Studienarbeit

Echtzeit-Objekterkennung mit Omnidirektionaler Kamera

Thomas Rückstieß

Betreuer: Dipl.-Technoinform. Patrick Heinemann

beendet am: 17. Juni 2005

Lehrstuhl Rechnerarchitektur
Wilhelm-Schickard Institut für Informatik
Universität Tübingen

Ehrenwörtliche Erklärung

Ich erkläre, dass ich die Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Tübingen, den 22. März 2006 Thomas Rückstieß

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel

Ziel dieser Studienarbeit ist es, eine Echtzeit-Objekterkennung mit einer omnidirektionalen Kamera zu realisieren, die verschiedene Features auf einem Bild findet und extrahiert und deren Positionen in Weltkoordinaten berechnet. Im vorliegenden Fall ist die Umwelt farbig codiert, es kann also davon ausgegangen werden, dass bestimmte Objekte stets die selbe Farbe besitzen. Eingesetzt werden soll dieses System bei mobilen Robotern im RoboCup, die sich autonom lokalisieren und orientieren müssen.

1.2 Übersicht

In diesem Kapitel soll zunächst eine Übersicht über den Einsatz von Sensoren bei mobilen Robotern, insbesondere Kameras gegeben werden, gefolgt von einer kurzen Beschreibung des RoboCups und seiner Bedeutung in der Robotik.

Eine wichtige Aufgabe für alle mobilen Roboter ist es, ein internes Model der unmittelbaren Umwelt aufzustellen, sich darin zu lokalisieren und sichere, kollisionsfreie Pfade um stationäre und bewegliche Hindernisse zu planen und diese abzufahren. Dabei ist es meistens völlig ausreichend, eine zweidimensionale Repräsentation der Umgebung zu erzeugen, die der Ebene des Fußbodens entspricht, auf dem sich der Roboter bewegt. Verschiedenste Sensoren werden eingesetzt, um ein solches Model der Umgebung zu bestimmen: Ultraschallsensoren, Infrarotsensoren, Laserscanner und Kamerasysteme sind einige davon. Die Effizienz dieser Sensoren hängt stark von der vorgegebenen Umwelt ab, in der sich der Roboter bewegen soll. Zu diesen Faktoren gehören unter anderem die Lichtverhältnisse in der Umgebung und die Größe und Materialeigenschaften der zu erkennenden Hindernisse.

Ultraschall- und Infrarotsensoren sind meist nicht genau genug um als Hauptsensoren bei mobilen Robotern eingesetzt zu werden. Sie werden jedoch häufig unterstützend als Sekundärsensoren verwendet, um bestimmte Situationen, wie z.B. Kollisionen, frühzeitig zu erkennen und zu verhindern. Laserscanner liefern unabhängig von vorherrschenden Lichtverhältnissen eine relativ exakte zweidimensionale Repräsentation der Umgebung und würden sich daher sehr gut als Sensor für mobile Roboter einsetzen lassen, wären sie nicht so schwer und teuer. Auch der Energieverbrauch, der mitunter durchaus eine Rolle bei der Wahl des geeigneten Sensors spielen kann, spricht eher gegen den Einsatz von Laserscannern. Der Trend geht mehr und mehr in Richtung Kamerasysteme als Hauptsensor, seien es simple Systeme be-

stehend aus einer Kamera, omnidirektionale Kamerasysteme, die mit Hilfe eines konvexen Spiegels eine Rundumsicht ihrer Umgebung aufnehmen können oder Stereokameras die es sogar erlauben, dreidimensionale Information der Umwelt zu sammeln. Kameras haben zudem den Vorteil, genau wie das menschliche Auge Farbinformation wahrnehmen zu können, was nicht zuletzt zu einem intuitiveren Verständnis und Umgang von Seiten der menschlichen Programmierer und Benutzer solcher Maschinen führt. Umgekehrt ist die Umgebung, in der mobile Roboter eingesetzt werden, oftmals künstlicher Art und entsprechend farbcodiert, was farbempfindlichen Sensoren einen Vorteil gegenüber den oben erwähnten "farbenblinden" Sensoren verschafft. Als Beispiel soll hier der Straßenverkehr angeführt werden: Verkehrsschilder, Ampeln und Fahrbahnmarkierungen erhalten durch ihre Farben eine bestimmte Bedeutung.

Im RoboCup ist die Umgebung relativ genau vorgegeben: konstante Helligkeit über das gesamte Spielfeld[1] und definierte Farben für alle Objekte (schwarze Roboter, blaue und gelbe Tore, orangefarbener Ball, grüner Boden und weiße Linien) sollen es dem Roboter erleichtern, sich zurechtzufinden. In der Tat ist kaum ein Team in der Lage, ohne diese Farbinformationen zu spielen. Zwar gibt es erste Ansätze zur Erkennung von Objekten ohne Farbinformation [1], in der Regel wird aber nach wie vor die Farbe zur Identifikation des Objektes verwendet.

Um Informationen von der Kamera interpretieren und verwenden zu können, um daraus Objekte für das interne Weltmodell zu erstellen, bedarf es einiger Vorverarbeitungsschritte. Zunächst einmal weiß ein Computer (genausowenig wie die Kamera) nichts von *rot* oder *blau*. Unser Kamerasystem liefert ein Bild mit 16 bit Farbtiefe (das entspricht 65536 verschiedenen Farbtönen), die man in die gewünschten Farbklassen einteilen muss. Auf diese sogenannte Farbsegmentierung wird in Kapitel 2.1 nochmal näher eingegangen. Des weiteren liefert eine Kamera stets ein verzerrtes Bild zurück, aus dem man nicht ohne weiteres die tatsächliche Distanz zu einem Objekt auslesen kann. Eine Funktion muss gefunden werden, die Bildkoordinaten in Pixeln auf tatsächliche Weltkoordinaten in Metern transformiert. Abschnitt 2.2 wird darauf eingehen. Im Kapitel 3 werden verschiedene Ansätze zur Erkennung von Objekten vorgestellt und ihre Vor- und Nachteile disskutiert. Im Weiteren wird im Detail beschrieben, wie das letztlich verwendete Verfahren funktioniert und umgesetzt wurde. Die Ergebnisse der Tests und Messreihen mit anschließender Disskussion werden in Kapitel 6 vorgestellt und zu guterletzt ein Ausblick für zukünftige Weiterentwicklung gegeben.

[1] wobei die Regelung bereits gelockert wurde. Lichtschwankungen zwischen 300 und 1200 Lux sind im Reglement vorgesehen.

1.3 RoboCup

RoboCup[2] ist ein internationales Projekt, an dem Universitäten und Firmen verschiedenster Länder und Fachbereiche an der Erforschung und Weiterentwicklung von Künstlicher Intelligenz, Robotik, Sensor- und Bildverarbeitung und verwandter Gebiete teilnehmen. Das oftmals zitierte große Ziel ist es dabei, bis zum Jahr 2050 in der Lage zu sein, mit einer Robotermannschaft die aktuellen menschlichen Fußballweltmeister zu schlagen. Auch wenn viele dieses Ziel als utopisch ansehen, ist es doch Motivation für alle Teilnehmer und ein ständiger Wegweiser für das RoboCup-Komitee. Jedes Jahr verändert dieses Komitee die Umgebung und die Regeln ein Stück in Richtung echtes Fußball und treibt so die Entwicklung immer neuerer und effizienterer Hard- und Softwaretechnologien voran.

Beim RoboCup wird in mehreren Ligen gespielt: In der *SmallSize League* spielen kleine Roboter mit einer Kantenlänge von ca. 10 cm auf einem Feld der Größe 2.8 m × 2.3 m mit einem orangenen Golfball. Kameras über dem Spielfeld überwachen das Spiel und liefern die Bildinformation an einen Zentralrechner, der für das jeweilige Team alle Roboter steuert. Aus diesem Grund wissen alle Roboter stets genau, wo sich Ball, Gegner und vor allem die eigenen Spieler befinden, was zu sehr schnellen, exakten Bewegungen der Roboter führt. Die SmallSize League ist technisch gesehen momentan nahezu ausgereizt und es wird nun versucht, den Entwicklern durch schwachere Beleuchtung und anderen Handicaps einen neuen Anreiz zu verschaffen. Die *MiddleSize League*, momentan noch die „Königsklasse“ beim RoboCup, spielt im Gegenatz zur SmallSize League mit völlig autonomen Robotern, die zwar untereinander kommunizieren dürfen, jedoch selbst Information mittels geeigneter Sensoren sammeln und die richtigen Entscheidungen treffen müssen. Gespielt wird mit 4 bis 7 Robotern, abhängig von der Größe der Roboter, auf einem Feld von ca. 8 m × 12 m. Es gibt keine Bandenbegrenzung, der Ball kann also das Spielfeld, dass mit weißen Linien auf grünem Grund markiert ist, verlassen und wird dann vom Schiedsrichter zurück auf die Auslinie gelegt. Die MiddleSize League ist noch lange nicht an ihre Grenzen gestoßen, Kooperation der Roboter (z.B. Pass-Spiel), das Spielen ohne farbige Markierungen und genaue globale Lokalisierung auf dem Spielfeld sind nur einige der Themen, an denen aktuell gearbeitet wird und bei denen noch viel Verbesserungspotential vorhanden ist. Weitere Ligen, wie z.B. die *Simulation League*, bei der überhaupt keine Roboter zum Einsatz kommen sondern lediglich Software-Agenten in einer sehr detailreich simulierten Umgebung gegeneinander antreten, oder die *4-Legged League*, bei der die Aibo Robo-

[2]Siehe auch `http://www.robocup.org/`

terhunde[3] von Sony um den Ball kämpfen, werden ebenfalls beim RoboCup ausgetragen. Seit kurzem existiert auch eine *Humanoid League*, bei der bisher aber noch nicht richtig Fußball gespielt wird, da die Technik noch nicht weit genug ausgereift ist um Spieldynamik aufkommen zu lassen. In Zukunft, so ist es geplant, sollen die humanoiden Roboter jedoch die MiddleSize League ablösen und irgendwann gegen menschliche Gegner ihr Können unter Beweis stellen.

1.4 Das „Attempto Tübingen“ Robot Soccer Team

Das Team „Attempto Tübingen“ des Wilhelm-Schickard-Instituts für Informatik der Universität Tübingen nimmt schon seit vielen Jahren in der MiddleSize League an deutschen und internationalen Wettkämpfen teil. Die größten Erfolge waren der 2. Platz bei der Weltmeisterschaft in Paris 1998, damals noch unter dem Namen „T-Team“ und der kürzlich errungene Weltmeistertitel in der *Technical Challenge* 2003 in Padova, Italien. Die Technical Challenge ist ein weiterer Wettbewerb des RoboCup, bei dem es weniger um das Fußballspielen ansich, sondern mehr um technische Neuerungen, Ball-Handling und Kooperation geht. Die Roboter müssen einige vorgegebene Aufgaben erfüllen und anschließend kann das Team – sozusagen als Kür – eine besondere Eigenschaft oder Fähigkeit seiner Roboter präsentieren. Unser Team hat dort zum ersten Mal gezeigt, dass es in der Lage ist, einen ganz normalen schwarz-weißen Fußball stabil zu verfolgen, sodass es nicht mehr auf die orangene Signalfarbe angewiesen ist [1].

Über die Jahre hinweg hat sich die Hardware und Software unserer Roboter ständig verändert, neuere Sensoren wurden integriert, die Roboterbasis wurde gewechselt und der Schwerpunkt hat sich vom Multisensorsystem auf einen Ansatz mit Kameras als Hauptsensor verlagert. Bis Ende des Jahres wollen wir unter Beibehaltung der Software die etwas veraltete Hardware der Spieler erneut komplett wechseln und auf kleinere, schnellere, ommnidirektional[4] fahrende Roboter umsteigen. Wir erhoffen uns dadurch, technisch mit anderen Teams wieder vorne mitspielen zu können und zudem den ersten Schritt zu kooperativem Teamspiel zu gehen, für das schnelle, agile Roboter notwendig sind. Auch ein elektrisch angesteuerter Kickmechanismus, der

[3] Unter `http://www.sony.net/Products/aibo/` gibt es mehr Information über dieses High-Tech Spielzeug.

[4] Im Gegensatz zu den bisherigen Robotern, die über einen Differentialantrieb mit zwei einzeln ansteuerbaren Rädern links und rechts und einer Stützrolle am hinteren Ende des Roboters verfügen, werden die neuen, dreieckig geformten, Roboter mit jeweils einer speziellen Rolle an jedem Eck in der Lage sein, aus ihrem Ausgangspunkt heraus in jede beliebige Richtung zu fahren und sich dabei unabhängig zu drehen.

einen Eisenkolben dosiert durch eine Spule nach vorne beschleunigt, soll den momentan verwendeten Pneumatik-Kicker ablösen. Bei den Sensoren wollen wir unser Konzept der Bildverabeitung beibehalten und sogar noch weiter verstärken und bewusst auf andere Sensoren, wie etwa dem Laserscanner, verzichten.

1.5 Das omnidirektionale Kamerasystem

Unsere Hauptsensoren sind seit einigen Jahren Kameras, von denen aktuell jeweils zwei pro Roboter zum Einsatz kommen. Eine Kamera ist nach vorne ausgerichtet und wird ausschließlich zur Lokalisierung des Balls verwendet. Die zweite Kamera nimmt ein 360°-Bild der Umgebung auf. Information über die Position des Balls und der Objekte, der Feldmarkierungen und der Tore werden aus diesem Bild extrahiert. Unser omnidirektionales Kamerasystem besteht aus einer Siemens SICOLOR C810 CCD-DSP Farbkamera mit einem 1/3" CCD-Chip, der eine Auflösung von 752×582 Pixeln liefert. Die Kamera überträgt ein standardisiertes PAL-Signal mit 50 Halbbildern pro Sekunde. Die 4.3 f Linse der Kamera zeigt senkrecht nach oben auf einen hyperbolischen Spiegel, der so konzipiert wurde, dass er die gesamte Umgebung in jede Richtung bis zum Horizont abbilden kann. Abbildung 1 zeigt eine solche Aufnahme.

2 Farb- und Distanzkalibrierung

2.1 Farbkalibrierung

Bei der Farbkalibrierung geht es in erster Linie darum, die vielen verschiedenen Farbtöne eines Farbraums in einige wenige Farbklassen wie *orange* oder *schwarz* einzuteilen. Wie bereits in der Einleitung erwähnt, ist die Umgebung beim RoboCup sorgfältig definiert und ermöglicht so eine Objekterkennung mittels Farben. Diese wurden so gewählt, dass sie sich hinsichtlich ihrer Position in gängigen Farbräumen deutlich unterscheiden und eine Farbsegmentierung erlauben.

Es gibt verschiedene Farbraum-Modelle, die alle unterschiedliche Vor- und Nachteile mit sich bringen. Im Prinzip tragen jedoch alle Farbmodelle die selbe Information und je nach Art der Weiterverarbeitung eignet sich ein Modell besser als das andere. Die drei wichtigsten sollen hier kurz erwähnt werden: Beim RGB-Farbraum werden nach dem additiven Farbmischprinzip auf den Achsen die Rot-, Grün- und Blau-Anteile aufgetragen. Er liegt dem natürlichen Farbensehen beim Menschen zugrunde, da das Auge drei

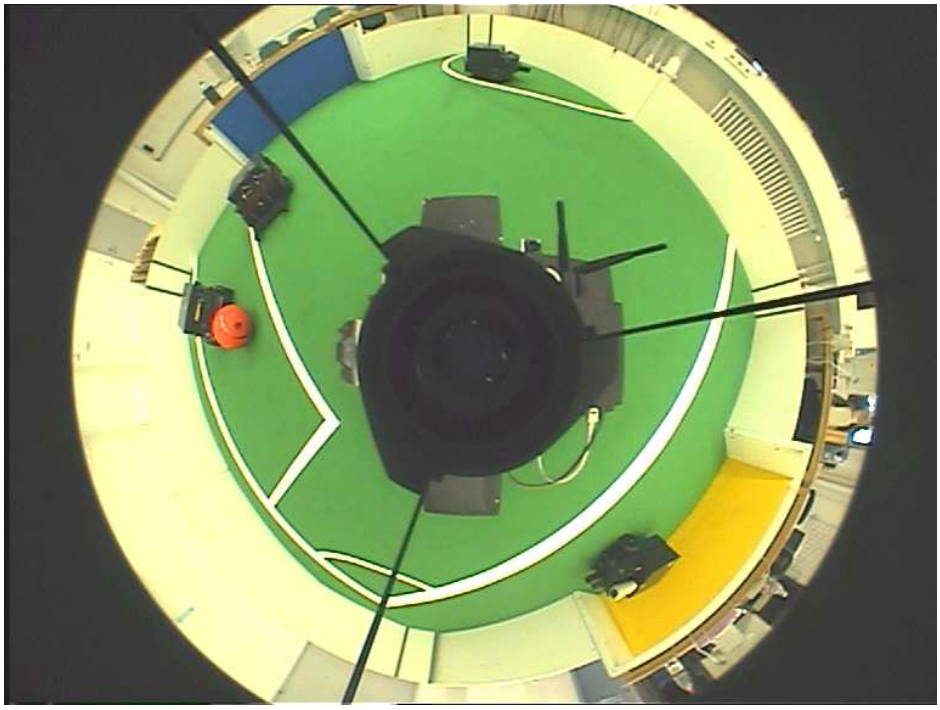

Abbildung 1: Dies ist ein typisches Bild einer omnidirektionalen Kamera. Leider ist es mit dieser Form des Spiegels nicht vermeidbar, dass ein großer Teil der Kameralinse im Bild mitabgebildet wird. Der Winkel vom Roboter zu einem Objekt bleibt im Bild erhalten, die Strecken verkürzen sich nach außen stark.

verschiedene Rezeptorzellen (Zapfen) für die jeweiligen Farben besitzt. Im Unterschied dazu trägt der HSI-Farbraum auf seinen Achsen den Farbton (Hue), die Sättigung der Farbe (Saturation) und die Farbintensität (Intensity) auf. Hierbei ist zu beachten dass der Farbton keine lineare sondern eine radiale Achse besitzt. Es handelt sich hier also eher um einen Kegel als einen Kubus wie bei den anderen beiden Farbräumen. Man spricht auch oftmals vom *Farbwinkel*, wenn man den Farbton meint. Der dritte Farbraum YUV enthält Informationen über die Helligkeit und die Farbchrominanz, letztere besteht wieder aus zwei Komponenten. Die Entwicklung des YUV Modells geht auf die Entwicklung des Farbfernsehens (PAL) zurück, bei der nach einer Möglichkeit gesucht wurde, eine Abwärtskompatibilität des Fernsehsignals zu alten Schwarz/Weiß-Fernsehgeräten zu ermöglichen.

Mit unserer Kalibrierungssoftware ist es möglich, alle 3 Farbraummodelle zu verwenden, gute Erfahrungen haben wir mit dem YUV-Modell gemacht. Hier lassen sich die für uns relevanten Farben besonders gut trennen. Der Vorgang der Trennung der Farben im Farbraum wird auch Segmentierung

genannt. Unterschiedliche Ansätze sind dabei möglich: Gewisse Farbklassen können durch geometrische Körper wie Würfel oder Kugeln im Farbraum abgegrenzt werden. Alle Farben, die innerhalb einer dieser Formen liegen, gehören dann zur selben Farbklasse. Je nach Farbraum bieten sich unterschiedliche Formen an. Anstatt eine Klasse komplett abzugrenzen, könnte man sie auch in nur zwei Farbdimensionen beschränken. Soll zum Beispiel die Helligkeit einer Farbe vernachlässigt werden, so könnte man die Farbklassen im HSI-Model beschreiben und lediglich Farbton und Farbsättigung beschränken. Um etwas spezifischere Farbklassen definieren zu können, wäre auch vorstellbar, mehrere nicht-zusammenhängende (disjunkte) Bereiche zu einer Farbklasse zu gruppieren. Die allgemeinste Farbeinteilung erhält man schließlich, wenn jeder einzelne Farbton einer Klasse zugeordnet werden kann. Hierbei ist jedoch anzumerken, dass dies unter Umständen recht viel Speicher belegen kann. Im obigen Beispiel müssten wir bei 8 bit Farbtiefe pro Farbkanal und weiteren 8 bit zur Speicherung der Farbklasse bereits 16,7 MB Speicher allokieren. Dies ist auf heutigen Home-PCs vielleicht nicht viel, bei Systemen die am RoboCup teilnehmen kann dies jedoch bereits zu Speicherknappheit führen.

Glücklicherweise sind wir mit unserem System in der Lage, eine Lookup-Tabelle für alle Farben anzulegen und zu verarbeiten. Dies liegt daran, dass unser Framegrabber nur 2^5 Farben darstellen kann, was sich zwar negativ auf die Farbauflösung auswirkt, uns aber im Gegenzug ermöglicht, jede der 32768 Farben einzeln zu einer Farbklasse zuzuordnen.

Wir können also eine Funktion f_c aufstellen, die jede Farbe (definiert durch die 3 Farbwerte c_1, c_2, c_3 in einem Farbraum R) zu einer Farbklasse C zuordnet:

$$f_c : \mathbb{Z}^3 \mapsto \mathbb{N}, \;\; (c_1, c_2, c_3) \rightarrow C \tag{1}$$

Als Farbkalibrierung bezeichnen wir den Vorgang der Einteilung aller Farben in Farbklassen. Dies geschieht in unserem System momentan noch von Hand, an automatischen Kalibrierungsmethoden durch Clustering wird jedoch zur Zeit gearbeitet. Zur Kalibirierung haben wir eine Software entwickelt, die es uns erlaubt, auf einem von der Kamera gelieferten Bild verschiedene Bereiche mit einer Art Pinsel zu maskieren – ähnlich wie es bei Grafikprogrammen wie Adobe Photoshop üblich ist. Farben, die von dieser Maske verdeckt werden, können dann auf Knopfdruck zu einer der Farbklassen addiert oder davon subtrahiert werden. Die Pinselgröße ist dabei variabel einstellbar. Zusätzlich können einzelne Farben oder Farbquader in einer dreidimensionalen Ansicht des Farbraums explizit zu einer Farbklasse hinzugefügt oder von ihr entfernt werden. Dieses Tool ermöglicht uns, bei

nahezu allen Lichtbedingungen in kurzer Zeit (ca. 5 Minuten) alle 8 von uns zum Spielen benötigten Farbklassen (*orange*, *schwarz*, *grün*, *weiß*, *blau*, *gelb*, *cyan* und *magenta*) so zu kalibrieren, dass eine Erkennung von Linien und verschiedenen Objekten im Bild möglich ist.

2.2 Distanzkalibrierung

Über die Farben lassen sich zwar Objekte im Bild identifizieren, will man aber ein Weltmodell für Lokalisierung und Pfadplanung aufstellen, so muss aus der Position im Bild eine Distanz zum Roboter in Weltkoordinaten errechnet werden. Es ist also eine Funktion f_d nötig, die die Abbildung von Pixel- nach Weltkoordinaten vornimmt.

Wüsste man die exakte Form des Spiegels und die Parameter des Objektivs, so könnte man ein physikalisches Modell des Kamerasystems aufstellen und damit die Koordinaten[5] im Bild auf Koordinaten in der Umgebung abbilden. Leider kennen wir die Kameraparameter und die Funktion nicht, die die Form unseres hyperbolischen Spiegels beschreibt, deshalb bleibt uns nur eine Kalibrierung „von Hand“. Um diesen Vorgang zu vereinfachen, gehen wir von einem rotationsinvarianten Kamerasystem aus. Dies bedeutet, dass die optische Achse der Kamera, die Symmetrieachse des Spiegels und die Drehachse des Roboters identisch sein müssen. Dies ist zwar meistens nicht exakt erfüllt, die Unterschiede sind aber so gering, dass sie für unsere Zwecke vernachlässigt werden können. Unter dieser Vorraussetzung können wir nun davon ausgehen, dass unser Bild winkeltreu ist, also der Winkel ϕ_p in Bildkoordinaten mit dem Winkel ϕ_w in Weltkoordinaten übereinstimmt. Somit ist unsere Abbildungsfunktion nur noch eindimensional und sieht folgendermaßen aus:

$$f_d : \mathbb{Z} \mapsto \mathbb{R}, \;\; r_p \rightarrow r_w \tag{2}$$

Um die Funktion f_d bestimmen zu können, müssen wir einige Bilder von genau ausgemessenen Punkten in der realen Welt machen und in diesen dann den Abstand zum Kamera-Mittelpunkt in Pixeln bestimmen. Durch die wenigen, so erhaltenen Stützpunkte legen wir eine stetige Funktion (in unserem Fall ein Spline) und interpolieren so die dazwischenliegenden Abstände mittels eines *Least Squares* Fittings (siehe Abbildung 2).

Bisher wurde stets von einer zweidimensionalen Umgebung des Roboters ausgegangen, offensichtlich befinden sich die Roboter aber in einer dreidimen-

[5]Falls nicht anders angegeben, werden im Folgenden Koordinaten stets als Polarkoordinaten (r, ϕ) relativ zum Kameramittelpunkt interpretiert

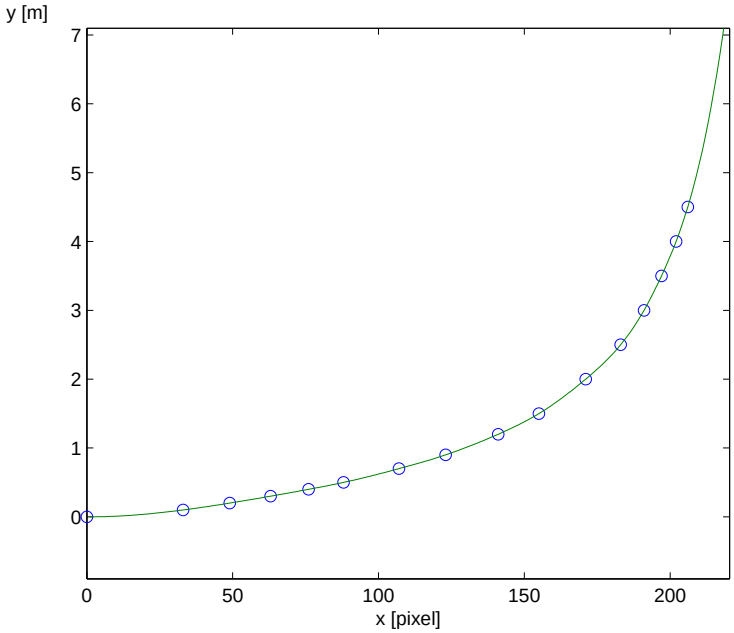

Abbildung 2: Abbildungsfunktion f_d von Kamerakoordinaten (x-Achse) nach Weltkoordinaten (y-Achse). Die Kreise zeigen die gemessenen Stützpunkte für die Interpolation an.

sionalen Welt, auch wenn sie sich nur in zwei Translations-Freiheitsgraden bewegen. Dass die Objekte dennoch dreidimensional sind, muss auf jeden Fall Berücksichtigung finden. In Abbildung 3 sieht man in Seitenansicht einen Roboter mit omnidirektionalem Kamerasystem und ein würfelförmiges Objekt. Zwei Sehstrahlen sind examplarisch eingezeichnet: Sehstrahl 1 zielt von der Kameralinse über den hyperbolischen Spiegel auf die vordere untere Ecke A des Objekts, während Sehstrahl 2 auf die hintere (dem Roboter abgewandte) obere Ecke B respektive auf einen Punkt C auf dem Fußboden hinter dem Objekt zielt. Im Fall 1 kann der Abstand des Objektes zum Roboter durch Einsetzen des Pixelabstandes f_d eindeutig bestimmt werden. Im Fall 2 hingegen ist dies nicht mehr möglich. Da sich Punkt B in einer anderen Höhe befindet als Punkte A und C, dies aber von unserer Abbildungsfunktion f_d nicht berücksichtig wird, bildet sie den Punkt fälschlicherweiße auf die Distanz von C ab, also zu weit nach hinten. Die beiden Punkte B und C in Weltkoordinaten entsprechen ein und dem selben Punkt in Bildkoordinaten. Eine Bijektivität von f_d ist nur dann gegeben, wenn man lediglich Weltkoordinatenpunkte verwendet, die auf einer Ebene liegen, in diesem Fall die des Fußbodens. Da beim RoboCup Objekte bei Schrägansicht Teile des

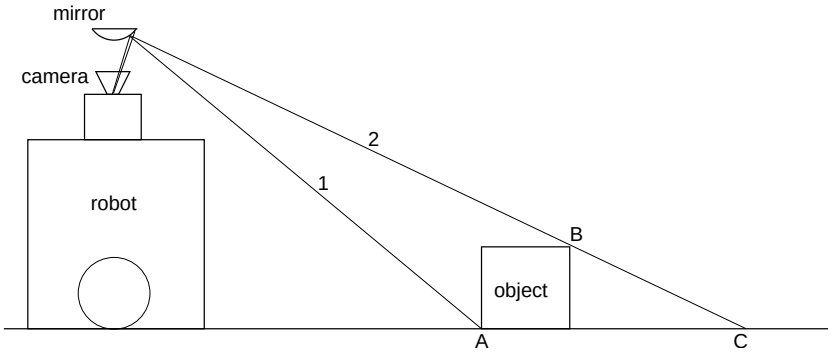

Abbildung 3: Eine eindeutige Bestimmung der Position eines Objektes in Weltkoordinaten ist nur bei Punkten möglich, die auf der selben Ebene liegen. Für die Abbildungen durch den parabolischen Spiegel ist diese Vorraussetzung lediglich für die am Nächsten zum Bildmittelpunkt liegenden Punkte möglich (Strahl 1). Würde man hingegen Strahl 2 auf die Welt abbilden, käme es zu großen Diskrepanzen zwischen vermuteter und tatsächlicher Position des Objekts.

Fußbodens verdecken können, lässt sich dieser Fehler nur mit einem Trick umgehen: Um die tatsächliche Distanz zu einem Objekt zu bestimmen, wird stets der am Nächsten zum Roboter liegende Punkt des Objekts (im Falle von Abbildung 3 Punkt A) zur Abstandsmessung verwendet. Dadurch kann man sicher sein, einen Punkt auf der Ebene des Fußbodens erwischt zu haben.

3 Objekterkennung im Kamerabild

3.1 Existierende Verfahren

In [2] werden die bekannten Methoden zur Objekterkennung in drei verschiedene Kategorien eingeteilt:

1. Kantenbasierte Methoden
2. Regionenbasierte Methoden
3. Methoden, die sowohl kanten- als auch regionenbasiert arbeiten

Kantenbasierte Methoden haben oftmals das Problem, dass die Kantenstruktur des zu erkennenden Objektes oder zumindest dessen perspektivische Abbildung nicht bekannt ist. Zudem können die Bildauflösung, ungünstige Lichtverhältnisse, Schatten, Texturen und Verdeckungen das Kantenprofil beinflussen und verändern und die Kantenerkennung schwierig oder sogar unmöglich machen.

Regionenbasierte Methoden zur Objekterkennung gehen von einer Homogenität der äußeren Erscheinung eines Objektes aus, zum Beispiel einer konstanten Farbe oder Textur. Zwar ist dies in natürlichen Umgebungen oftmals nicht der Fall, im RoboCup wurde diese Vorraussetzung jedoch bewusst erfüllt. Objekte des selben Typs (z.B. Roboter) haben alle die selbe Farbe (schwarz) und können mit regionenbasierten Verfahren aus dem Bild extrahiert werden. Die in dieser Arbeit vorgestellte Methode zählt ebenfalls zu dieser Gruppe.

Nur wenige Arbeitsgruppen im RoboCup arbeiten momentan mit kantenbasierten Methoden. Zum einen ist die Umgebung streng definiert und farblich codiert, was die Verwendung von regionenbasierten Methoden forciert, zum anderen unterscheidet sich die Form der Roboter der einzelnen Teams recht stark, was eine Erkennung eines allgemeinen Roboterprofils erschwert. Einige Ansätze zur farbunabhängigen Detektion des Balls sind jedoch bekannt (siehe [1], hier wird sowohl die Form als auch das typische schwarzweiße Muster des Balles zur Erkennung verwendet, es handelt sich also um eine Methode des 3. Typs). Doch selbst bei regionsbasierten Methoden gibt es große Unterschiede in der Art und Weise der Erkennung. Zwei typische Varianten der Objekterkennung sollen hier stellvertretend vorgestellt werden.

Radiale Suche

Bei dieser Art von Objekterkennung mittels omnidirektionaler Kamera wird ausgehend von der Mitte des Bildes strahlenförmig nach außen nach Besonderheiten im Bild (z.B. Farbübergängen oder größeren Flächen der gewünschten Farbe) gesucht. Vorteile der Methode sind eine gute Abdeckung in der Nähe des Roboters und die Tatsache, dass naheliegende Objekte zuerst gefunden werden. Oftmals genügt es, das nächstliegende Objekt oder Hindernis in einer gegebenen Richtung zu finden. Ein weiterer Vorteil ist die lineare Laufzeit des Algorithmus, abhängig von der Gesamtlänge der Strahlen. In [3] wird eine Variante dieses Prinzips für Freiraumdetektion vorgestellt.

Nachteilig bei der radialen Suche ist, dass die Auflösung der Strahlen im äußeren Teil des Bildes immer geringer wird (da der Winkel konstant bleibt, wächst der Abstand zwischen zwei Strahlen nach außen hin an). Bedingt durch die Form des Spiegels eines omnidirektionalen Kamerasystems (meist konisch oder hyperbolisch) wird die Abbildung der Welt im äußeren Bereich des Bildes ebenfalls schlechter. Diese beiden Effekte verstärken sich gegenseitig und machen eine Erkennung von Objekten, die sich nicht unmittelbar neben dem Roboter befinden, sehr ungenau.

Flood-Fill Algorithmus

Beim Flood-Fill Algorithmus werden einige Pixel im Bild (entweder zufällig oder in gewissen Abständen) ausgewählt und nach gewissen Features hin untersucht. Entspricht ein Pixel dem Profil des zu erkennenden Objektes (z.B. die richtige Farbe), so wird in dessen lokalen Umgebung nach weiteren Pixeln gesucht, die ebenfalls dem Profil entsprechen. Der Flood-Fill Algorithmus geht dabei rekursiv vor, und „füllt“ den gesamten Bereich des vorliegenden Objektes auf. Das Resultat kann eine Bounding-Box des erkannten Bereichs oder – falls dies nicht ausreicht – eine genaue Beschreibung der Kontur des Objektes sein. Zwar erhält man beim Einsatz dieser Methode sehr viel mehr Information als bei anderen Verfahren, doch ist der Algorithmus wegen seiner rekursiven Natur recht langsam und hat je nach Implementierung einen hohen Speicherbedarf. Außerdem gilt es zu vermeiden, dass Punkte, die in eine bereits gefundene Region fallen, wieder einen Flood-Fill auslösen. In [4] und [5] wird ein solches Verfahren mit einer festen Punktmaske verwendet, deren Auflösung nach außen hin immer feiner wird. Dadurch wird dem Problem der schlechten Auflösung für entfernte Punkte im Bild, das bei der radialen Suche angesprochen wurde, entgegengewirkt.

Ziel dieser Studienarbeit war es, die Stärken beider Methoden zu vereinen und ein schnelles und robustes Verfahren zu entwickeln, dass sowohl Objekte (Roboter und Ball) als auch Linien auf dem Feld, Tore und Eckpfosten erkennen und in ein Weltmodell umsetzen kann. Dieser Ansatz wird im Folgenden detailiert beschrieben.

3.2 Beschreibung des Algorithmus

3.2.1 Gittermaske

Um eine gleichmäßige Abdeckung über den gesamten sichtbaren Bereich zu garantieren und dennoch die Anzahl der zu betrachtenden Pixel zu minimieren, wird zunächst – ähnlich des Rezeptorfeldes in [5] oder der „Jump Points“ in [4] – eine gitterförmige Punktmaske erstellt, die folgende Eigenschaften besitzt: Die Gitterlinien haben einen Abstand zueinander, der kleiner ist, als das kleinstmögliche Objekt (in unserem Fall beträgt der Abstand der Gitterlinien 20 cm). Es werden nur Punkte für die Maske zugelassen, die auf diesen Gitterlinien verlaufen. Auf den Gitterlinien selbst werden die Punkte im Abstand von 2 cm gewählt. Die ursprüngliche Maske ist in Abbildung 4a zu sehen. Das Gitter muss nun in das Kamerakoordinatensystem transformiert werden. Für die später stattfindende Abbildung der Objekte vom Kamera-

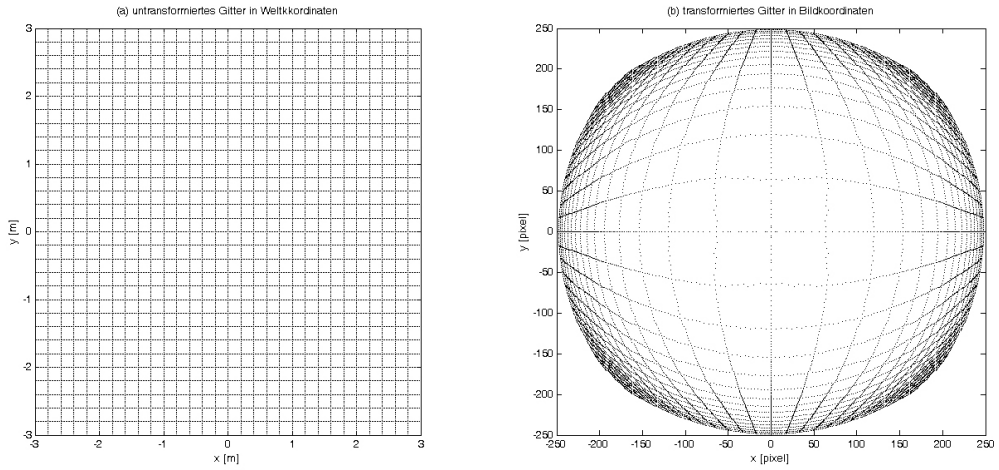

Abbildung 4: (a) Untransformierte gleichmäßige Gittermaske mit einer Auflösung von 20 cm zwischen zwei Gitterlinien und 2 cm entlang einer Gitterlinie. Dieses Gitter wird mit Hilfe der inversen Abbildungsfunktion $f_d^{-1} : \mathbb{R} \mapsto \mathbb{Z},\ r_w \to r_p$ transformiert. (b) zeigt das Ergebnis der Transformation. Die Abbildung ist surjektiv, doppelte Bildpunkte wurden entfernt.

koordinatensystem zurück ins Weltkoordinatensystem haben wir bereits die Abbildungsfunktion f_d erstellt (siehe Kapitel 2.2), für die Transformation des Gitters muss die Inverse dieser Abbildung errechnet werden. Dies geschieht ebenfalls mittels einer Interpolation durch ein Spline. Abbildung 5 zeigt die Umkehrfunktion f_d^{-1}.

Das in Kamerakoordinaten transformierte Gitter (im Weiteren *Gittermaske* genannt), kann nun in einer Datei gespeichert werden. Dabei ist die Reihenfolge der Punkte des Gitters relevant: Punkte entlang einer Gitterlinie werden fortlaufend mit x- und y-Koordinate gespeichert. Wird das Ende einer Gitterlinie auf der Gittermaske erreicht, so wird ein besonderer Punkt – ein sogenannter Stopp-Marker – eingefügt. Erst dann wird mit der nächsten Gitterlinie fortgesetzt. Diese Stopp-Marker haben die Koordinaten $(0, 0)$ und werden später bei der Implementierung des Algorithmus noch wichtig. Zunächst werden auf diese Art alle horizontal verlaufenden Gitterlinien gespeichert, dann die vertikalen Linien. Die fertige Datei wird vom ausführenden Programm zu Beginn eingelesen und für schnellen Zugriff im Hauptspeicher abgelegt.

Die hier verwendete Gittermaske mit einer Auflösung von 20 cm respektive 2 cm besteht – nach Entfernen von etwaigen doppelten Pixeln (die Abbildung f_d^{-1} ist am Rand stark surjektiv) – aus 16848 Punkten inklusive Stopp-Marker. Dies entspricht etwa $1/23$ des gesamten Bildes ($768 \times 576 = 442368$

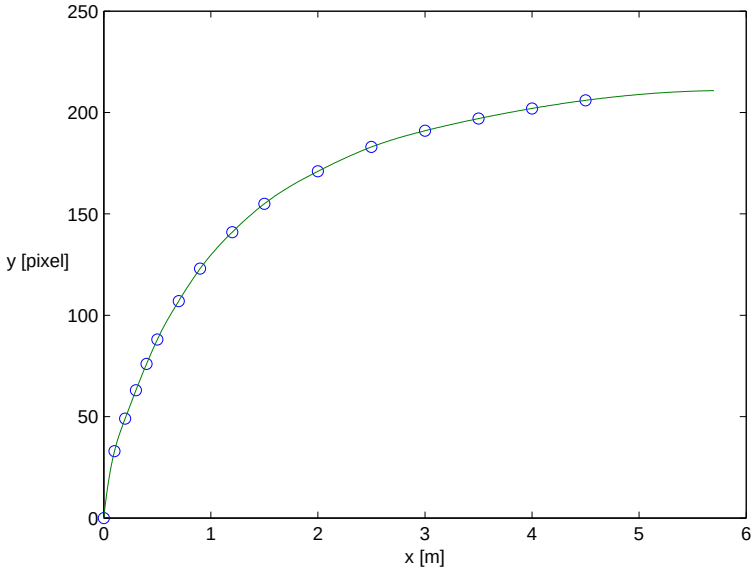

Abbildung 5: Abbildungsfunktion f_d^{-1} von Weltkoordinaten (x-Achse) nach Kamerakoordinaten (y-Achse). Die Kreise zeigen die gemessenen Stützpunkte für die Interpolation an.

Pixel), also ca. 4% – dementsprechend gering ist der Rechenaufwand dieser Methode gegenüber einem Verfahren, dass alle Bildpunkte des übertragenen Bildes betrachtet.

3.2.2 Smith-Watherman Verfahren

Die erstellte Gittermaske wird nun über jedes Bild gelegt; lediglich Punkte, die auf der Gittermaske liegen, werden im Weiteren betrachtet. Die ursprüngliche Idee war nun, fortlaufend über die einzelnen Gitterpunkte zu gehen und die Farbklasse des Pixels (nach Transformation des Farbwertes mittels f_c) zu betrachten um relevante Farbsprünge, etwa von grün nach schwarz, zu erkennen. Leider ist dies in der Praxis nicht ohne weiteres möglich. Störungen treten durch schlechte Beleuchtung, Schatten, Verdeckungen oder schlicht der begrenzten Auflösungsfähigkeit der Kamera in nahezu jedem Bild auf und verursachen unsaubere Ränder an Objekten und Pixel in Fehlfarben. So kommt es zu grauen und schwarzen Pixeln auf dem sonst grünen Spielfeld, weissen Pixeln auf dem Ball (durch Lichtreflektionen) und anderen unerwarteten Effekten. Ein System für die Objekterkennung muss also robust und fehlertolerant sein. Mit der Hoffnung, dass andere wissenschaftliche Gebiete mit ähnlichen Problemen zu tun haben, sind wir bei der Recherche nach exi-

stierenden Systemen, die die oben genannten Eigenschaften besitzen, schließlich in der Bioinformatik gelandet. Ein großes Teilgebiet beschäftigt sich hier mit dem Vergleich und der Analyse von DNA-Sequenzen. Bei der biologischen Reproduktion von DNA treten unvermeidlich Fehler auf: Einzelne Nukleotide werden übersprungen, vertauscht oder versehentlich eingefügt. Die Bioinformatik hat viele Methoden hervorgebracht, die speziell auf diese Problematik angepasst sind. Eine davon ist der sogenannte *Smith-Waterman Algorithmus* (siehe [6]), ein Verfahren zur Bestimmung von am besten korrelierenden Teilsequenzen zwischen zwei DNA-Strängen. Die beiden Sequenzen werden dabei nukleotidweise miteinander verglichen; für eine Übereinstimmung wird eine hohe positive Punktzahl vergeben, ungleiche Paare erhalten geringere positive oder sogar negative Punktzahl. Die Punktzahl wird dabei laufend aktualisiert, am Ende erhält man schließlich die am besten übereinstimmenden Teilsequenzen. Diese Idee schien uns für unsere Zwecke sehr geeignet, und so modifizierten wir den Algorithmus etwas, um ihn an unsere Problemstellung anzupassen.

3.2.3 Algorithmus zur Auffindung von Farbsegmenten

Die für uns interessanten Schlüsselfarben sind *schwarz* (für andere Roboter), *weiß* (für Linien auf dem Spielfeld) und *orange* (für den Ball). In Zukunft werden noch *cyan* und *magenta* hinzukommen, die die jeweiligen Teamfarben repräsentieren (siehe Kapitel 7). Für jede der Farben wird zunächst ein Profil erstellt, in dem die Primärfarbe, Sekundärfarben und Fehlfarben festgehalten werden. Für das Ballprofil zum Beispiel wäre *orange* die Primärfarbe, *weiß* wäre eine Sekundärfarbe, da oftmals Lichtspiegelungen auf dem Ball zu weißen Flecken führen, und *grün*, *schwarz*, *blau* und *gelb* wären Fehlfarben, da diese selten oder nie im Abbild des Balls vorkommen. Alle anderen Farben werden wie Sekundärfarben behandelt. Entsprechende Profile werden ebenfalls für Roboter und die Linien auf dem Feld angelegt. Alle diese Profile werden gleichzeitig verarbeitet, der Algorithmus läuft nur einmal pro Bild über alle Gitterpunkte und aktualisiert simultan die Punktzahl für jedes Profil.

Zu Beginn werden die Punktezahlen aller Profile auf null gesetzt. Dann wird die im Speicher abgelegte Gittermaske über das aktuelle Bild gelegt und der erste Pixel betrachtet. Entspricht dessen Farbe der Primärfarbe eines Profils, wird die Punktzahl dieses Profils um einen vorher festgelegten Wert erhöht. Für unsere Zwecke hat sich ein Wert von $+2$ für die Primärfarbe, -1 für die Sekundärfarben und -3 für Fehlfarben als geeignet erwiesen[6]. Zusätzlich wird für das Profil eine Startmarke an den aktuellen Pixel gesetzt. Bei Se-

[6]Zunächst erscheint die Wahl der Punktzahlen für Primär-, Sekundär- und Fehlfarben

kundär- und Fehlfarben wird dementsprechend von der jeweiligen Punktzahl subtrahiert. Nachdem alle Punktzahlen aktualisiert wurden, wird geprüft, ob für eines der Profile bereits einen abgeschlossenen Bereich gefunden wurde. Dies ist dann der Fall, wenn eine Punktzahl unter null sinkt und zuvor irgendwann das Maximum erreicht hat. Diese Maximumpunktzahl kann für jedes Profil unterschiedlich sein; sie hängt von der typischen Größe der Objekte und der Auflösung der Gittermaske ab. Wurde die Maximalpunktzahl nicht erreicht, so wird davon ausgegangen, dass das gefundene Segmentstück zu klein war, um als relevantes Objekt zu gelten. Eine Punktzahl, die am Ende eines Schrittes negativ ist, wird für die nächste Runde zurück auf null gesetzt. Daraufhin wird der nächste Pixel auf der Gittermaske betrachtet. Erneut werden die Punktezahlen aller Profile aktualisiert und geprüft, ob ein abgeschlossener Bereich gefunden wurde. Abildung 6 verdeutlicht den Ablauf des Algorithmus grafisch anhand eines orangenen Teilstücks, bei dem es sich um den Ball handeln könnte. Wie rechts in Abb. 6 gezeigt, sinkt die Punktzahl des orangenen Profils unter null und hat zuvor die Maximalpunktzahl erreicht. Somit wird ein orangener Bereich vom ersten orangenen Pixel bis zum Pixel, der als letztes die Maximalpunktzahl aufweisen konnte, für spätere Zwecke gespeichert. Da alle Profile gleichzeitig behandelt werden, muss jeder Pixel der Gittermaske pro Bild nur ein einziges Mal betrachtet werden, was wesentlich zur Geschwindigkeit dieses Algorithmus beiträgt.

3.2.4 Prüfung der gefundenen Segmente

Da man nicht davon ausgehen kann, das stets alle auf die oben beschriebene Weise gefundene Segmente zwangsläufig zu einem für uns relevanten Objekt gehören, wird jedes Segment noch auf seine Umgebung hin geprüft. Erst wenn vor und hinter dem Segment einige Pixel bestimmter, im jeweiligen Profil festgelegter, Farben auftritt, wird das Segment letztendlich übernommen. Damit vermeidet man, dass z.B. Zuschauer, die ausserhalb des Feldes stehen, als Roboter erkannt werden, oder dass Lichtreflektionen auf dem Ball als weiße Linien interpretiert werden. Jedes Profil enthält einige Farben, die es als Umgebung akzeptiert. Roboter können nur in grünem, blauem oder gelbem Kontext auftreten. Entweder stehen Sie vor einem Tor oder frei auf

recht willkürlich. Klar ist, dass die zu erkennende Farbe die Punktzahl eines Profils erhöhen muss, während andere Farben die Punktzahl reduzieren müssen. Es wäre nun möglich, eine große Anzahl von Bildern zu analysieren und statistische Werte für jedes Profil und jede Farbe zu ermitteln. Unsere Versuche zeigten aber, dass im Vergleich zu den arbiträr gewählten Punktzahlen nur wenig, wenn überhaupt etwas, gewonnen werden könnte. Auch bei vielen Implementierungen des ursprünglichen Smith-Waterman Algorithmus wird von dieser Vereinfachung Gebrauch gemacht.

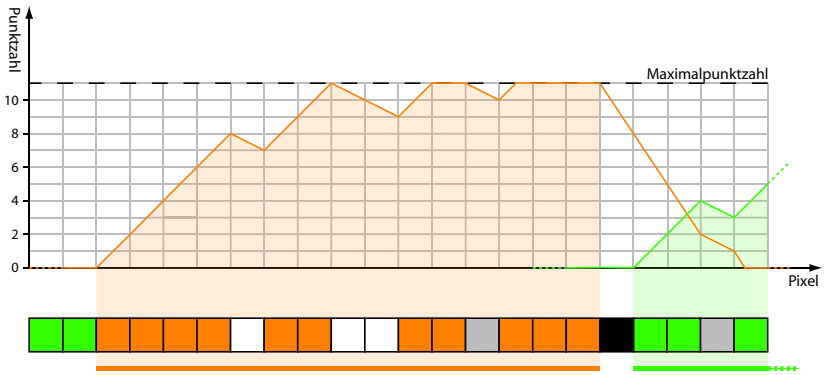

Abbildung 6: Beispielverlauf der Punktzahl für das Ballprofil: Bevor orangene Pixel gefunden werden, ist die Punktzahl gleich null (links). Für jeden gefundenen orangenen Pixel steigt die Punktzahl an, bei Fehlpixeln wird sie reduziert. Ab einem gewissen Maximum erhöht sich die Punktzahl nicht mehr weiter. Sinkt die Punktzahl unter null (rechts), wird der orangene Bereich vom ersten orangenen Pixel bis zum letzten Pixel mit Maximalwert als gefunden markiert und gespeichert. Verschiedene Profile laufen unabhängig und können sich – wie rechts in grün gezeigt – überlappen.

dem Feld. Befindet sich hauptsächlich andere Farben um ein schwarzes Segment herum, handelt es sich aller Wahrscheinlichkeit nach nicht um einem auf dem Feld befindlichen Roboter. Für die weißen Linien ist die Kontextprüfung noch stärker eingeschränkt. Sie können nur von grün umgeben sein. Ist dies nicht der Fall, wird das gefundene Segment verworfen. Mit diesem Verfahren kann es passieren, dass hin und wieder gültige Segmente nicht berücksichtigt werden. Dies hat sich jedoch als weitaus weniger tragisch herausgestellt, als einige „falsche“ Segmente in die weiterführenden Berechnungen einzubeziehen. Gerade die globale Lokalisierung der Roboter auf dem Feld, bei der die Feldlinien als Orientierung verwendet werden, ist in dieser Hinsicht recht empfindlich. Sie kommt zwar mit wenigen extrahierten Linienpunkten zurecht, nicht jedoch mit Phantomlinien, die an falschen Stellen auftauchen. Anders ist es mit dem Ball. Hier wird in unserem System keine Kontexterkennung für die orangenen Segmente mehr durchgeführt, da zu späterem Zeitpunkt eine Formerkennung des gefundenen Objektes stattfindet. Das Ergebnis der oben beschriebenen Schritte ist in Abbildung 7 zu sehen.

3.3 Erkennung von Toren und Eckpfosten

Zusätzlich zu den Objekten auf dem Spielfeld werden noch die beiden Tore und die vier Eckpfosten detektiert, falls sie im Bild sichtbar sind. Die

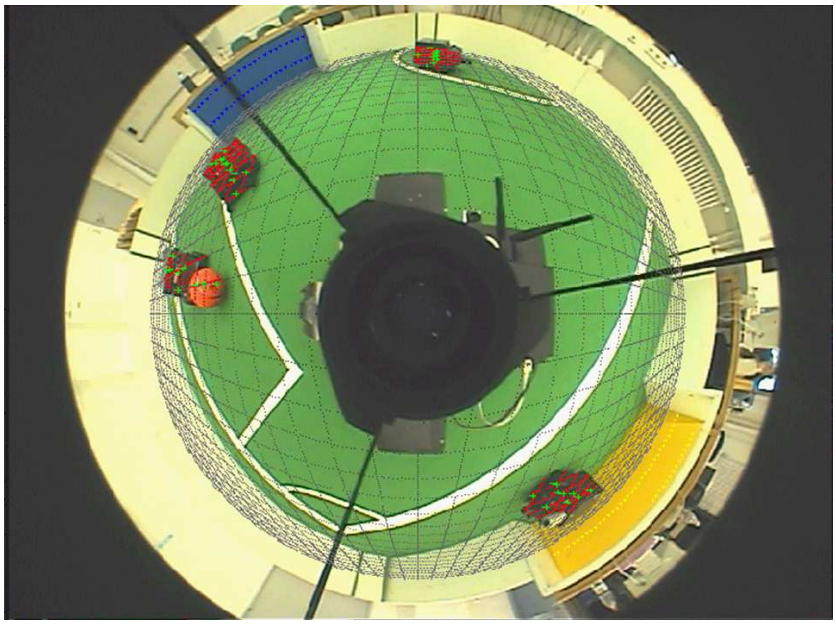

Abbildung 7: Nach Ausführung des Objekterkennungsalgorithmus wurden alle Segmente gefunden, die auf Objekten liegen (rot). Zusätzlich ist der Mittelpunkt eines jeden Segments (grünes Kreuz) mit eingezeichnet. Dieser wird für das Clustering benötigt. Ebenfalls erkennbar sind die beiden Tore, die von zwei Scanlinien durchlaufen werden. Die Eckpfosten waren bei der Aufnahme des Bildes nicht am Feldrand installiert.

Tore sind innen vollständig gelb bzw. blau angestrichen und haben eine Abmessung von 200 cm Breite, 125 cm Höhe und mindestens 50 cm Tiefe. Die Eckpfosten sind Zylinder mit einem Durchmesser von 20 cm Durchmesser und einer Höhe von 100 cm. Das obere und untere Drittel der Pfosten hat die selbe Farbe wie das näher gelegene Tor, das mittlere Drittel hat die Farbe des gegenüber liegenden Tores. Die Pfosten werden ca. 50 cm außerhalb des Spielfeldes an jede der vier Ecken postiert.

Diese Landmarken werden nicht mit dem oben beschriebenen Algorithmus erkannt, sondern separat detektiert. Zwei kreisförmige Linien mit Radien, die etwas kleiner sind als der sichtbare Kreisausschnitt des Bildes, werden nach blauen und gelben Pixeln abgesucht. Bei mehreren aufeinanderfolgenden Pixeln der selben Farbe wird an dieser Stelle radial von außen nach innen nach weiteren blauen bzw. gelben Pixeln gesucht. Ein Übergang von blau nach gelb oder umgekehrt vor Erreichen des grünen Bodens wird als Pfosten der entsprechenden Farbe interpretiert, bleibt die Farbe einheitlich bis zum Boden, wird das entsprechend gefärbte Tor an dieser Stelle vermutet.

Die Erkennung von Toren und Eckpfosten dient der Auflösung von Dop-

peldeutigkeit der Position eines Roboters auf dem Feld. Die Lokalisierung mittels Feldlinien ergibt stets zwei Positionen, die punktsymmetrisch zum Spielfeldmittelpunkt liegen. Erst durch zusätzliches in Betracht ziehen der Tore und Pfosten kann die Roboterposition eindeutig bestimmt werden.

4 Clustering

In diesem Kapitel wird beschrieben, wie die noch einzeln vorliegenden gefundenen Segmente aus Kapitel 3 zu Objekten gruppiert werden. Geclustert werden nur Roboter (schwarze Segmente) und der Ball (orangene Segmente). Die gefundenen weißen Liniensegmente werden direkt in ein Shared-Memory Segment geschrieben, wo sie für die Lokalisierungskomponente bereit liegen.

Nach einer kurzen Einführung zum Thema Clustering werden einige bewährte Clusteringmethoden vorgestellt und Ihre Vor- und Nachteile diskutiert. Anschließend wird ein modifiziertes Verfahren zum Clustern der Segmente beschrieben und die Funktionsweise anhand eines Beispiels dargestellt.

4.1 Allgemeines zu Clusteringverfahren

Ganz allgemein fassen Clustering-Methoden beliebige Datenpunkte einer Messung zu Gruppen zusammen, wobei die Punkte in einem Cluster in irgendeiner Weise ähnlich zueinander sind und sich von Punkten in anderen Clustern unterscheiden. Bereits die recht allgemein gehaltene Definition des Clustering lässt jedoch erkennen, dass es kein allgemein gültiges Kriterium gibt, dass unabhängig vom Ziel des Clusterings zum besten Ergebnis führt. Ein solches Kriterium muss deswegen stets vom Benutzer vorgegeben, und unter Umständen während des Clusteringvorgangs genauer spezifiziert oder verändert werden. Im Allgemeinen kann jedoch gesagt werden, dass beim Clustering hauptsächlich geometrische Beziehungen der Datenpunkte im hochdimensionalen Parameter-Raum – häufig der euklidische Abstand der Punkte untereinander – als Kriterium herangezogen wird.

Clusteringmethoden finden heute in sehr vielen Bereichen Anwendung. Der Oberbegriff hierfür heißt *Datamining* und beschreibt den Versuch, aus großen Datenmengen verallgemeinerte Informationen und Tendenzen abzuleiten. Gerade im Bereich des Marketings wird sehr viel Datamining betrieben. In großen Supermarkt-Ketten zum Beispiel werden jeden Tag Millionen von Produkten beim Verkauf an den Registrierkassen gescannt und gespeichert, wobei Gigabytes an auswertbaren Daten entstehen. Aus den Kombinationen von gekauften Artikeln vom selben Kunden kann ein Konzern die Kaufgewohnheiten der Kunden ermitteln und seine Marketingstrategien dar-

an anpassen. Auch bei Versicherungen wird Clustering eingesetzt, um etwa „Ausreißer“ bei Versicherungsfällen zu entdecken um damit Versicherungsbetrug vorzubeugen. Weitere Einsatzmöglichkeiten sind die Klassifikation von Pflanzen und Tieren nach deren äußerlichen Merkmalen oder die Sortierung und Gruppierung von ähnlichen Dokumenten und Webseiten im Internet für eine zielgerichtetere Suche, um nur einige Beispiele zu nennen.

4.2 Verschiedene etablierte Clusteringmethoden

Einige Clusteringverfahren sind in Wissenschaft und Wirtschaft wohluntersucht und weit verbreitet, darunter das *k-Means* Verfahren und das *nearest Neighbour* Clustering. Die beiden Methoden sollen hier exemplarisch vorgestellt werden.

4.2.1 k-Means

k-Means ist vermutlich das bekannteste Clusteringverfahren, es wurde bereits 1967 von J. MacQueen vorgestellt [7]. Die Hauptidee hierbei ist, die Datenpunkte in eine fixe, vorher festgelegte Anzahl von Clustern einzuteilen. Dabei erzeugt man k Schwerpunkte, die die Cluster repräsentieren. Jeder dieser Schwerpunkte wird zufällig im Parameter-Raum verteilt, wobei es je nach Initialisierung der Schwerpunkte zu unterschiedlichen Ergebnissen kommen kann. Dann werden alle Messpunkte dem jeweils nächstliegenden Cluster zugeordnet. Nun werden die Schwerpunkte aller Cluster entsprechend der zugeordneten Punkte neu berechnet und verschoben. Erneut werden alle Messpunkte dem jetzt am nächsten liegenden Cluster zugeordnet. Diese Schleife wird solange wiederholt, bis sich die Clusterzentren nicht mehr – oder nur noch unwesentlich – verändern. Es ist dabei durchaus möglich, dass Messpunkte mehrmals den Cluster wechseln. Es kann aber gezeigt werden, dass die Clusterzentren nach endlich vielen Schritten nicht mehr bewegt werden müssen.

Im Wesentlichen wird bei diesem Ansatz die Summe der quadratischen Distanz von jedem Punkt zu seinem nächsten Cluster minimiert: Bei einer gegebenen Menge von n Messpunkten $(p_1, ..., p_n)$ berechnet der Algorithmus die Clusterzentren $(c_1, ..., c_k)$, sodass die Summe

$$\sum_{i=1}^{n} D^2(p_i, c(p_i))$$

minimiert wird, wobei $c(p_i)$ das Clusterzentrum mit dem geringsten Abstand zum Punkt p_i darstellt.

Eine Schwäche dieses Verfahrens ist seine nicht-deterministische Eigenschaft durch die zufällig gewählten initialen Clusterzentren.

4.2.2 Nearest Neighbour

Das Nearest Neighbour Verfahren ist ein relativ einfaches Clusterverfahren, das jedoch in vielen Fällen bereits ausreicht. Gerade seine Einfachheit (und damit verbunden auch die relativ kurze Laufzeit) machen diese Clusteringmethode sehr attraktiv.

Jeder Messpunkt wird nur einmal betrachtet und dabei einem Cluster zugewiesen. Allerdings müssen alle Distanzen zwischen je zwei Punkten berechnet werden. Der Algorithmus läuft dabei wie folgt:

1. Setze $i = 1$ und $k = 1$. Ordne Messpunkt p_1 dem Cluster C_1 zu.
2. Setze $i = i + 1$. Betrachte Punkt p_i und finde den nächsten Nachbarn zu p_i aus der Menge von Punkten, die bereits klassifiziert wurden. Sei d_m die Distanz von p_i zum nächsten Nachbarn und C_m der Cluster des nächsten Nachbarn.
3. Wenn d_m unterhalb eines vorher festgelegten Schwellwertes t liegt, dann ordne p_i dem Cluster C_m zu. Andernfalls setze $k = k + 1$ und ordne p_i dem neuen Cluster C_k zu.
4. Wurden noch nicht alle Punkte betrachtet, fahre bei (2) fort.

Wie der Schwellwert t gewählt wird, hängt vom Datensatz ab, der geclustert werden soll.

4.3 Clustern der Objektsegmente

Der Objekterkennungsalgorithmus, der in Kapitel 3 beschrieben wurde, liefert als Ergebnis eine Liste von Segmenten, die im Bereich der schwarzen Objekte im Bild liegen (siehe Abbildung 7). Jedes dieser Segmente besteht aus einem Start- und Endpunkt und einem Mittelpunkt, wobei die drei Punkte in Polarkoordinaten bezüglich dem Bildmittelpunkt gespeichert sind. Da in der Regel mehrere Segmente vom selben Objekt gefunden werden, müssen sie nun den Hindernissen korrekt zugewiesen werden. Für diese Aufgabe ist ein Clusteringalgorithmus prädestiniert, denn die Segmente eines Hindernisses liegen geometrisch dicht beinander, während Segmente unterschiedlicher Objekte einen größeren Abstand aufweisen. Da im Vorraus nicht bekannt ist, wie viele Objekte sich auf dem Spielfeld befinden und wie viele davon vom

Roboter gesehen werden, eignet sich das k-Means Verfahren nur bedingt. Zwar gibt es eine Variante des Verfahrens für eine allgemeine Clusteranzahl, bei der k variiert und für jeden Durchlauf ein Qualitätsmaß berechnet wird, doch dieser viel höhere Aufwand steht in keinem Verhältnis zur eigentlichen Aufgabe. Ein leicht verändertes Nearest Neighbour Clustering hat sich für das hier vorliegende Problem als vollkommen ausreichend und sehr effizient herausgestellt.

Um ein noch besseres Ergebnis beim Clustering zu erzielen, werden nicht die Segmente mit Start- und Endpunkten geclustert, sondern lediglich ihre Mittelpunkte. Die Mittelpunkte liegen für ein Objekt noch dichter aufeinander und die Cluster können besser getrennt werden. Zudem reduziert sich die Anzahl der zu clusternden Punkte nochmals um die Hälfte.

4.3.1 Nearest Cluster statt Nearest Neighbour

Prinzipiell wird beim Nearest Neighbour Verfahren jeder Punkt nur einmal betrachtet und einem Cluster zugewiesen. Leider wird im Originalverfahren die Distanz jeden Punktes zu jedem anderen Punkt berechnet, was eine quadratische Laufzeit mit sich bringt. Dies ist aber nicht unbedingt nötig. Statt für einen Punkt seinen nächsten Nachbarn zu finden, wird im hier verwendeten Verfahren sein nächster Cluster gesucht. Dabei ist der Abstand eines Punktes p zu einem Cluster C die Distanz $D(p, m(C))$, wobei $m(C)$ der Schwerpunkt aller bereits vorhandenen Punkte im Cluster C darstellt. Schritt 2 und 3 im Algorithmus von 4.2.2 können also ersetzt werden durch folgende Zeilen:

2. Setze $i = i + 1$. Betrachte Punkt p_i und finde den nächsten Cluster C_m aus der Menge von bereits existierenden Clustern. Sei d_m die Distanz von p_i zu C_m.

3. Wenn d_m unterhalb eines vorher festgelegten Schwellwertes t liegt, dann ordne p_i Cluster C_m zu und berechne den neuen Schwerpunkt von C_m. Andernfalls setze $k = k + 1$ und ordne p_i dem neuen Cluster C_k zu mit Schwerpunkt p_i.

Dies hat nicht nur den Vorteil, dass weit weniger Distanzen berechnet werden müssen. Die Anzahl der Cluster ist vernachlässigbar gering im Vergleich zur Anzahl der Punkte. Zusätzlich ist garantiert, das der Radius eines Clusters nicht größer werden kann als t. In dieser Hinsicht ähnelt das hier verwendete Verfahren eher dem k-Means Clustering. Dort werden die Punkte auch dem nächsten Cluster zugewiesen anstatt dem Cluster des nächsten Nachbarpunktes. Es handelt sich hier also eher um ein *Nearest Cluster* Verfahren.

4.3.2 Wahl des Schwellwerts t

Die Frage ist nun, wie man t passend wählt, um ein zufriedenstellendes Ergebnis zu erzielen. Bei zu groß gewähltem Schwellwert gibt es unter Umständen nur einen großen Cluster, dem alle Punkte zugeordnet werden. Umgekehrt kann es bei zu kleinem t passieren, dass jeder Messpunkt in einen eigenen Cluster eingeteilt wird.

Beim RoboCup sind Größenvorgaben für die Roboter in den Regeln angegeben, diese Information kann zur Festlegung des Schwellwertes t genutzt werden. Allerdings wird nicht im Weltkoordinatensystem sondern im Bildkoordinatensystem geclustert, wo weiter entfernte Objekte im Bild kleiner erscheinen als nahe Objekte. Die Größe der Objekte im Bild varriert also mit der Distanz, und der Schwellwert t muss demnach abhängig vom Abstand der Objekte gewählt werden. Hier kann die Transformationsfunktion f_d (Kapitel 2.2) eingesetzt werden. Dabei tritt jedoch genau das Problem auf, das in Abbildung 3 dargestellt ist: Um eine genaue Position des Objektes in Weltkoordinaten zu bekommen, muss der nächste Punkt des Objektes zum Bildmittelpunkt für die Umrechnung mit f_d verwendet werden. Nach Bestimmung des Abstands des Objekts in Weltkoordinaten kann dessen maximale Ausdehnung im Bild berechnet und daraus der Schwellwert t bestimmt werden. Um nun tatsächlich für jeden Cluster den Punkt mit kleistem Radius r zu erhalten, können die Punkte beim Clustering nicht zufällig gezogen werden, sondern müssen vor Beginn des Clusterings einmal aufsteigend nach r sortiert werden. Mit diesem zusätzlichen Schritt ist sichergestellt, dass bei Beginn eines neuen Clusters (Schritt 3 im Algorithmus) stets der Punkt mit kleinstem Abstand zur Mitte zur Berechnung von t herangezogen wird.

4.3.3 Clustern nur nach Winkel ϕ

Theoretisch wäre es möglich, hintereinander stehende Roboter (also zwei Objekte, die den selben Winkel ϕ haben und unterschiedlichen Abstand r) beim Clustering zu trennen. Dazu dürften die Roboter jedoch nicht besonders hoch sein. Ein Objekt, dass höher ist als der Spiegel unseres Kamerasystems, verdeckt im Bild alles, was hinter ihm steht. Da unsere eigenen Roboter bereits etwas höher sind als ihre Spiegel, haben wir auf diese Möglichkeit im Moment verzichtet und begnügen uns damit, den jeweils ersten Roboter in einem gewissen Winkelsegment zu finden. Eventuell dahinter stehende Roboter werden in den selben Cluster miteinbezogen und nicht getrennt behandelt. Dies ist nicht weiter schlimm, da weiter entfernt liegende Objekte meist wenig Einfluss auf den eigenen Roboter haben, und sogar noch weniger, wenn in der direkten Linie noch ein anderes Hindernis liegt. Diese Vereinfachung hat

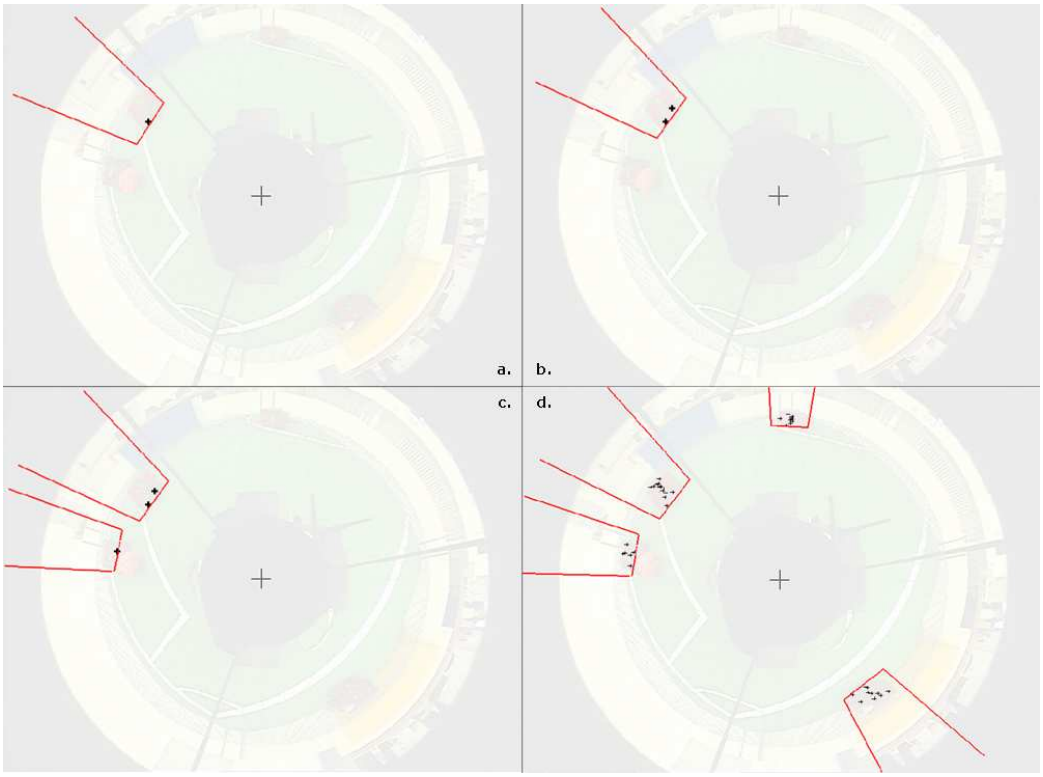

Abbildung 8: (a) Nachdem die Punkte nach ihrem Radius r sortiert wurden, beginnt das Clustering mit dem am nächsten zum Mittelpunkt liegenden Punkt. Der erste Punkt wird automatisch dem ersten Cluster zugeordnet. (b) Der nächste Punkt hat einen Winkelabstand zum Cluster, der kleiner ist als der Schwellwert t. Der Punkt wird dem selben Cluster zugeordnet und der Schwerpunkt des Clusters wird aktualisiert. (c) Die Distanz vom dritten Punkt zum Cluster liegt weit über t und ein neuer Cluster wird begonnen. (d) Nachdem alle Punkte betrachtet wurden, lieftert der Clusteringalgorithmus korrekt 4 Cluster zurück.

jedoch einen ganz entscheidenden Vorteil: Wir können den Abstand r beim Clustering einfach vernachlässigen und die Distanz zwischen Punkt und Cluster nur über den Winkel ϕ berechen. Die Distanzfunktion D zweier Punkte p und q lautet dann $D(p,q) = abs(\phi_p - \phi_q)$. Abbildung 8 zeigt einige Schritte des Clustering.

5 Nachbearbeitung

In Kapitel 4 wurde beschrieben, wie die gefunden Segmente der Roboter und des Balls durch einen Clusteringalgorithmus zu Clustern gruppiert werden. Ob jeder dieser Cluster tatsächlich einem Objekt in der Welt darstellt, wird in einem zusätzlichen Verifikationsschritt überprüft. Erst wenn sichergestellt ist, dass es sich tatsächlich um ein Objekt handelt, wird die Clusterposition in Weltkoordinaten transformiert und das Ergebnis ausgegeben. Dieses Kapitel beschreibt die Vorgehensweise der Prozedur.

5.1 Objekte verifizieren

In der Theorie sollte alles, was schwarz bzw. orange ist, tatsächlich auch ein Objekt bzw. der Ball sein. Dies ist jedoch in der Praxis leider nicht der Fall. Zuschauer, die um das Feld herum stehen, können am äußersten Rand des Bildes meist noch erkannt werden, das Kameraträgergerüst ist im Bild als drei nach außen laufende schwarze Streifen zu erkennen (siehe auch Abbildung 1), Schiedsrichter und Helfer tragen nicht immer die vorgeschriebenen schwarzen Hosen, und die magentafarbenenen Team-Marker an den Roboter mancher Teams haben häufig nicht den exakt definierten Farbton, sondern abweichende Farben. In solchen Situationen nutzt es oftmals nichts, bereits die Segmente mittels ihres Kontexts auszuschließen (wie in 3.2.4 beschrieben). Störungen der eben beschriebenen Art passen – auf der Ebene der Segmente betrachtet – gleichermaßen ins Profil wie die Segmente der tatsächlichen Objekte. Erst weiter oben im Erkennungsprozess – auf der Ebene der fertigen Objekte – können diese Fehlobjekte ausgeschlossen werden.

5.1.1 Zu kleine Objekte

Da es einen fest vorgegebenen Bereich für die Größe der Roboter gibt, können Objektkandidaten, die stark von der Vorgabe abweichen, verworfen werden. Dies betrifft insbesondere die schwarzen, radial nach außen verlaufenden Balken des Kameraträgers sowie Schatten, die häufig unter dem Ball für schwarze Flächen im Bild sorgen. Beide Typen von Objektkandidaten sind zu klein, um tatsächliche Objekte zu sein, und müssen vor der Ausgabe der gefundenen Objekte entfernt werden. Um diese Störungen zuverlässig vom Rest der Objekte zu unterscheiden, muss zunächst deren Größe bestimmt werden. Dies ist nicht trivial, da eine Distanzkalibrierung nur in einer Dimension – durch die Abbildungsfunktion f_d – im Bild von innen nach außen verlaufend vorliegt. Da der vom eigenen Roboter abgewandte Punkt eines Objekts sich nicht korrekt auf die Welt abbilden lässt (erläutert in Kapitel 2.2 und Abbildung

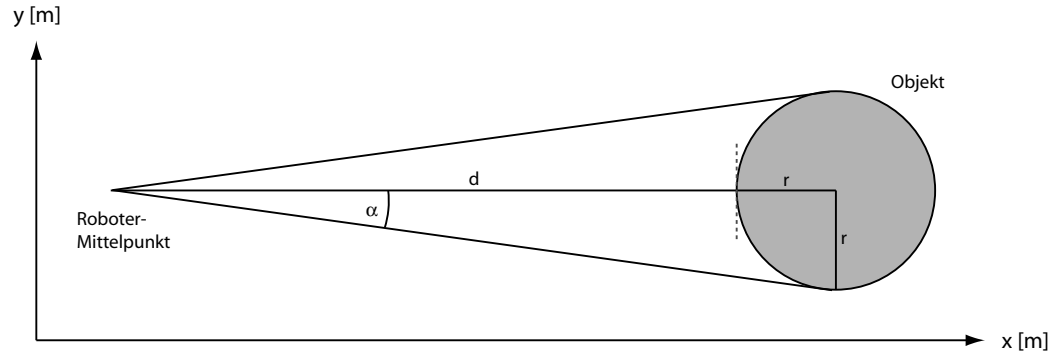

Abbildung 9: Aus dem Abstand d zum Objekt und dem Öffnungswinkel α lässt sich der Radius r des Objekts bestimmen. Der Abstand d kann durch Umrechnung des Abstands im Bild mittels Abbildungsfunktion f_d berechnet werden, der Winkel α bleibt bei der Transformation vom Bild- ins Weltkoordinatensystem erhalten.

3), muss hier eine Annäherung an die tatsächliche Objektgröße ausreichen. Aus Abbildung 9 ergibt sich folgender Zusammenhang aus Öffnungswinkel α des Objekts, dem Abstand d vom eigenen Roboter zum nächsten Punkt des Objekts in Weltkoordinaten und dem Radius r des Objekts, ebenfalls in Weltkoordinaten:

$$tan(\alpha) = \frac{r}{r + d} \tag{3}$$

Durch Umformen und Auflösen nach r erhält man aus (3) folgende Gleichung:

$$r = d \cdot \frac{tan(\alpha)}{1 - tan(\alpha)} \tag{4}$$

Da der Abstand vom Bildmittelpunkt zum nächsten Punkt des Objektes im Bild bekannt ist, lässt sich die Distanz d in Weltkoordinaten leicht mit Hilfe der Abbildungsfunktion f_d berechnen. Der Öffnungswinkel α bleibt bei der Transformation vom Bild- ins Weltkoordinatensystem erhalten. Somit kann der Objektradius r einfach bestimmt werden. Liegt r signifikant unterhalb der vorgeschriebenen Mindestgröße eines Roboters, kann das potentielle Objekt verworfen werden.

5.1.2 Zu große Objekte

Die in den Regeln vorgegebene Größenrestriktion schreibt nicht nur eine Mindestgröße, sondern auch eine Maximalgröße der Roboter vor. Analog zum oben beschriebenen Verfahren für zu kleine Objekte könnten zu große Objekte theoretisch ebenfalls verworfen werden. Zwei Punkte sind hier jedoch

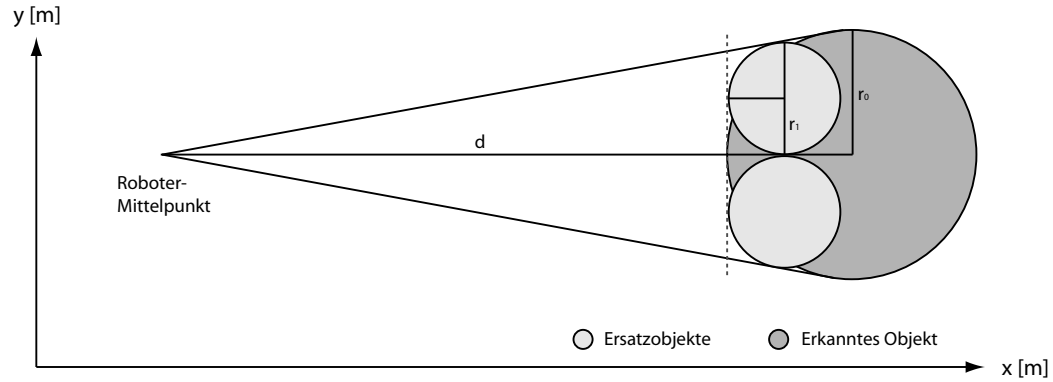

Abbildung 10: Ein zu großes Objekt wird durch zwei kleinere Objekte ersetzt. Die kleineren Objekte sollen den selben Abstand zum Roboter haben (gestrichelte Linie) und den selben Öffnungswinkel wie das große Objekt. Durch die Ersetzung entsteht viel Freiraum hinter den kleinen Objekten (dunkle Fläche).

zu beachten: Zum Einen erlaubt der Schwellwert t des verwendeten Clusteringalgorithmus keine beliebig großen Cluster, und zum Anderen ist die Interpretation bei zu großen Objekten eine andere als bei zu kleinen Objekten. Letztere können durch Schatten, schwach ausgeleuchtete Stellen oder den Kameraträger zum Vorschein kommen, und es kann davon ausgegangen werden, dass sich an der relevanten Stelle kein tatsächliches Objekt befindet. Im Fall eines großen schwarzen Bereichs auf dem Kamerabild muss (bei korrekter Farbkalibrierung der Kamera) jedoch angenommen werden, dass tatsächlich ein reales Objekt diesen schwarzen Fleck verursacht. Mögliche Interpretationen sind die Beine des Schiedsrichters nahe an der Kamera oder aber mehrere Roboter, die dicht beinander stehen, ohne dass eine Lücke zwischen ihnen erkennbar ist.

Die hier verwendete Lösung dieses Problems ist recht einfach: Anstelle des großen Objektes werden mehrere kleine Objekte so eingesetzt, dass sie das große Objekt möglichst gut repräsentieren. Abbildung 10 verdeutlicht dies. Die kleinen Objekte sollten die selbe Entfernung zum Robotermittelpunkt haben (gestrichelte Linie in Abbildung 10) und ihr gemeinsamer Öffnungswinkel sollte ebenfalls identisch sein. In der am häufigsten auftretenden Situation – nämlich der, dass mehrere Roboter aneinander stehen – ergibt diese Methode ein realistisches Modell der Umwelt. In den anderen oben genannten Fällen stellt die Umwandlung in mehrere kleinere Objekte zumindest keine Einschränkung dar, da Abstand und Öffnungswinkel des Hindernisses erhalten bleiben. Im Gegenteil entsteht durch das Ersetzen der Objekte sogar mehr Freiraum hinter den neuen Objekten (deutlich erkennbar in Abb. 10). Die Größe und Position der neuen Objekte lässt sich mit Hilfe des Strahlen-

satzes berechnen. Es gilt folgende Beziehung zwischen dem Abstand d und den Kreisradien r_0 (großes Objekt) und r_1 (kleine Objekte):

$$\frac{d + r_0}{r_0} = \frac{d + r_1}{2r_1} \tag{5}$$

Durch Umformen und Auflösen nach r_1 ergibt sich daraus:

$$r_1 = \frac{dr_0}{2d + r_0} \tag{6}$$

Die einzufügenden Objekte haben also den Radius r_1, die Positionen der Objekte lassen sich daraus leicht berechnen. Das alte Objekt kann nun aus der Liste der Objekte gelöscht und durch die neuen, kleineren Objekte ersetzt werden.

5.1.3 Verifikation des Ballobjekts

Der Ball spiel im Roboterfußball eine besondere Schlüsselrolle. Die Spielstärke eines Teams hängt unter anderem von der korrekten Erkennung und Lokalisierung des Balls ab. Anders als bei gegnerischen oder eigenen Robotern ist es unumgänglich, dass der Ball als solcher eindeutig auf dem Feld identifiziert wird. Falsch oder gar nicht erkannte Hindernisse führen zwar auch zu taktischen Fehlern, etwa einer Kollision mit einem anderen Roboter, oder einem falsch geplanten Pfad, den der Roboter abzufahren versucht. Wird aber der Ball nicht erkannt oder an der falschen Stelle vermutet, hat dies für das Spiel fatale Folgen, da das Primärziel für ein Team in jedem Fall der Ballbesitz sein muss.

Aus diesen Überlegungen heraus wurde für unser Team eine spezielle Ballerkennungskomponente entwickelt, die nach Eingabe eines Bildausschnittes, an dem der Ball vermutet wird, diesen genau analysiert und als Ergebnis zurückliefert, ob es sich hierbei tatsächlich um einen Ball handelte oder nicht. Diese Ballerkennung basiert auf einer *Hough-Transformation* (siehe [8]), einer Methode zur Erkennung von Formen – hier speziell Kreise – in einem Bild. Die Ballerkennung ist nicht Teil dieser Studienarbeit und es soll hier nicht weiter auf sie eingegangen werden. Liefert die Ballerkennung ein negatives Ergebnis, wird der Ball aus der Objektliste entfernt. Bei einer positiven Antwort wird zusätzlich überprüft, ob die Bounding-Box um das Ballobjekt ungefähr quadratisch ist, da der Ball im Bild stets als Kreis zu sehen ist. Bei extremen Unterschieden in den Seitenlängen der Bounding-Box wird vermutet, dass das Ball sehr nahe am eigenen Roboter liegt und deshalb teilweise verdeckt ist. Die Bounding-Box wird zum Robotermittelpunkt hin auf ein Quadrat

erweitert und der Kontaktpunkt zwischen Ball und Fußboden auf der neuer Kante des Quadrats liegend angenommen. So kann trotz Verdeckung des eigentlichen Kontaktpunktes eine verbesserte Ballposition berechnet werden.

5.2 Transformation in Weltkoordinaten

Alle Objekte, die durch den Verifikationsschritt nicht entfernt wurden, werden von der Bildverarbeitungskomponente als tatsächlich existierende, gültige Objekte angesehen. Diese Objekte müssen nun in das lokale Roboterkoordinatensystem umgerechnet werden, damit sie von höheren Routinen – etwa der Taktikkomponente – sinnvoll in die Planung mit einbezogen werden können. Die Objekte entsprechen dabei einer Struktur, die aus einer Position – dargestellt in Polarkoordinaten r und ϕ – und einem Radius bestehen. Zusätzlich bekommt das Objekt einen Typ, der momentan die Werte `BALL`, `ROBOT` oder `UNKNOWN` annehmen kann. Die Berechnung des Objektradius und der Distanz r geht analog zu der in Abbildung 9 gezeigten Methode. Der Winkel ϕ ist durch die omnidirektionale Kamera im Bild wie in der Welt der selbe. Schließlich bekommt das Objekt noch den korrekten Typ zugeordnet und wird dann in ein *Shared-Memory* Segment geschrieben, wo es für andere Programmteile zugänglich ist.

6 Resultate

Das in den vorangegangenen Kapiteln beschriebene System zur Objekt- und Linienerkennung wurde im Rahmen dieser Studienarbeit in C++ umgesetzt und läuft seit einiger Zeit zuverlässig als Softwarekomponente auf den RoboCup-Robotern. Besonderer Wert wurde bei der Programmierung auf Effizienz gelegt, da das gesamte Bilderkennungssystem in Echtzeit (d.h. aktuell mit 25 Bildern pro Sekunde) ausgeführt wird, wobei ein Großteil der Resourcen für andere Module, wie etwa der Lokalisierung, der Taktik und der Lowlevel-Steuerung des Roboters verfügbar bleiben soll. Wie in diesem Kapitel ausgeführt, ist es dennoch gelungen, innerhalb des eng vorgegebenen Zeitrahmens von wenigen Millisekunden eine robuste, genaue Objekt- und Liniendetektion zu implementieren.

Zunächst werden die Ergebnisse der Objekterkennung bei Eingabe des in Abbildung 1 gezeigten Bildes qualitativ diskutiert, gefolgt von einer quantitativen Laufzeitanalyse des Programms. Anschließend wird ein Experiment beschrieben, das die hier vorgestellte Erkennung von Objekten mit einer Messung durch das sehr genaue 4-Kamera Positionierungssystem W-CAPS [9] vergleicht.

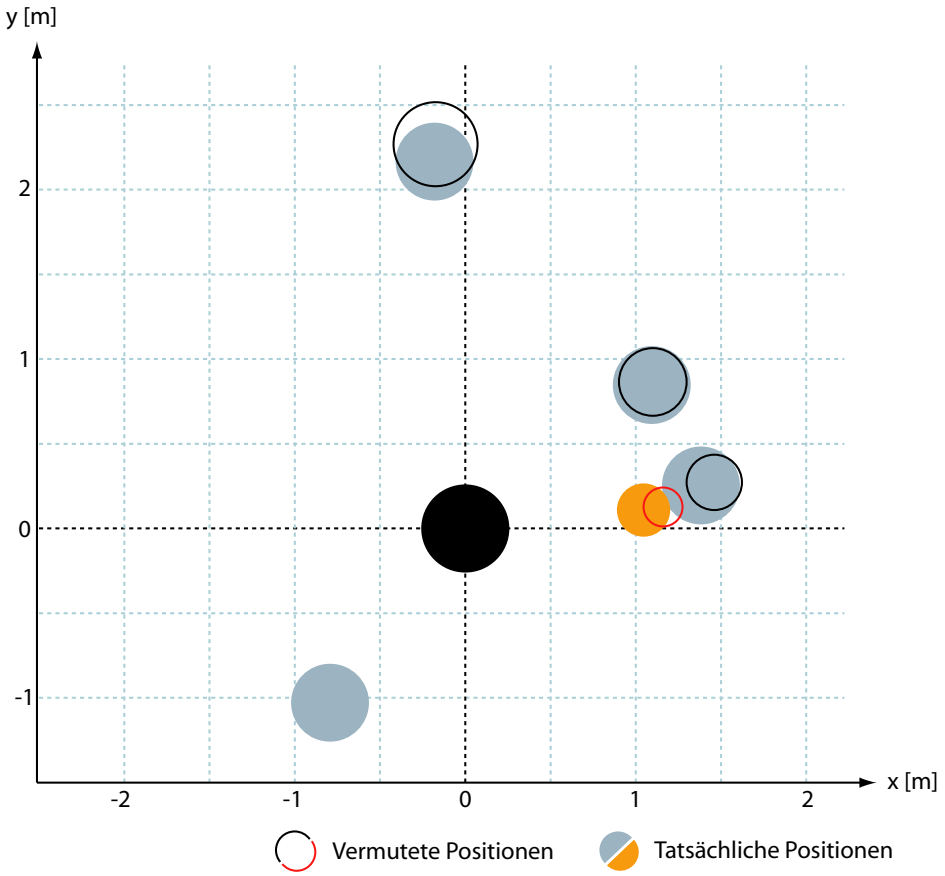

Abbildung 11: Vergleich von tatsächlicher Position der Objekte (ausgefüllte Kreise) mit den vom Objekterkennungsalgorithumus gelieferten Positionen (hohle Kreise).

6.1 Qualitative Diskussion der Objekterkennung

Das Ergebnis der Objekterkennung nach Eingabe der Situation aus Abbildung 1 kann in Abbildung 11 gesehen werden. Die ausgefüllten Flächen stellen die tatsächlichen Objekte dar, deren Positionen von Hand gemessen wurden. Die Roboter sind in grau eingezeichnet, während orange den Ball repräsentiert. Die hohlen Kreise symbolisieren die vom Algorithmus erkannten Objekte mit schwarzen Kreisen für die Roboter und einem roten Kreis für den Ball. Der ausgefüllte schwarze Kreis im Koordinatenursprung steht für den eigenen Roboter, auf dem sich die Kamera befindet.

Es fällt zunächst auf, dass sowohl die Distanz zum Ball, als auch zum Roboter, der den Ball führt, jeweils überschätzt wird. Dies liegt daran, dass der Kontaktpunkt zwischen Fußboden und Objekt in beiden Fällen nicht sichtbar ist. Der Kontaktpunkt des Roboters wird vom Ball verdeckt, während der

Ball selbst einen dunklen Schatten auf den Boden wirft, der nicht als orange erkannt wird. Da der Ball stets am unteren Rand etwas dunkler wirkt, handelt es sich in diesem Fall um einen systematischen Fehler, der im Algorithmus ausgeglichen werden kann. Anders verhält es sich bei Robotern. Das Problem der Verdeckung von Objekten lässt sich (zumindest im Rahmen dieser Studienarbeit) nicht ohne weiteres lösen.

Die Distanz zum am Weitesten entfernten Objekt (in Abbildung 11 oben) wird ebenfalls leicht überschätzt. Dies liegt schlicht an der Auflösung der omnidirektionalen Kamera, die nach außen hin beträchtlich schlechter wird als im Zentrum des Bildes. Bei einer Entfernung von mehr als vier Metern wird der Fehler sogar noch gravierender. Ein einzelnes Pixel im Bild kann bei diesen Distanzen mit dem verwendeten Kamerasystem bereits einen Unterschied von 30 cm oder mehr ausmachen.

Die gemessenen Positionen der anderen beiden Objekte weichen nur geringfügig von ihren tatsächlichen Positionen ab.

6.2 Quantitative Laufzeitanalyse

Für eine Laufzeitanalyse des Algorithmus wurde die Zeit für einen vollständigen Zyklus, die Zeit für die Objekterkennung alleine (wie sie in Kapitel 3 beschrieben ist) und die Zeit, die nur für das Clustering benötigt wird, über viele Wiederholungen gemessen und ein Mittelwert errechnet. Getestet wurde auf einem Pentium III Prozessor mit 850 MHz Taktfrequenz – die selbe Konfiguration, wie sie auch bei unseren Robotern verwendet wird. Ein kompletter Zyklus des Programms benötigte durchschnittlich 9.5 ms, wobei mehr als 97% der Rechenzeit von der Objekterkennungskomponente beansprucht werden. Das Clustering und die Nachbearbeitung fallen mit jeweils weniger als 1% kaum ins Gewicht. Die restliche Zeit wird intern für Funktionsaufrufe und Speicherzugriffe verwendet. Der große Anteil der Rechenzeit bei der Objekterkennung liegt unter anderem an der hohen Anzahl von Punkten auf der Gittermaske – etwaige Optimierungsversuche müssen demnach bei der Erkennung der Segmente ansetzen. Auch eine Reduzierung der Auflösung der Gittermaske würde eine bedeutende Steigerung der Performance mit sich bringen. Mit gröberer Auflösung sinkt jedoch auch die Genauigkeit der Lokalisierung der Objekte. Da der Algorithmus mit der aktuellen Gittermaske (20 cm Grobauflösung, 2 cm Feinauflösung) unseren Prozessornur zu ca. einem Viertel auslastet, ist es vorstellbar, dass wir in der Zukunft noch feiner aufgelöste Gittermasken mit 10 cm Grobauflösung verwenden, um die Genauigkeit weiter zu erhöhen.

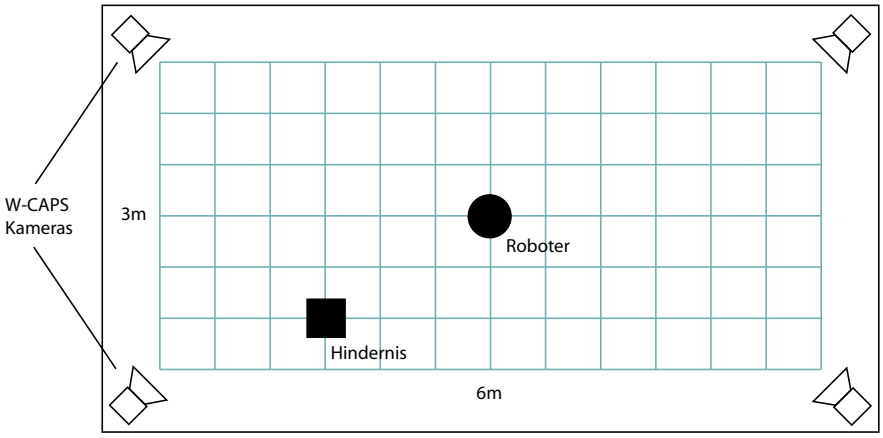

Abbildung 12: Schematischer Aufbau des Versuchs mit dem absoluten Positionierungssystem W-CAPS: In den vier Ecken des Raums wurden Kameras installiert, mit deren Hilfe durch Triangulierung eine exakte Position des Hindernisses (Quadrat) gemessen werden kann, das auf 90 verschiedene Positionen um den Roboter (Kreuzungspunkte des Gitters) gestellt wurde. Diese Position wird mit den gemessenen Werten des Omnikamerasystems auf dem Roboter in der Mitte (Kreis) verglichen.

6.3 Vergleich mit W-CAPS

Um eine objektive Aussage über den durschnittlichen Fehler des vorgestellten Systems machen zu können, wurde ein Experiment durchgeführt, dass die berechnete Distanz zu einem Objekt mit der von einem Absoluten Positionierungssystem (W-CAPS) gemessenen Distanz über viele verschiedene Positionen hinweg miteinander vergleicht. Dieses Positionierungssystem arbeitet mit vier in den Ecken eines Raumes angebrachten Kameras und basiert auf Triangulierung. Die Genauigkeit des Systems liegt bei sorgfältiger Kalibrierung unter 1 cm Abweichung von der tatsächlichen Position.

6.3.1 Versuchsaufbau

Das Experiment wurde in einem ca. 4×7 Meter großem Raum durchgeführt, in dem alle dunklen Gegenstände zuvor entfernt und Schatten so weit wie möglich durch Scheinwerfer ausgeleuchtet wurden, um Fehlmessungen auszuschließen. In jeder Ecke des Raums wurde eine Kamera in etwa 2.5 m Höhe installiert und an das W-CAPS Positionierungssystem angeschlossen. Die Kameras wurden, wie in [9] beschrieben, kalibriert.

In die Mitte des Raums wurde der Roboter mit Omnikamerasystem gestellt, während ein weiterer Roboter, der das Hindernis darstellte, nachein-

ander auf 90 Positionen um den im Zentrum stehenden Roboter plaziert wurde. Der Hindernisroboter hatte den für W-CAPS benötigten farbig markierten Hut montiert, der in einer Höhe angebracht war, die außerhalb des sichtbaren Bereichs der Omnikamera liegt. Eine schematische Darstellung des Versuchsaufbaus ist in Abbildung 12 zu sehen.

6.3.2 Durchführung

Der Hindernisroboter fuhr nacheinander 90 Positionen an, die in Abbildung 12 durch die Kreuzungspunkte des Gitters zu erkennen sind. Dabei kam es nicht auf eine exakte Positionierung durch den Roboter selbst an, da die Position durch W-CAPS nachgemessen wurde und dieser Wert beim Vergleich zugrunde lag. Am jeweiligen Punkt angekommen wurden einige hundert Lokalisierungen durch W-CAPS und durch das Omnikamerasystem durchgeführt und die Ergebnisse zur späteren Auswertung in eine Datei geschrieben. Bei einer mehrdeutigen Ausgabe des Algorithmus (trotz guter Beleuchtung wurden vereinzelt Schatten als Objekt erkannt und deshalb mehrere Positionen zurückgeliefert) wurde die Messung verworfen. Aus allen gültigen Messungen wurde sowohl für die W-CAPS Position als auch für die Omnikameraposition ein Mittelwert bestimmt und die Differenz der beiden gemittelten Distanzen berechnet.

6.3.3 Ergebnisse

Die Fehlerfunktion über alle gemessenen Punkte ist in Abbildung 13 dargestellt. Zwischen den gemessenen Punkten wurde linear interpoliert. Rote Flächen spiegeln einen hohen Fehler wider, während blaue Flächen einen geringen Fehler repräsentieren. Die Abweichungen liegen zwischen 0.77 cm und 60.11 cm mit einem mittleren Fehler von 8.16 cm. Im Bereich zwischen 0.7 m und 3.0 m Abstand ist das System sehr genau, wie die große dunkelblaue Fläche im Bild zeigt. Ganz nahe am Roboter werden Objekte vom eigenen Robotergehäuse teilweise verdeckt, der Kontaktpunkt zum Boden ist nicht sichtbar un die Positionen werden überschätzt. Nach vorne (in der Abbildung unten), wo die Frontkamera des Roboters montiert ist, macht sich dieser Fehler mit bis zu 35 cm besonders bemerkbar. Ein schlankeres, pyramidenförmiges Gehäuse des Roboters, zusammen mit einer höher montierten 360°-Kamera, wären erforderlich, um dieses Problem zu beheben, und sind für eine neue Robotergeneration bereits vorgesehen.

Der extreme Ausreißer links im Bild an der Position (-3.0, 0.0) beruht wahrscheinlich auf einer ungenauen Lokalisierung des W-CAPS Systems, da der Punkt nur von zwei der vier Kameras gesehen wurde (dieses Problem wird

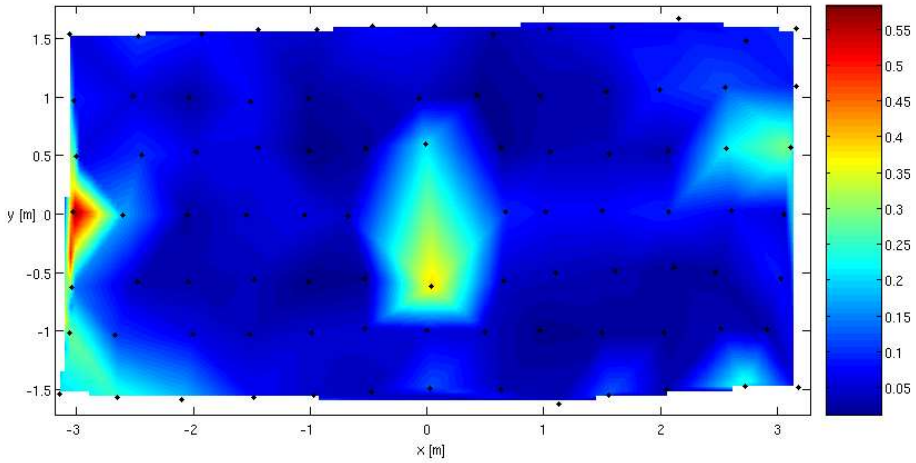

Abbildung 13: Linear interpolierter Fehler über 90 gemessene Positionen. Das Omnikamerasystem befindet sich auf einem Roboter im Koordinatenursprung. Bei sehr nahen und weit entfernten Objekten wird der Fehler größer.

auch in Abbildung 2 in [9] deutlich). Zwar ist die Distanz an dieser Position schon relativ hoch, alle anderen Punkte mit gleicher oder größerer Distanz wurden aber mit deutlich geringerem Fehler lokalisiert, was die Vermutung eines Messfehlers durch W-CAPS aufkommen lässt. Trotz des Ausreißers hat das System einen geringen durchschnittlichen Fehler und ist damit für eine Lokalisierung von Objekten beim RoboCup geeignet.

7 Ausblick

Das hier vorgestellte System zur Erkennung von Objekten und anderen Landmarken in einer farblich markierten Umgebung, wie sie unter anderem beim RoboCup vorkommt, wird seit einiger Zeit erfolgreich auf unseren Robotern eingesetzt und hat inzwischen den Laserscanner als Hauptsensor für die Lokalisierung von Hindernissen abgelöst. Dennoch gibt es einige Punkte, an denen das System verbessert werden kann. Darunter fällt zum Beispiel eine Unterscheidung von gegnerischen und eigenen Robotern. Beim RoboCup werden die Roboter einer Manschaft mit cyan- und magentafarbenen Markern versehen, die von jeder Seite aus sichtbar sein müssen. Die Segmenterkennung müsste für Cyan und Magenta erweitert werden, um zusätzlich die Marker im Bild zu finden. Diese Erweiterung ließe sich rasch durchführen, da sie völlig analog zu den anderen Farbprofilen durchgeführt werden kann. Anschließend müsste aber eine weitere Überprüfung auf Überlappungen von schwarzen und

cyan- bzw. magentafarbenen Bereichen stattfinden. Die Marker müssten den gefundenen schwarzen Objekten zugeordnet werden, was bei nahe aneinander oder hintereinander stehenden Robotern zu falschen oder mehrdeutigen Zuordnungen führen könnte und weitere Überlegungen erforderlich macht.

Desweiteren könnte, durch eine präzisere Bestimmung des Kontaktpunktes zwischen Roboter und Fußboden, die Genauigkeit der Lokalisierung eines Objekts – besonders am äußeren Rand des Bildes – erheblich gesteigert werden. Anstatt jedoch die Auflösung der Gittermaske weiter zu erhöhen (was letztlich auch zu einer genaueren Lokalisierung, aber auch, wie in Kapitel 6.1 beschrieben, zu erheblich mehr Rechenzeit führen würde), könnte stattdessen eine lokale Suche an interesanten Stellen im Bild zum selben Ergebnis führen, ohne die Rechenzeit erheblich zu vergrößern. Diese lokale Suche sollte nur für im äußeren Bildrand liegende Cluster erfolgen, da dank der hohen Auflösung in der Mitte des Bildes nur wenig Verbesserung zu erwarten ist.

Zuguterletzt ist eine Umsetzung des Algorithmus für eine perspektivische Kamera ebenfalls denkbar. Lediglich die Gittermaske und die Abbildungsfunktion f_d müssten angepasst werden, die Idee des Algorithmus könnte aber in großen Teilen übernommen werden. Dieser Schritt wird irgendwann auch notwendig werden, da bereits nach wenigen Metern die Auflösung der omnidirektionalen Kamera nicht mehr für eine exakte Lokalisierung ausreicht. Das RoboCup-Komitee hat bereits eine weitere Vergrößerung des Spielfelds auf bis zu 12×16 Meter für die Weltmeisterschaft 2005 in Osaka, Japan angekündigt. Bei immer größer werdenden Feldern ist abzusehen, dass die omnidirektionale Kamera irgendwann nur noch für lokale Weltmodellierung im nahen Umfeld des Roboters eingesetzt werden kann. Der Überblick über das gesamte Spielfeld muss schließlich durch eine perspektivische Kamera (oder auch mehrere) erfolgen.

Literatur

[1] André Treptow, Andreas Masselli and Andreas Zell *Real-Time Object Tracking for Soccer-Robots without Color Information.* Proceedings of the European Conference on Mobile Robotics (ECMR 2003)

[2] Robert Hanek, Thorsten Schmitt, Sebastian Buck, Michael Beetz *Fast Image-based Object Localization in Natural Scenes.* Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, EPFL, Lausanne, Switzerland (2002)

[3] Daisuke Sekimori, Tomoya Usui, Yasuhiro Masutani, and Fumio Miyazaki *High-Speed Obstacle Avoidance and Self-Localization for Mobile Robots Based on Omni-directional Imaging of Floor Region.* RoboCup 2001, LNAI 2377, pp.204-213, Springer-Verlag Berlin Heidelberg (2002)

[4] M. Jamzad, B.S. Sadjad, V.S. Mirrokni, M. Kazemi, H. Chitsaz, A.Heydarnoori, M.T. Hajiaghai, and E. Chiniforooshan *A Fast Vision System for Middle Size Robots in RoboCup.* RoboCup 2001, LNAI 2377, pp. 71-80, Springer-Verlag Berlin Heidelberg (2002)

[5] Andrea Bonarini, Paolo Aliverti, Michele Lucioni *An omnidirectional vision sensor for fast tracking for mobile robots.* IEEE Instrumentation and Measurement technology Conference, IMTC 99, Piscataway, NJ, IEEE Computer press (1999)

[6] T.F. Smith, M.S. Waterman *Identification of common molecular subsequences.* Journal of Molecular Biology, Volume 147(1), pp. 195-197 (1981)

[7] J. MacQueen *Some methods for classification and analysis of multivariate observations.* Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (1967)

[8] P.V.C. Hough *Method and means for recognizing complex patterns.* U.S. Patent 3,069,654 (1962)

[9] Achim Lilienthal, Tom Ducket *An Absolute Positioning System for 100 Euros.* Proceedings of the IEEE International Workshop on Robotic Sensing (2003)