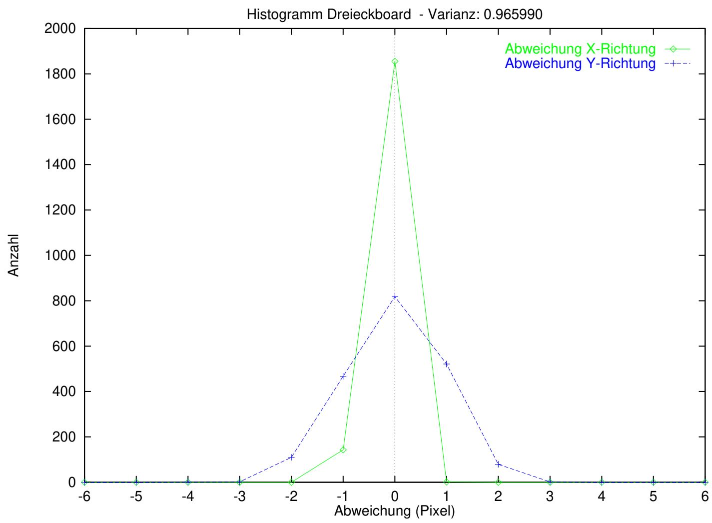

Abbildung B.11: Dreiecksverfahren-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.98.

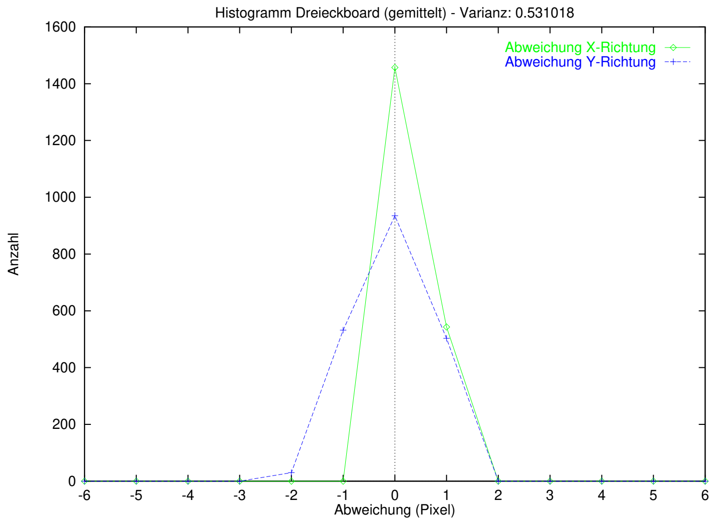

Abbildung B.12: Dreiecksverfahren-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.73.

Die Kreisfitting-Messdaten aus der zweiten Messreihe

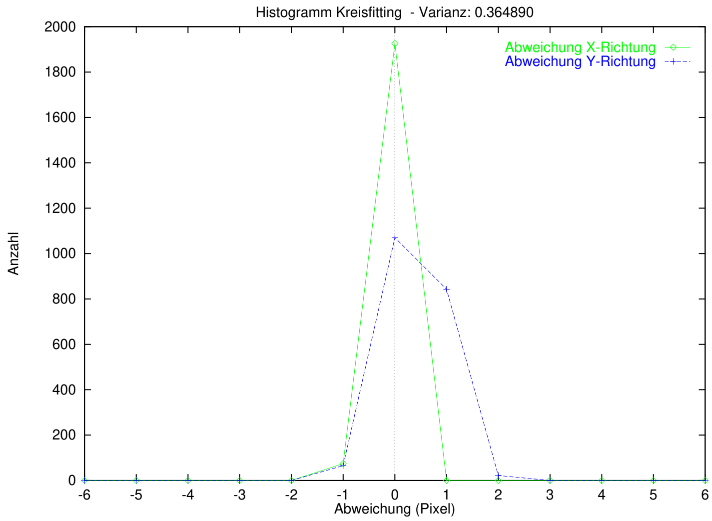

Abbildung B.13: Kreisfitting-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.60.

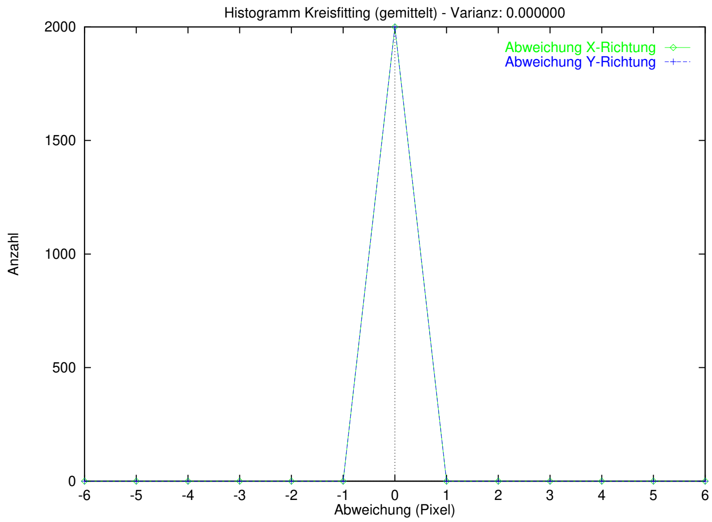

Abbildung B.14: Kreisfitting-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.00.

Anhang C

Die Struktur der Datenpakete

In diesem Kapitel werden die verschiedenen Vision- und Motor-Pakettypen behandelt, die auf der seriellen Leitung zwischen Notebook und CPU-Board übertragen werden. Auf die anderen Pakettypen wird nur kurz eingegangen, da sie für das Verständnis dieser Arbeit nicht notwendig sind.

Zunächst wird die Grundstruktur der verwendeten Datenpakete beschrieben. Diese Grundstruktur gilt nicht nur für die Vision- und Motor-Pakete, sondern für alle Pakete, die über das CPU-Board gesendet werden.

Als nächstes wird die Struktur des Vision-Informationspaketes des *Pioneer-Vision-98*-Programm behandelt, um die Beschränkungen, die durch dieses Programm gegeben sind, aufzuzeigen.

Dann erfolgt eine Beschreibung der Vision-Informationspakete des *Pioneer-Vision-99*-Programmes und anschließend eine Beschreibung des Vision-Steuerpaketes.

Am Schluß werden die Motor-Informations- und die Motor-Steuerpakete vorgestellt.

C.1 Grundstruktur der Datenpakete

Die Kommunikation zwischen Notebook, CPU-Board und Pioneer-Motorcontroller geschieht mit Hilfe von Datenpakete. Die Datenpakete sind dabei unterteilt in solche, die von dem Saphira-Programm auf dem Notebook zur Steuerung des CPU-Board bzw. des Motorcontrollers versendet werden (Vision- und Motor-Steuerungspakete), und solche, die vom CPU-Board und Motorcontroller an das Saphira-Programm geschickt werden und Informationen über das Kamerabild bzw. über den Motorstatus enthalten (Vision- und Motor-Informationspakete).

Die Datenpakete besitzen außer den zu übermittelnden Daten noch zwei Synchronisationsbytes, ein Paketlängenbyte, ein Identifikationsbyte und zwei Prüfsummenbytes. Die Synchronisationsbytes sind für das Lesen der Pakete wichtig, da diese den Anfang eines Daten-Paketes signalisieren. Das Paketlängenbyte gibt die Länge des Paketes an, inklusive des Identifikationsbytes sowie den zwei Prüfsummenbytes aber ohne die zwei Synchronisationsbytes und ohne das Paketlängenbyte. [1] Da die Information der Paketlänge nur in einem einzigen Byte gespeichert wird, ist die maximale Größe der eigentlich zu übertragenden Information somit pro Paket auf 252 Byte beschränkt.[2]

Das Identifikationsbyte gibt den Typ des Paketes an, während die Prüfsummenbytes zur Überprüfung der gesendeten Pakete bestimmt sind. Über die Prüfsumme kann ermittelt werden, ob bei der Übertragung der Paketdaten ein Fehler aufgetreten ist. Die Größe eines Datenpaketes beträgt somit mindestens sechs Bytes.

Die Grundstruktur der verwendeten Datenpakete der Pioneer-Vision-Programme ist in der Tabelle C.1 aufgeführt.

C.2 Das Vision-Informationspaket im *Pioneer-Vision-98*-Programm

Die Bibliothek auf dem CPU-Board stellt eine Funktion zur Verfügung, die von dem größten Blob des jeweiligen Binärbildes u.a. den Schwerpunkt, die Fläche und die Eckpunkte des umschreibenden Rechteckes in einer Struktur speichert. Das *PioneerVision-98*-Programm erzeugt für jeden Farbkanal mittels dieser Funktion eine solche Struktur und speichert daraus bestimmte Werte des größten Blobs in das Vision-Informationspaket. Welche Werte aus der Struktur gespeichert werden, hängt von einem der drei einstellbaren Modi ab: dem Blob-Modus, dem erweiterten Blob-Modus und dem sogenannten

[1] Das Paketlängenbyte gibt genaugenommen die Anzahl der noch nachfolgenden Bytes an.

[2] Gemäß Saphira-Dokumentation wird eine maximale Paket-Größe von 200 Bytes empfohlen [Saphira, 1997].

Größe	Beschreiung
2 Bytes	Synchronisations-Bytes („FAh“ und „FBh“)
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id
N Bytes	spezielle Paketinformation ($N \leq 252$)
2 Bytes	Prüfsumme

Tabelle C.1: Grund-Struktur der Datenpakete des *PioneerVision-98*-Programm und *PioneerVision-99*-Programm

Line-Modus. Für RoboCup '98 war lediglich der erweiterte Blob-Modus von Interesse. Auf die anderen Modi wird daher nicht weiter eingegangen.

Das Vision-Informationspaket enthält neben den Blob-Daten noch einige Statusinformationen, die nur für den *Line*-Modus eine Rolle spielen. Die Größe eines Paketes beträgt 47 Byte. Das von *PioneerVision-98*-Programm erzeugte Vision-Informationspaket hat im erweiterten Blob-Modus das in der Tabelle C.2 wiedergegebene Format.

C.3 Das Vision-Informationspaket im *Pioneer-Vision-99*-Programm

Im *PioneerVision-99*-Programm hängt der Aufbau des Vision-Informationspaketes von einem entsprechenden Modus ab, der mit dem Modus des *PioneerVision-98*-Programm nicht zu verwechseln ist. Die Modi des *PioneerVision-99*-Programm sind der normalen Blob-Modus, der Blob-Outline-Modus und der *Debug*-Blob-Modus.

C.3.1 Der normale Blob-Modus

Im normalen Blob-Modus wird zunächst die Anzahl der in dem Vision-Informationspaket gesendeten Blobs gespeichert. Da die Werte von höchstens einem Blob gesendet werden, kann der Wert für „Anzahl“ nur 0 oder 1 sein.

Größe	Beschreiung
2 Bytes	Synchronisations-Bytes („FAh“ und „FBh“)
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id („50h“)
8 Bytes	Status-Informationen (nur interessant für den *Line*-Modus)
11 Bytes	Blob-Information aus dem 1. Farbkanal
11 Bytes	Blob-Information aus dem 2. Farbkanal
11 Bytes	Blob-Information aus dem 3. Farbkanal
2 Bytes	Prüfsumme

Tabelle C.2: Das Vision-Informationspaket des *PioneerVision-98*-Programm im erweiterten Blob-Modus

Größe	Beschreiung
1 Byte	Modus-Id (beträgt „02h“ für erweiterten Blob-Modus)
4 Bytes	Fläche des größten Blobs
2 Bytes	je ein Byte für die X- und die Y-Koordinate des Schwerpunktes des größten Blobs
2 Bytes	je ein Byte für die X-Werte der linken und rechten Seite des umschreibenden Rechteckes des größten Blobs
2 Bytes	je ein Byte für die Y-Werte der oberen und unteren Seite des umschreibenden Rechteckes des größten Blobs

Tabelle C.3: Die Blob-Informations-Struktur pro Farbkanal des Vision-Informationspaketes ’98

Ist die „Anzahl“ gleich 1 wird die X- und Y-Koordinate des ermittelten Mittelpunktes $\tilde{M}$ sowie der Radius $\tilde{r}$ in diesem Paket gespeichert. Da die Werte für die X- und Y-Koordinate größer als 255 und auch kleiner als 0 sein können,[3] müssen für diese Werte jeweils zwei Byte vorgesehen werden. Die Strukur des Vision-Informationspaketes im normalen Blob-Modus ist in Tabelle C.3.1 abgebildet. Die maximale Paketgröße beträgt demnach 12 Bytes.

Größe	**Beschreiung**
2 Bytes	Synchronisations-Bytes („FAh“ und „FBh“)
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id („E8h“)
1 Byte	Anzahl N der gespeicherten Blobs(0 oder 1)
$N \times 5$ Bytes	jeweils 2 Bytes für die X- und Y-Koordinate des Mittelpunktes $\tilde{M}$ sowie 1 Byte für den Radius $\tilde{r}$
2 Bytes	Prüfsumme

Tabelle C.4: Das Vision-Informationspaket ’99 im normalen Blob-Modus

C.3.2 Der Blob-Outline-Modus

In diesem Modus wird außer dem geschätzten Mittelpunkt $\tilde{M}$ und dem Radius $\tilde{r}$ noch die gesamte Kontur des Blobs gesendet. Dieser Modus wird eingesetzt, wenn mit dem Saphira-Programm eine noch aufwendigere Mittelpunktschätzung mittels Kreisfittingverfahren durchgeführt werden soll.

Die Kontur eines Blobs kann im Binärbild mehrere hundert Pixel betragen. Im extremsten Fall sind dies $2 \times (223 + 254) = 954$ Pixel. Dies tritt dann ein, wenn die Kontur des Blobs identisch mit dem Rand des Kamerabildes ist (siehe Abschnitt 5.2). Möchte man jeweils die X- und Y-Koordinate eines jeden Kontur-Pixels speichern, würden 1908 Bytes benötigt werden. Bei einer Baud-Rate von 38400 Baud könnte in diesem Fall die Kontur-Information von maximal zwei Blobs pro Sekunde an das Notebook gesendet werden. Um diese Rate zu steigern wurde ein spezieller Komprimierungsalgorithmus entwickelt.

[3] Befindet sich der Ball am Rande des Sichtfeldes der Kamera, kann der errechnete Mittelpunkt auch außerhalb des Kamerabildes liegen.

Komprimierung der Kontur-Information

Der Komprimierungsalgorithmus werten den Blob im Binärbild zeilenweise aus. Dabei werden die Zeilen absteigend sortiert. Aufgrund dieser Sortierung genügt es, die Zeilennummer der in der Sortierung ersten Zeile zu speichern. Die Zeilennummer der anderen Zeilen ergeben sich automatisch, da die jeweilige Differenz zu der Zeilennummer der vorangehenden Zeile stets 1 ist.

Anstatt pro Zeile die absoluten X-Koordinaten der Anfangs- und Endpunkte des Blobs in je einem Byte zu speichern, wird versucht, die Differenzen zu den X-Koordinaten der vorangehenden Anfangs- und Endpunkte zu kodieren. Beträgt die absolute Differenz maximal je 7 Pixel, so werden die Differenzen für Anfangs- und Endpunkt in einem einzigen Byte gespeichert. Dabei stellen die oberen vier Bits die Differenz der Anfangs- und die unteren vier Bits die Differenz der Endpunkte dar. Anhand dieser vier Bit werden die Differenzen von -7 („0h“) bis $+7$ („Eh“) dargestellt.

Konkret sieht der Komprimierungsalgorithmus folgendermaßen aus: Pro Zeile werden zwischen einem und vier Bytes gespeichert. Wird nur ein Byte gespeichert, so sind darin die obenbeschriebenen Anfangs- und Endpunkt-Differenzen gespeichert. Die Zeilennummer ergibt sich, indem von der vorangehenden Zeilennummer die Zahl 1 subtrahiert wird.

Wird mehr als ein Byte benutzt, so ist das erste ein Identifiaktionsbyte (ID-Byte), das die Art und Weise der Speicherung der Zeileninformation anzeigt. Das ID-Byte kann die in der Tabelle C.3.2 aufgeführten Werte annehmen.[4]

Der Puffer für die Konturdaten

Die in dem Beispiel oben angegebenen 1908 Bytes werden mit Hilfe des Komprimierungsalgorithmus' auf 257 Bytes[5] reduziert. Es gibt durchaus Blob-Konturen, bei der die aus der Komprimierung resultierende Anzahl mehr als 257 Bytes beträgt.[6]

Da die maximal zu sendende Datenmenge auf insgesamt 252 Bytes pro Paket beschränkt ist (siehe Abschnitt C.1), muß die zu übertragende Information auf mehrere Pakete verteilt werden. Die Komprimierungsdaten werden dabei

[4] Der Komprimierungsalgorithmus sieht die Komprimierung der Kontur-Informationen von mehreren Blobs auf einmal vor. Deshalb existieren spezielle Identifikationsbytes für die Speicherung von einzelnen Zeilennummern.

[5] 4 Bytes für die erste Zeile plus 253×1 Byte für die restlichen Zeilen.

[6] Die maximale Anzahl der Bytes beträgt bei einem 254-Zeilen großen Blob $4+253\times 3 = 763$ Bytes

ID-Byte	**Beschreibung**
„F0h“	Die nächsten 3 Bytes geben die Zeilennummer sowie die absoluten X-Koordinaten der Anfangs- und Endpunkte an.
„F1h“	Die nächsten 2 Bytes geben die Zeilennummer sowie die Anfangs-/Endpunkt-Differenzen zu den vorangehenden Anfangs- und Endpunkten wieder
„F2h“	Die nächsten 2 Bytes geben die absoluten X-Koordinaten der Anfangs- und Endpunkte an. Die Zeilennummer ergibt sich aus der Subtraktion der Zahl 1 von der vorangehenden Zeilennummer.

Tabelle C.5: Die Identifikationsbytes der Komprimierung und ihre Bedeutung

zunächst zusammen mit den ermittelten Mittelpunkt- und Radius-Daten in einem Puffer zwischengespeichert. Abgesehen davon wird noch ein interner Zähler mitgespeichert. Die Pufferstruktur ist in Tabelle C.3.2 wiedergegeben.

Größe	**Beschreiung**
1 Byte	interner Zähler
4 Bytes	X- und Y-Koordinate von $\tilde{M}$
1 Byte	Radius $\tilde{r}$
S-6 Bytes	Konturdaten des Blobs in komprimierter Form

Tabelle C.6: Die Struktur des Konturdaten-Puffers. S gibt die gesamte Größe des Puffers an.

Die Paketstruktur des *HugeData*-Paketes

Zur Übertragung großer Datenmengen wurde für das *PioneerVision-99*-Programm eine spezielle Paketstruktur geschaffen. Das Paket wird im folgenden als *HugeData*-Paket bezeichnet. Die Datenmengen werden hierbei zunächst in einen ausreichend großen Puffer geschrieben. Dann werden die Pufferdaten in kleine Einheiten unterteilt. Jede der Einheiten wird dann in einem *HugeData*-Paket verschickt.

Die *HugeData*-Pakete selbst enhalten zusätzlich zu den Daten noch die Gesamtgröße des Puffers, einen Zeiger auf einen Speicherplatz des Puffers sowie die Anzahl der ab diesem Speicherplatz aus dem Puffer übertragenen Bytes. Der Zeiger auf den ersten Speicherplatz des Puffers hat den Wert 0. Das Format des *HugeData*-Paketes kann aus der Tabelle C.3.2 entnommen werden.

Größe	**Beschreiung**
2 Bytes	Synchronisations-Bytes („FAh“ und „FBh“)
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id
4 Bytes	Gesamtgröße S des zu übermittelnden Puffers
4 Bytes	Zeiger auf einen Speicherplatz des Puffers (zwischen 0 und $S-1$)
1 Bytes	Anzahl N der gespeicherten Bytes des Puffers
N Bytes	Pufferdaten
2 Bytes	Prüfsumme

Tabelle C.7: Die Struktur des *HugeData*-Paketes

Mit Hilfe der *HugeData*-Paketstruktur werden die Vision-Informationen im Blob-Outline-Modus gesendet. Die Paket-Id beträgt in diesem Falle „E7h“.

Aber auch alle anderen Datenstrukturen benutzen dieses Format, wenn diese zu groß sind, um in einem einzigen Paket versendet werden zu können (z.B. das 64×48 große Kamerabild in PGM-Format oder der Inhalt der Farbtabellen des Visionboards).

C.3.3 Der *Debug*-Blob-Modus

Dieser Modus ermöglicht es, Werte von bis zu max_{number} Blobs eines Binärbildes gleichzeitig in einem Paket zu senden. Der maximale Wert für max_{number} ist auf 16 festgesetzt worden.

Zunächst werden die maximal max_{number}-größten Blobs eines Binärbildes der Größe nach absteigend sortiert. Anders als im normalen Blob-Modus wird aber im *Debug*-Blob-Modus weder eine Blob-Validierung noch eine Mittelpunktschätzung gemäß dem *Dreiecksverfahren* durchgeführt. Statt dessen werden von jedem Blob die X- und Y-Koordinaten des Blob-Schwerpunktes bzw. des Zentrums Z des umschreibenden Rechteckes sowie der jeweilige Minimal-Radius $r_{min} = \frac{max_{length}}{2}$[7] im Paket gespeichert. Die maximale Paketgröße beträgt demnach $7 + 3 \times 16 = 55$ Bytes. Die Strukur des Vision-Informationspaketes im *Debug*- Blob-Modus wird in Tabelle C.3.3 abgebildet.

Größe	**Beschreiung**
2 Bytes	Synchronisations-Bytes („FAh“ und „FBh“)
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id („E1h“)
1 Byte	Anzahl N der gespeicherten Blobs ($N <= 16$)
$N \times 3$ Bytes	jeweils 1 Byte für die X- und Y-Koordinate des Blob-Schwerpunktes oder des Zentrums des umschreibenden Rechteckes sowie 1 Byte für den Minimal-Radius r_{min}
2 Bytes	Prüfsumme

Tabelle C.8: Das Vision-Informationspaket ’99 im *Debug*-Blob-Modus

C.4 Das Vision-Steuerungspaket

Zur Steuerung des CPU-Boards werden von dem, auf dem Notebook laufenden, Saphira-Programm sogenannte Vision-Steuerungspakete gesendet.

[7] max_{length} ist die Länge des größten Seite des umschreibenden Rechteckes.

Es könnnen statische (persistente) Variablen des *PioneerVision-99*-Programm modifiziert, spezielle Funktionen auf dem CPU-Board ausgeführt, Statusinformationen abgefragt sowie die Farbtabellen des Visionboards entsprechend angepaßt werden. Das Vision-Steuerungspaket enthält zu diesem Zwecke einen Steuerungsbefehl als nullterminierter String. Die Struktur des Vision-Steuerungspaket ist im *PioneerVision-98*-Programm sowie im *PioneerVision-99*-Programm identisch und wird in Tabelle C.4 wiedergegeben.

Größe	**Beschreiung**
2 Bytes	Synchronisations-Bytes („FAh" und „FBh")
1 Byte	Paket-Längen-Byte
1 Byte	Paket-Id („64h")
1 Byte	Id für String-Daten-Typ („2Bh")
N Bytes	Steuerungsbefehl als nullterminierter String
2 Bytes	Prüfsumme

Tabelle C.9: Das Vision-Steuerungspaket

C.5 Kommunikation mit dem Motorcontroller

C.5.1 Empfangen und Senden von Motor-Informationspaketen

Etwa 10 Mal in der Sekunde wird ein Motor-Informationspaket vom Motorcontroller an das CPU-Board geschickt. Dieses Paket wird vom Pioneer-Vision-Programm an das Notebook weitergesendet. Das Paket enthält Informationen über die Motoren, die das linke und rechte Vorderrad antreiben. Weiterhin enthält es die Odometriedaten[8] und die Sensordaten der Ultra-

[8] Odometrie: Mittels spezieller Sensoren wird die Drehbewegung von jedem der Vorderräder gemessen. Dadurch kann einerseits der zurückgelegte Weg sowie die Richtungsänderung des Roboters ermittelt werden. Die Meßdaten sind jedoch u.a. aufgrund von Gleit- und Rutscheffekten des Roboters meist nicht sehr exakt. Über längere Strecken addieren sich etwaige Meßfehler und liefern dadurch ein ungenaues Ergebnis.

schallsensoren[9] (Sonare), die mit dem Motorcontroller verbunden sind (siehe Abbildung 5.1). Die Größe des Motor-Informationspaketes ist abhängig von der Anzahl der Sonare, die ausgewertet werden. Bei sieben Sonaren beträgt die maximale Größe 53 Bytes. Für eine genaue Beschreibung der Paketstruktur sei auf [ActiveMedia, 1997] verwiesen.

C.5.2 Empfangen und Senden von Motor-Steuerpaketen

Mit einer maximalen Frequenz von 10 Hz sendet das Saphira-Programm des Notebooks Motor-Steuerungspakete an das CPU-Board. Die emfangenen Pakete werden unmittelbar an den Motorcontroller weitergeleitet. Anhand dieser Steuerungspakete wird die Kommunikation zwischen dem Saphira-Programm und dem Motorcontroller auf- bzw. abgebaut, die Geschwindigkeit sowie die Fahrtrichtung des Roboters beeinflußt und die Impulsrate der Ultraschallsensoren modifiziert. Weiterhin können mittels dieser Pakete bestimmte Pins auf dem Motorcontroller, die zur Steuerung speziell angefertigter Geräte (wie etwa der Schußvorrichtung) belegt wurden, angesteuert werden. Für eine genaue Beschreibung der Paketstruktur sei auf das Saphira-Handbuch [Saphira, 1997] verwiesen.

[9] Die Ultraschallsensoren senden Ultraschallsignale aus und empfangen diese wieder. Aus der Zeit zwischen dem Aussenden und Empfangen der Signale kann auf die Entfernung des schallreflektierenden Objektes geschlossen werden. Das mit Ultraschallsensoren verbundene Problem ist, daß aufgrund ihres großen Öffnungswinkels und der Tatsache, daß Signale an glatter Oberfläche abgelenkt werden, die Entfernungsdaten meist nicht sehr genau sind.

Anhang D

Softwarebeschreibung

Dieses Kapitel beschreibt die Anwendung und Einstellung der im Zuge dieser Diplomarbeit erstellten Software.

D.1 Das Debug-Fenster

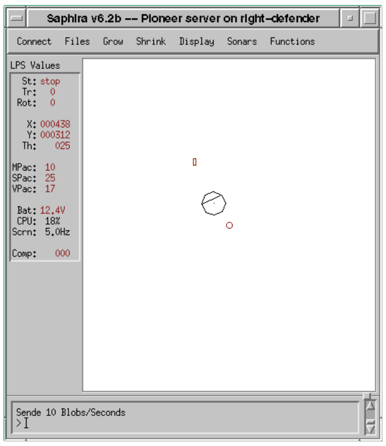

Abbildung D.1: Das Saphira-Fenster

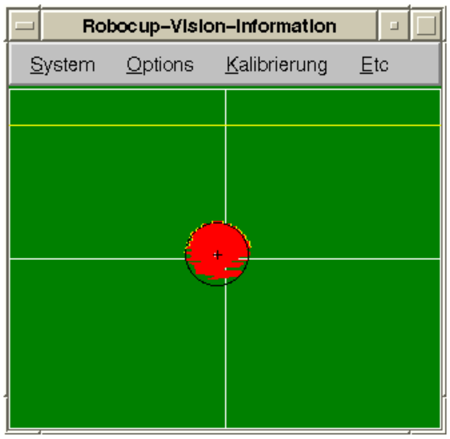

Abbildung D.2: Das Debug-Fenster

Durch die Betätigung der Tastenkombination „@C“ im Saphirafenster wird das Debug-Fenster aktiviert (Siehe Abbildungen D.1 und D.2). In diesem Fenster werden die Blobs des Binärbildes wiedergegeben. Über die Kopfleiste können verschiedene Funktionen aufgerufen bzw. Parameter eingestellt werden.

System

Anhand dieses Untermenüs kann das Debug-Fenster bis zu dreifach vergrößert werden. Weiterhin können alle einstellbaren Werte auf ihren Default-Wert zurückgesetzt werden. Mit „Close" wird das Debug-Fenster geschlossen.

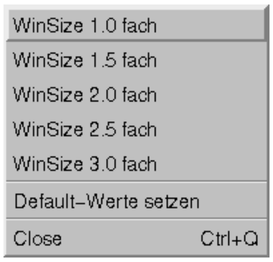

Abbildung D.3: Das System-Untermenü

Options

In diesem Untermenü kann ein Parameter-Fenster aufgerufen werden, mit Hilfe dessen die einstellbaren Werte des *PioneerVision-99*-Programm verändert werden können.

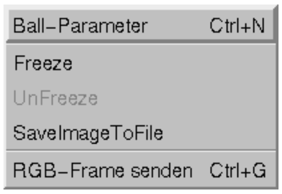

Abbildung D.4: Das Optionen-Untermenü

Weiterhin kann die Aktualisierung des Fensters gestoppt und wieder reaktiviert werden. Sinnvoll ist der Aufruf, wenn man das aktuelle Bild des Debug-Fensters abspeichern möchte. Letzteres wird durch Betätigen des Menüs „SaveImageToFile" realisert. Dabei wird das aktuelle Bild in Bitmap-Form (*.bmp) in eine Datei geschrieben.

Durch Aktivieren von „RGB-Frame senden" wird das aktuelle RGB-Kamerabild mit einer Auflösung von 64×48 in PGM-Form auf die Platte gespeichert und mittels dem Programm „xv" (xview) angezeigt.

Abbildung D.5: Kalibrierungs-Untermenü

Kalibrierung

Mittels dieses Untermenüs wird die Kalibrierung gestartet, bei der zu verschiedenen (zum Roboter relativen) Ballpositionen auf dem Spielfeld die zugehörigen Mittelpunkte und Größen des Balles im Kamerabild gemessen werden. Die gemessenen Daten dienen zur Generierung einer Positionstabelle. Eine genaue Beschreibung des Kalibrierungsablaufs wird im letzten Abschnitt dieses Kapitels gegeben.

Es kann sowohl eine automatische wie auch eine manuelle Kalibrierung gestartet werden. Bei der automatischen Kalibrierung fährt der Roboter vorher definierte Positionen ab und nimmt entsprechende Messungen auf. Bei der manuellen Kalibrierung dagegen muß zu jeder Messung der Roboter an eine gewünschte Position bewegt werden.

Da der Ball sich während des Kalibrierungsvorgangs nicht bewegt, wird die Ballposition einmal vor Beginn der Kalibrierung erfasst. Dabei muß der Ball direkt vor den Roboter gelegt werden.

Eine einmal gestarte Kalibrierung kann jederzeit gestoppt werden. Die bis dahin erfassten Kalibrierungsdaten können in einer Datei gespeichert werden. Möchte man mit Hilfe der manuellen Kalibrierung zusätzliche Meßpunkte aufnehmen, können die Kalibrierungsdaten wieder aus der Datei geladen werden.

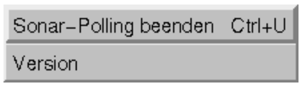

Abbildung D.6: Menü für Sonstiges

Etc

An Hand dieses Untermenüs kann die Versions-Nummer des Programmes abgefragt werden. Auch können die Sonare des Roboters, die mit dem Microcontroller verbunden sind, abgeschaltet werden.

D.2 Parametereinstellungen

D.2.1 Ballerkennung und Kalibrierung

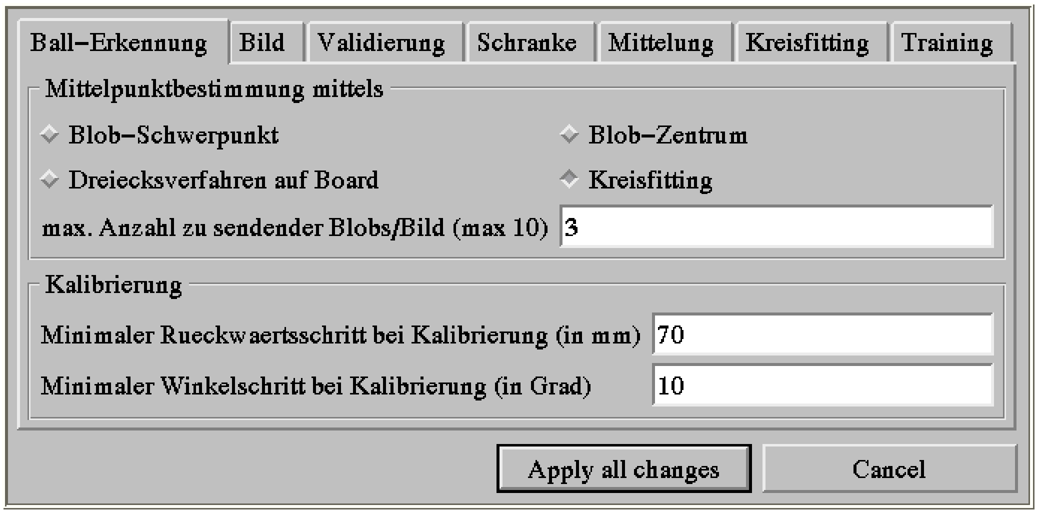

Abbildung D.7: Parametereinstellung für Ballerkennung

Ballerkennungs-Parameter

Anhand eines Referenzpunktes des Balles im Kamerabild wird die Position des Balles geschätzt. Das Programm bietet dabei vier verschiedene Möglichkeiten der Referenzpunktbestimmung an. Der Referenzpunkt des Balles im Kamerabild kann bestimmt werden

- durch den Schwerpunkt des Blobs
- durch das Zentrum des umschreibenden Rechteckes des Blobs
- durch den Mittelpunkt mit Hilfe des Dreiecksverfahren
- durch den Mittelpunkt mittels Kreisfitting-Verfahren

Wurde eines der ersten drei Verfahren gewählt, kann die maximale Anzahl der gesendeten Blob-Mittelpunkte pro Kamerabild angegeben werden. In diesem Falle werden auch die Mittelpunkte derjenigen Blobs gesendet, die als *Geisterbälle* identifiziert worden sind.

Kalibrierungs-Parameter

Bei der Kalibrierung werden die verschiedenen Ballpositionen gemessen. Dies geschieht, indem sich der Roboter von einem zuvor vor ihm gelegten Ball schrittweise entfernt. Nach jedem Rückwärtsschritt R beginnt eine Messreihe, bei dem sich der Roboter dreht. Nach jedem Winkelschritt W wird die Ballposition im Bild mit zugehörigem Radius gemessen. Sowohl die Länge des Rückwärtsschrittes, als auch die Größe des Winkelschrittes kann eingestellt werden.

D.2.2 Kamerabild-Kalibrierung und Debug-Einstellung

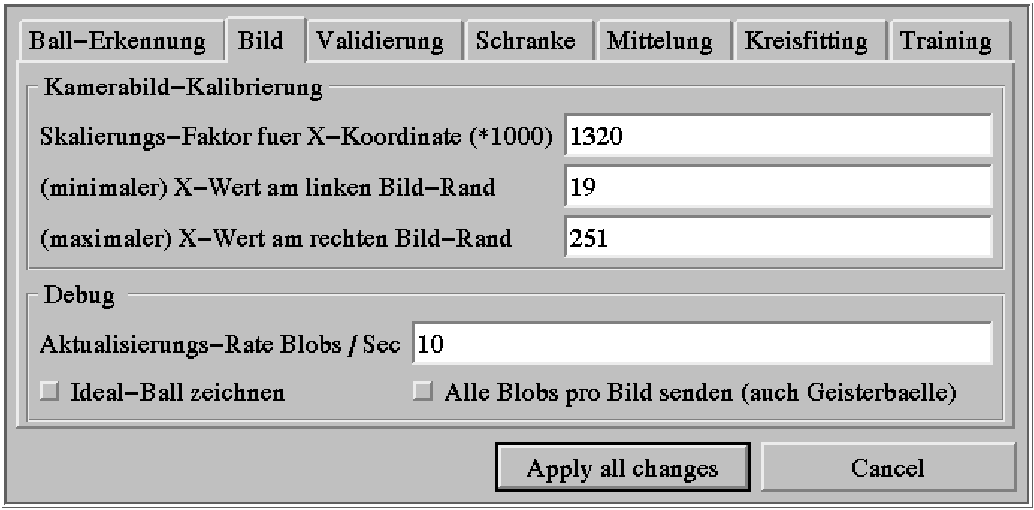

Abbildung D.8: Parametereinstellung für Kamerabild-Kalibrierung und den Debug-Modus

Kamera-Kalibrierung

Das Bildverarbeitungssystem Cognachrome-Vision-System vermindert die Auflösung des Binärbildes und skaliert zusätzlich die X-Koordinate um etwa den Faktor $\frac{3}{4}$. Desweiteren wird das Binärbild in X-Richtung bezüglich des Kamerabildes um ca. 20 Pixel verschoben. Der Nachteil dabei ist, daß einerseits der Ball im Binärbild eine ovale Form aufweist und andererseits der Mittelpunkt der Kameralinse im Binärbild nicht im Mittelpunkt des Bildes liegt. Da eine Konturenauswertung nur auf einem unskalierten Bild sinnvoll ist, wird das Bild mit dem einstellbaren Skalierungsfaktor wieder reskaliert. Die anderen Parameter für die Kamerakalibrierung sind für eine exakte Positionierung des Kameralinsen-Mittelpunktes im Binärbild bestimmt.

Debug-Einstellung

Der Debug-Modus kann im Saphira-Fenster mit der Tastenkombination „@C“ aufgerufen werden. Dabei geht ein sogenanntes Debug-Fenster auf, das die Blobs des Binärbildes wiedergibt. Die Häufigkeit mit der das Fenster in der Sekunde aktualisiert wird, kann eingestellt werden. Ebenso kann angegeben werden, ob nur jeweils der bei der Validierung ausgewählte Blob oder alle im Binärbild vorhandenen Blobs angezeigt werden. Wird nicht nur der ausgewählte Blob angezeigt, so werden die anderen Blobs im Bild gelb gezeichnet. Ein weiterer Parameter gibt an, ob die tatsächliche oder die idealisierte Form des Blobs angezeigt werden soll. Anstatt der Blobs werden dann Kreisscheiben dargestellt.

D.2.3 Parameter für die Blob-Validierung

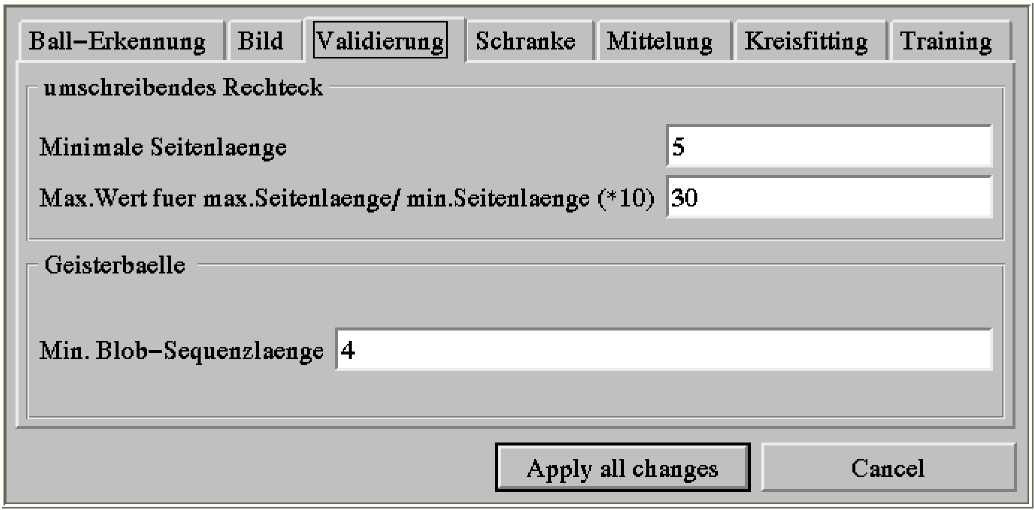

Abbildung D.9: Parametereinstellung für Validierung von Blobs

Bei der Validierung werden Blobs verworfen, die zu klein oder zu schmal sind. Die minimale Seitenlänge des umschreibenden Rechteckes sowie das maximale Verhältnis von $\frac{gr\ddot{o}\beta te Rechteckslänge}{kleinste Rechteckslänge}$ kann eingestellt werden. Weiterhin werden Blobs verworfen, die nur kurzzeitig im Kamerabild auftauchen. Wie lange ein Blob mindestens im Kamerabild vorhanden sein muß, um als Ball erkannt zu werden, wird durch einen weiteren Parameter bestimmt.

D.2.4 Obere Schranke

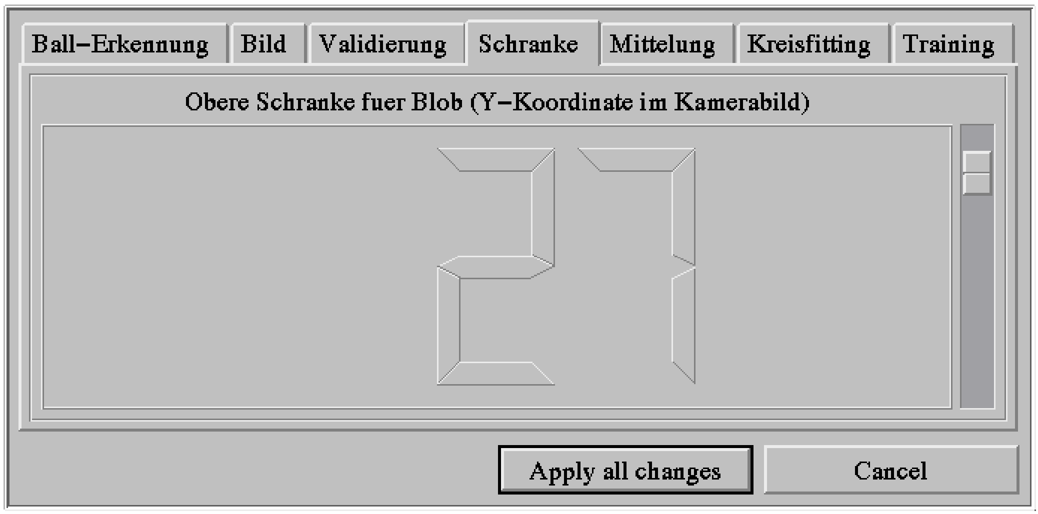

Abbildung D.10: Parametereinstellung für die obere Schranke

Betrachtet man sich den Ball im Debug-Fenster an verschiedenen Positionen im Spielfeld, so gibt es eine obere Grenze im Kamerabild, die der Ball nicht überschreitet. Diese Grenze ist auch davon abhängig, wie weit der Ball gesehen werden soll. Mit Hilfe eines Schiebebalkens kann interaktiv die Schranke im Debug-Fenster verschoben, und somit der entsprechende Wert verändert werden.

D.2.5 Mittelung

Aufgrund des Rauschens des Binärbildes verändert selbst ein ruheliegender Ball im Binärbild stets seine Form. Damit verändert sich nicht nur die Kontur des Blobs sondern auch der jeweils bestimmte Mittelpunkt. Daher kann sowohl über die Kontur wie auch über den Mittelpunkt gemittelt werden.

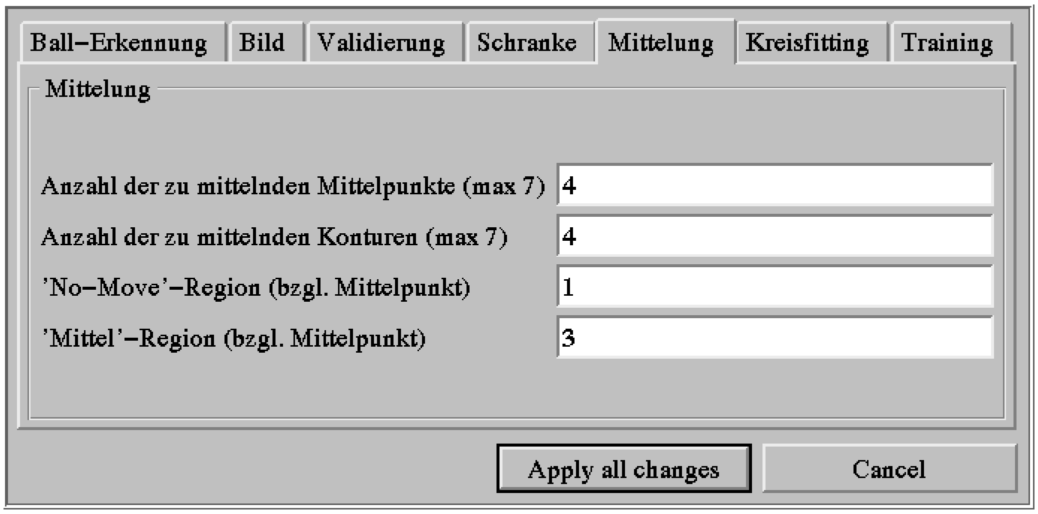

Abbildung D.11: Parametereinstellung für die Mittelung

Die Anzahl der Konturen wie auch die Anzahl der Mittelpunkte, über die gemittelt wird, kann getrennt eingestellt werden. Da aber nicht über einen sich bewegenden Ball gemittelt werden soll, existiert ein zusätzlicher Parameter, der angibt, welche maximale Entfernung (in Pixel) die Mittelpunkte vom aktuellen Mittelpunkt haben dürfen, um in die Mittelung miteinbezogen zu werden.

Ein weiterer Parameter gibt an, bis zu welcher Pixeldifferenz des aktuellen Mittelpunktes zum vorhergehenden Mittelpunkt davon ausgegangen wird, daß der Ball sich nicht bewegt hat. Ist die Pixeldifferenz kleiner gleich dem eingestellten Wert, wird der aus dem vorangehenden Kamerabild errechnete Mittelpunkt als aktueller Mittelpunkt benutzt.

D.2.6 Kreisfitting

In diesem Fenster kann festgelegt werden, welche Größe der bei dem Dreiecksverfahren errechnete Radius mindestens haben muß, damit der Kreisfitting-Algorithmus auf die übermittelte Kontur angewendet wird. Ebenso kann für die einzelnen Kontursegmente eine Mindestanzahl von benötigten Punkten angegeben werden.

In dem *Saphira-99*-Programm wurde ein alternatives Kreisfittingverfahren

Abbildung D.12: Parametereinstellung für den Kreisfitting-Algorithmus

implementiert, bei dem zunächst die oberen und unteren 10 Prozent des Blobs abgeschnitten werden. Anschließend wird der Kreisfitting-Algorithmus auf die verbleibende linke und rechte Kontur des Blobs und dann jeweils getrennt auf die linke wie auch auf die rechte Kontur angewendet. Ein zurückgegebener Fehlerwert gibt die Qualität der jeweiligen Berechnung an. Die Berechnung, die den kleinsten Fehler liefert, dient als Grundlage für die Mittelpunktsbestimmung.

Da der Algorithmus noch nicht ausreichend getestet wurde und auch während der RoboCup-Spiele noch nicht zum Einsatz kam, wurde auf die Beschreibung des Verfahrens in dieser Arbeit verzichtet.

D.2.7 Training der Farbtabelle

Welche Objekte im Binärbild wiedergegeben werden, hängt von einer internen Farbtabelle des Visionboards ab. In der Farbtabelle sind die Werte des Farbbereiches markiert, die im Kamerabild von Interesse sind. Die Farbtabelle kann dabei interaktiv verändert werden, was als „Training“ bezeichnet wird. Für RoboCup wurde das Cognachrome-Vision-System mit der Farbe des Balles „trainiert“.

Beim Trainieren wird der Ball so auf das Spielfeld gelegt, daß dieser in

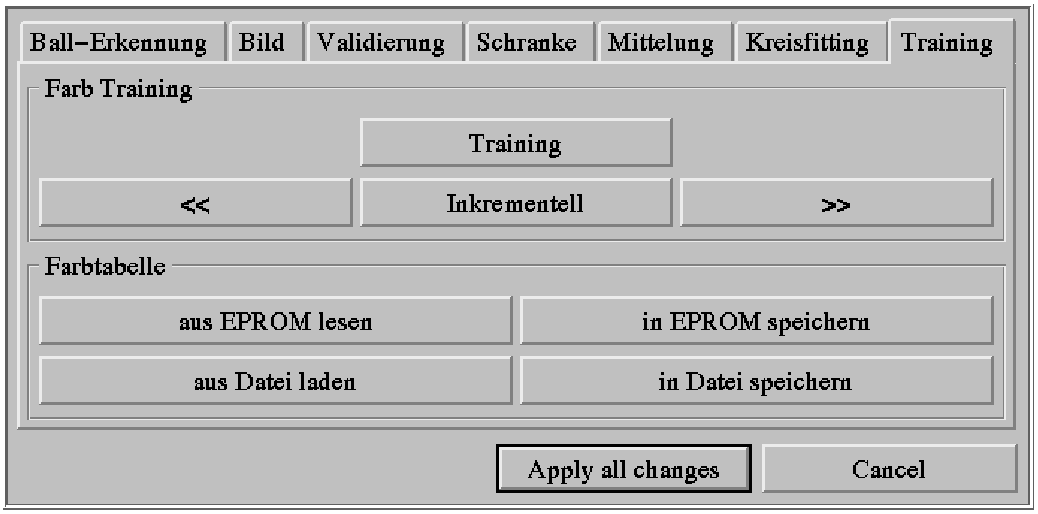

Abbildung D.13: Das Trainieren der Farbtabellen mittels Schaltflächen

der Mitte des Kamerabildes erscheint. Durch Betätigung der Schaltfläche „Training" werden diejenigen Farben des Balles in der Farbtabelle markiert, bei denen der Farbwert sich von dem Farbwert des Pixels im Bildzentrum um maximal einem festgelegten Schwellenwert unterscheidet.

Durch Betätigen der Schaltflächen $\ll$ und $\gg$ kann der obengenannte Schwellenwert erniedrigt, bzw. erhöht werden, was im Binärbild zu einer Blob-Verkleinerung, bzw. Vergrößerung führt.

Ist der Farbbereich des Balles eingestellt, kann der Ball noch auf verschiedene Positionen im Spielfeld platziert werden. Anhand der Schaltfläche „Incrementell" können dann diejenigen Farbwerte, die der Ball an den verschiedenen Positionen im Bild annimmt, in der Farbtabelle zusätzlich noch markiert werden.

Die initialisierte Farbtabelle kann dann ins EPROM oder in eine Datei gespeichert werden. Erfolgt ein „Reset" auf dem CPU-Board, wird diejenige Farbtabelle geladen, die sich im EPROM befindet.

Eine Farbtabelle, die in einer Datei abgespeichert ist, kann geladen und dabei in das EEPROM gespeichert werden. Möchte man während des Trainierens die ursprüngliche Farbtabelle wieder herstellen, muß die Schaltfläche „aus EPROM laden" betätigt werden.

D.3 Die einzelnen Schritte der Kalibrierung

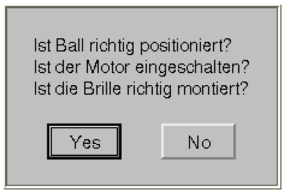

Abbildung D.14: Ballposition o.k.?

Start der Kalibrierung

Beim Start der automatischen Kalibrierung oder bei der Initialisierung der Startposition des Roboters muß der Ball direkt vor den Roboter gelegt werden. Das Programm überprüft anhand der Position des Balles im Kamerabild, ob der Ball tatsächlich direkt vor dem Roboter liegt. Falls nicht, erscheint eine dementsprechende Meldung.

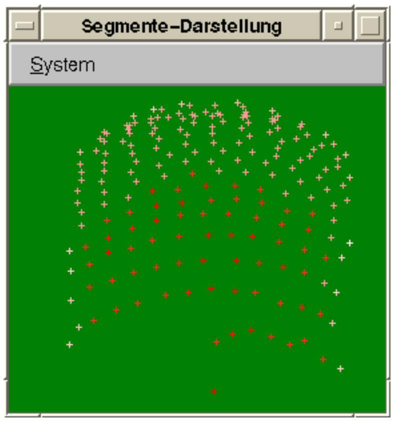

Abbildung D.15: Qualität der aufgenommenen Meßpunkte

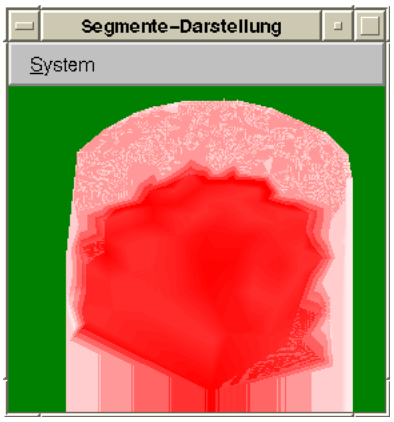

Abbildung D.16: Die Qualität der interpolierten Meßpunkte

Anschließend fährt der Roboter eine vordefinierte Strecke auf dem Spielfeld ab, und mißt dabei verschiedene zum Roboter relative Positionen des Balles. Der jeweils bei einer Messung errechnete Mittelpunkt des Balles im Bild wird dabei in einem Kalibrierungsfenster durch Kreuze dargestellt (siehe Abbildung D.15). Die Farbe des Kreuzes spiegelt dabei die Qualität der Kalibrierung für diese Position wieder. Je dunkler das Rot des Kreuzes ist, desto exakter ist der Mittelpunkt errechnet worden, und desto sicherer kann von diesem Mittelpunkt im Bild auf die Position des Balles im Spielfeld geschlossen werden.

Interpolation

Für die übrigen Bildpunkte, für die keine (relative) Ballposition gemessen wurde, werden die entsprechende Werte interpoliert. Die jeweilige Qualität der interpolierten Kalibrierungswerte ist in der Abbildung D.16 dargestellt.

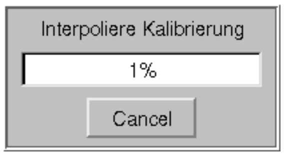

Abbildung D.17: Interpolationsanzeige

Alternative Darstellung

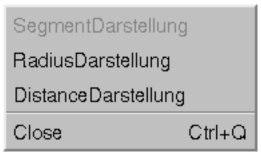

Abbildung D.18: Darstellungsauswahl

Anhand einer alternativen Darstellungsform im Kalibrierungsfenster werden ebenfalls Qualitätskriterien für die Kalibrierung wiedergegeben. Im System-Untermenü des Fensters kann z.B. die sogenannte Radius-Darstellung aktiviert werden. Bei dieser Darstellung wird zu jedem möglichen Ballmittelpunkt im Bild der ermittele Radius als Grauwert dargestellt. Je heller der Farbwert des Pixels ist, desto größer ist der ermittelte Radius (siehe Abbildung D.19). Da mit zunehmender Entfernung des Balles im Spielfeld der entsprechende Radius im Bild kleiner wird, muß ein idealer Meßvorgang in der Radius-Darstellung einen gleichmäßigen Farbverlauf ergeben.

Das Gleiche gilt für die Distanz-Darstellung (siehe Abbildung D.20). Bei der Distanzdarstellung werden zu jedem Ballmittelpunkt im Bild die ermittelten Ballabstände zum Roboter als Grauwert angegeben. Dabei bedeutet ein heller Farbwert eines Pixel eine große Distanz. Ist der Farbwert schwarz, so ist der Ball in nächster Nähe.

Abbildung D.19: ermittelter Ballradius im Bild

Abbildung D.20: ermittelte Ballentfernung

Anhang E

Resultate 1999

In diesem Kapitel werden die einzelnen Spiel-Ergebnisse des CS-Freiburg-RoboCup-Teams bei der RoboCup-Weltmeisterschaft in Stockholm sowie die Spiel-Ergebnisse bei der Deutsche Meisterschaft in Stuttgart in tabellarischer Form wiedergegeben.

E.1 Technical Challenge

Sowohl bei der Deutschen Meisterschaft wie auch bei der Weltmeisterschaft endete das Halbfinalspiel des CS-Freiburg-RoboCup-Teams unentschieden. Das anschließende Elfmeterschießen ging ebenfalls unentschieden aus. Bei der nun nachfolgenden sogenannten „technical challenge“, bei der endgültig der Sieger ermittelt wurde, hatte das CS-Freiburg-RoboCup-Team bei der Weltmeisterschaft Pech, dafür aber bei der Deutschen Meisterschaft das entsprechende Glück.

Bei der „technical challenge“ wird ein Ball in die Mitte des Spielfeldes gelegt. Von nur einer Mannschaft befindet sich ein Feldspieler auf dem Spielfeld. Der Feldspieler wird so plaziert, daß er etwa in der Mitte zwischen Ball und einem vorher festgelegten Tor steht. Dabei wird der Roboter so ausgerichtet, daß seine Fahrtrichtung zum Tor zeigt. Dann wird die Zeit gemessen, die der Roboter benötigt, um den Ball in der Mitte des Spielfeldes in das (leere) Tor zu schießen.

Die „technical challenge“ wird für jede Mannschaft einmal durchgeführt. Anschließend werden die Zeiten verglichen. Wenn die Differenz kleiner als eine Sekunde ist, wird der gesamte Vorgang solange wiederholt, bis die Zeit-

Differenz größer als eine Sekunde ist.[1] Als Sieger geht diejenige Mannschaft hervor, die am schnellsten den Ball ins Tor gebracht hat.

E.2 Resultate

E.2.1 Weltmeisterschaft RoboCup 1999

Runde	Spiel	Ergebnis
Vorrunde:		
	CS-Team – ISocRob	5 : 0
	CS-Team – GMD-Bonn	6 : 0
	CS-Team – KIRC	6 : 0
	CS-Team – RMIT	6 : 0
	CS-Team – Wisley	4 : 0
Viertelfinale:		
	CS-Team – NAIST	4 : 1
Halbfinale:		
	CS-Team – Italien	0 : 0
Spiel um 3.Platz:		
	CS-Team – Wisley	2 : 1

Tabelle E.1: Die Spiele des CS-Freiburg-RoboCup-Teams bei der Weltmeisterschaft 1999

Die Weltmeisterschaft RoboCup 1999 fand vom 28.Juli - 4.August in Stockholm statt. Das CS-Freiburg-RoboCup-Team belegte dabei den dritten Platz.

[1] Bei der ersten „technical challenge" wurde für die Italiener eine Zeit von $19,94$ Sekunden und für das CS-Freiburg-RoboCup-Team eine Zeit von 19.90 Sekunden gemessen.

E.2.2 Deutsche Meisterschaft RoboCup 1999

Runde	Spiel	Ergebnis
Vorrunde:		
	CS-Team – GMD-Bonn	6 : 0
	CS-Team – Tübingen	4 : 0
Halbfinale:		
	CS-Team – München	2 : 2
Finale:		
	CS-Team – Stuttgart	4 : 1

Tabelle E.2: Die Spiele des CS-Freiburg-RoboCup-Teams bei der Deutschen Meisterschaft 1999

Die Deutsche Meisterschaft RoboCup 1999 fand vom 12.Oktober - 13.Oktober in Stuttgart statt. Das CS-Freiburg-RoboCup-Team belegte dabei den ersten Platz.

Abbildungsverzeichnis

1.1 Ein Spiel des CS-Freiburg-RoboCup-Teams 2

2.1 Spieler des „GMD“-Teams mit schwenkbarer Kamera 6

2.2 Omnidirektionale Kamera 7

2.3 Torwart des „Trackies 99“- Teams mit omnidirektionaler Kamera 8

2.4 Der HSV-Farbraum (Hue-Farbton, Saturation-Sättigung, Value-Helligkeit) 10

2.5 Italienischer Torwart mit zwei Kameras 13

3.1 Torwart und zwei Feldspieler des CS-Freiburg-RoboCup-Teams 20

4.1 Digitalisiertes Bild 23

4.2 Segmentiertes Kamerabild 23

4.3 Verschiedene Objekte 24

4.4 Obere Schranke (gelb) 24

4.5 Ausgewähltes Objekt 25

4.6 Geschätzter Mittelpunkt 25

4.7 Gemitteltes Objekt 26

4.8 Kreisfittingverfahren 26

4.9 Mittelpunkt und Radius 27

4.10 Ermittelte Ballposition 27

5.1 Architektur mit Cognachrome-Bildverarbeitungssystem . . . 29

5.2 Verschiedene Grünwerte mit ihren RGB-Werten in hexadezimaler Schreibweise . . . 32
5.3 Kamerabild mit einer Auflösung von 64×48 Pixeln . . . 33
5.4 das entsprechende Binärbild . . . 33
5.5 Glanzlichter und Schatten des Balles im Kamerabild . . . 34

6.1 Die umschreibenden Rechtecke $\vec{A}_1\vec{B}_1\vec{C}_1\vec{D}_1$, $\vec{A}_2\vec{B}_2\vec{C}_2\vec{D}_2$ und $\vec{A}_3\vec{B}_3\vec{C}_3\vec{D}_3$ . . . 40
6.2 Das gleichschenklige Dreieck ABC . . . 43

7.1 Blob1 . . . 49
7.2 Blob2 . . . 49
7.3 Blob3 . . . 49
7.4 Das Kreisfittingverfahren . . . 52
7.5 Das Roboter-Koordinatensystem . . . 55
7.6 Der vom Torwart gesehene Geisterball (grau) wird durch die Fusionierung der Ballpositionsdaten als solcher erkannt und verworfen. Die vom Zentralrechner geschätzte Ballposition ist in der Abbildung weiß dargestellt. . . . 57

8.1 Links: Meßpunkte innerhalb einer vorgegebenen Entfernung und innerhalb des Sichtfeldes der Kamera - Rechts: Triangulation der Meßpunkte . . . 61
8.2 Interpolation mittels der Dreiecksflächen F_1, F_2 und F_3 . . . 64

10.1 Das Binärbild zur 1. Messreihe . . . 73
10.2 Das Binärbild zur 2. Messreihe . . . 73

B.1 Das Binärbild zur 1. Messreihe . . . 83
B.2 Schwerpunkt-Daten aus der ersten Messreihe; die Standardabweichung beträgt 0.68. . . . 84
B.3 Schwerpunkt-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 0.46. . . . 84
B.4 Dreiecksverfahren-Daten aus der ersten Messreihe; die Standardabweichung beträgt 1.38. . . . 85

B.5 Dreiecksverfahren-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 1.10. 85
B.6 Kreisfitting-Daten aus der ersten Messreihe; die Standardabweichung beträgt 1.55. 86
B.7 Kreisfitting-Daten gemittelt aus der ersten Messreihe; die Standardabweichung beträgt 1.31. 86
B.8 Das Binärbild zur zweiten Messreihe 87
B.9 Schwerpunkt-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.72. 87
B.10 Schwerpunkt-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.57. 88
B.11 Dreiecksverfahren-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.98. 88
B.12 Dreiecksverfahren-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.73. 89
B.13 Kreisfitting-Daten aus der zweiten Messreihe; die Standardabweichung beträgt 0.60. 89
B.14 Kreisfitting-Daten gemittelt aus der zweiten Messreihe; die Standardabweichung beträgt 0.00. 90

D.1 Das Saphira-Fenster 102
D.2 Das Debug-Fenster 102
D.3 Das System-Untermenü 103
D.4 Das Optionen-Untermenü 103
D.5 Kalibrierungs-Untermenü 104
D.6 Menü für Sonstiges 104
D.7 Parametereinstellung für Ballerkennung 105
D.8 Parametereinstellung für Kamerabild-Kalibrierung und den Debug-Modus 106
D.9 Parametereinstellung für Validierung von Blobs 107
D.10 Parametereinstellung für die obere Schranke 108
D.11 Parametereinstellung für die Mittelung 109
D.12 Parametereinstellung für den Kreisfitting-Algorithmus 110

D.13 Das Trainieren der Farbtabellen mittels Schaltflächen 111
D.14 Ballposition o.k.? 112
D.15 Qualität der aufgenommenen Meßpunkte 112
D.16 Die Qualität der interpolierten Meßpunkte 112
D.17 Interpolationsanzeige 113
D.18 Darstellungsauswahl 113
D.19 ermittelter Ballradius im Bild 114
D.20 ermittelte Ballentfernung 114

Tabellenverzeichnis

10.1 Standardabweichung und Varianz aus den Messreihen 74

C.1 Grund-Struktur der Datenpakete des *PioneerVision-98*-Programm und *PioneerVision-99*-Programm 93
C.2 Das Vision-Informationspaket des *PioneerVision-98*-Programm im erweiterten Blob-Modus 94
C.3 Die Blob-Informations-Struktur pro Farbkanal des Vision-Informationspaketes ’98 94
C.4 Das Vision-Informationspaket ’99 im normalen Blob-Modus . 95
C.5 Die Identifikationsbytes der Komprimierung und ihre Bedeutung 97
C.6 Die Struktur des Konturdaten-Puffers. S gibt die gesamte Größe des Puffers an. 97
C.7 Die Struktur des *HugeData*-Paketes 98
C.8 Das Vision-Informationspaket ’99 im *Debug*-Blob-Modus . . . 99
C.9 Das Vision-Steuerungspaket 100

E.1 Die Spiele des CS-Freiburg-RoboCup-Teams bei der Weltmeisterschaft 1999 116
E.2 Die Spiele des CS-Freiburg-RoboCup-Teams bei der Deutschen Meisterschaft 1999 117

Literaturverzeichnis

[ActiveMedia, 1996] ActiveMedia INC. *Pioneer 1 Mobile Robot with Saphira Software. Operation Manual.* Version 1, ActiveMedia Incorporated, 1996

[ActiveMedia, 1997] ActiveMedia INC. *Gripper and Experimenter's Module Manual.* Version 1.2, ActiveMedia Incorporated, 1997

[Adobbati *et al.*, 1999] Adobbati, Jay Modi, Gehnam Salemi und Wei-Min Shen. *DREAMTEAM 99: Team Description Paper.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 146-149. Linkoeping University Electronic Press 1999.

[Amoroso *et al.*, 1999] Carmelo Amoroso, Antonio Chella, Vito Morreale und Pietro Storniolo. *A Segmentation System for Soccer Robot Based on Neural Networks.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 5-10. IJCAI Press 1999.

[ARC, 1996] Anne Wright, Randy Sargent und Carl Witty. *ARC Development System User's Guide.* Version 1.4, http://www.newtonlabs.com, 1996

[Bandlow *et al.*, 1999] Thorsten Bandlow, Michael Klupsch, Robert Hanek und Thorsten Schmitt. *Fast Image Segmentation, Object Recognition and Localization in a RoboCup Scenario.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 22-27. IJCAI Press 1999.

[Bonarini, 1999] Andrea Bonarini. *The Body, the Mind or the Eye, first?.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 40-45. IJCAI Press 1999.

[Brusey *et al.*, 1999] James Brusey, Andrew Jennings, Mark Makies, Chris Keen, Anthony Kendall, Lin Padgham und Dhirendra Singh. *RMIT Ro-*

boCup Team. In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 181-188. Linkoeping University Electronic Press 1999.

[Cassinis *et al.*, 1999] Riccardo Cassinis, Alessandro Rizzi. *Design Issues for a Goalkeeper Robot.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 74-79. IJCAI Press 1999.

[Cogna, 1996] Anne Wright, Randy Sargent, Carl Witty und Jeremy Brown. *Cognachrome Vision System User's Guide.* Version 27.0, Newton Research Labs, 1998

[Demura *et al.*, 1999] Kosei Demura, Kenji Miwa, Yasuyuki Asano, Hiroki Igarashi und Daitoshi Ishihara. *Matto: Towards a Pass-Based Tactics.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 163-169. Linkoeping University Electronic Press 1999.

[Dietl, 1999] M. Dietl. *Globale Sensorfusion im Roboterfußball.*. Studienarbeit, Universität Freiburg, 1999.

[Dümler, 1999] B. Dümmler. *Positionsbestimmung mit Laserscannern in RealTime.*. Studienarbeit, Universität Freiburg, 1999.

[Fortune, 1992] Fortune Steven. *VoronoiDiagrams and Delaunay Triangulations, Computing in Euclidean Geometry.* World Scientific, Lecture Notes Series on Computing, Vol. 1, 1992

[Gutmann *et al.*, 1999] Jens-Steffen Gutmann, Thilo Weigel und Bernhard Nebel. *Fast, Accurate, and Robust Self-Localiztion in the RoboCup Environment.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 109-114. IJCAI Press 1999.

[Herrman, 1998] I. Herrman. *Roboter-Schußvorrichtung.* Technischer Bericht, Universität Freiburg, 1998.

[Jamzad, 1999] Mansour Jamzad. *CS-Sharif ROCS99 team in middle-sized robots league.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 118-126. Linkoeping University Electronic Press 1999.

[Levi *et al.*, 1999] Paul Levi, Moritz Schule, Micahel Schanz, Matthias Muscholl, Reinhard Lafrenz, Georg Kindermann, Guenther Hetzel, Thorsten Buchheim, Michael Becht und Norbert Oswald. *CoPS-Team Description.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 195-199. Linkoeping University Electronic Press 1999.

[Nakamura *et al.*, 1999] Takayuki Nakamura, Kazunori Terada, Hideaki Takeda, Akihiro Ebina und Hiromitsu Fujiwana. *Team Description of the RoboCup-NAIST*. In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 170-174. Linkoeping University Electronic Press 1999.

[Padgham *et al.*, 1999] Lin Padgham, James Brusey. *Techniques for obtaining robus, real-time, colour-based vision for robotics.* In *IJCAI-99: The Third International Workshop on RoboCup*, Seiten 63-67. IJCAI Press 1999.

[Piaggio *et al.*, 1999] Maurizio Piaggio, Enrico Pagello, Antonio Chella, Giorgio Clemente, Riccardo Cassinis, Andrea Bonarini, Giovanni Adorni und Daniele Nardi. *Azzurra Robot Team - ART.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 98-106. Linkoeping University Electronic Press 1999.

[Plagge *et al.*, 1999] Michael Plagge, Richard Guenther, Joern Ihlenburg, Dirk Jung und Andreas Zell. *The Attempto RoboCup Robot Team.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 200-209. Linkoeping University Electronic Press 1999.

[Reetz, 1999] C. Reetz. *Aktionsauswahl in dynamischer Umgebung am Beispiel Roboterfußball.*. Diplomarbeit, Universität Freiburg, 1999.

[Sampaio *et al.*, 1999] Sergio Sampaio, Carlos Machado, Ilidio Costa und Antonio Ribeiro. *Patriarcas/MINHO Football Team.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 175-180. Linkoeping University Electronic Press 1999.

[Saphira, 1997] ActiveMedia. *Saphira Software Manual.* Version 6.1, ActiveMedia, 1997

[Schmitt *et al.*, 1999] Thorsten Schmitt, Michael Klupsch, Robert Hanek und Thorsten Bandlow. *Agilo RoboCuppers: RoboCup Team Description.* In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 90-97. Linkoeping University Electronic Press 1999.

[SICK, 1997] SICK. *LMS/LMI 400 Telegram Listing.* Version 02.01 e, SICK, Dezember 1997

[Siggelkow, 1998] S.Siggelkow und H. Burkhardt. *Praktikumsversuch Farbe.* Technischer Bericht, Universität Freiburg, 1998.

[Topor, 1998] A. Topor. *Ballerkennung und Positionsschätzung des CS-Freiburg Teams bei RoboCup '98*. Studienarbeit, Universität Freiburg, 1998.

[Topor, 1999] A. Topor. *Pfadplanung in dynamischer Umgebung.*. Diplomarbeit, Universität Freiburg, 1999.

[Toshiba, 1998] Toshiba. *Libretto Benutzerhandbuch Modell 110CT*. Erste Auflage, Toshiba Corporation, Oktober 1998

[Ventura *et al.*, 1999] Rodrigo Ventura, Pedro Aparicio, Pedro Lima und Luis Custodio. *ISocRob - Intelligent Society of Robots*. In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 150-159. Linkoeping University Electronic Press 1999.

[Weigel, 1999] T. Weigel. *Roboter-Fußball: Selbstlokalisierung, Weltmodellierung, Pfadplannung und verhaltensbasierte Kontrolle*. Diplomarbeit, Universität Freiburg, 1999.

[Yong *et al.*, 1999] FookSeng Yong, BengKiat Ng, KwokMun Loh, EngCheong Ong, Kokkiaw Teo und KiaMeng Loh. *Alpha++*. In *RoboCup-99: Team Descriptions Small and Middle Leagues*, Seiten 107-112. Linkoeping University Electronic Press 1999.

Erklärung

Name: Maximilian Thiel Matr.Nr.: 8149997

Ich erkläre hiermit, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Freiburg, den 15. Dezember 1999