Selbstlokalisierung im Roboter-Fußball unter Verwendung einer omnidirektionalen Kamera

Diplomarbeit an der Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften

vorgelegt von:

Erik Schulenburg

Betreuer: Prof. Dr. B. Nebel

2003

Zusammenfassung

Roboter-Fußball bietet seit einigen Jahren der internationalen Forschung die Möglichkeit, verschiedene Lösungsansätze für Probleme der mobilen Robotik zu vergleichen. Zu diesen Problemen gehören Lokalisierung, Pfadplanung, Bildverarbeitung und Multi-Agenten-Koordination, um nur einige zu nennen. Diese Arbeit beschäftigt sich mit einer *omnidirektionalen Kamera*, die durch einen Spiegel einem mobilen Roboter ein 360°-Blickfeld ermöglicht. Dabei beschreiben wir den Aufbau einer solchen Kamera und unseren Entwurf eines speziellen Spiegels, der an die Anforderungen beim Roboter-Fußball angepasst ist. Weiter erläutern wir unser Vorgehen bei der *Mustererkennung*. Dieses ermöglicht eine effiziente Erkennung der Spielfeldlinien, Tore und Eckpfosten. Anschließend entwickeln wir zwei Verfahren zur *Selbstlokalisierung* eines Fußballroboters anhand dieser Daten. Das erste Verfahren ist eine Erweiterung des *LineMatch-Algorithmus*, der bereits sehr erfolgreich zur Lokalisierung durch Laser-Scanner eingesetzt wird. Das andere Verfahren ist die *Monte-Carlo-Lokalisierung*, eine mächtige und weit verbreitete Methode zur Positionsbestimmung eines mobilen Roboters. In unseren abschließenden Tests versuchen wir zu bewerten, wie hilfreich die unterschiedlichen Beobachtungen (Linien, Tore, Eckpfosten) bei der globalen bzw. lokalen Selbstlokalisierung sind.

Danksagung

Für die Betreuung meiner Arbeit bedanke ich mich bei Alexander Kleiner. Er hatte stets viele Anregungen und Ideen. Außerdem danke ich Thilo Weigel für die freundliche und hilfreiche Unterstützung bei der LineMatch-Lokalisierung. Besonderer Dank gilt Björn Fischer für den fruchtbaren Gedankenaustausch bei der Monte-Carlo-Lokalisierung und die Tipps zur Handhabung der Roboter. Jorge Blanco Iglesias danke ich für seine große Hilfe bei der Berechnung des Spiegelprofils und für seine „Entwerfung" des Kameragehäuses.

Weiterer Dank gilt meiner Freundin Livia Predoiu für ihre Geduld und die wertvollen Vorschläge bei der Ausformulierung der Arbeit. Auch meiner Familie danke ich für ihre Unterstützung, besonders meinen Eltern für die Rechtschreibkorrektur. Schließlich möchte ich mich bei Herrn Prof. Nebel bedanken, der mir diese Arbeit ermöglicht und mich durch seine freundliche Art motiviert hat.

Für meine Eltern.

Die Gesetze der Robotik:

1. Ein Roboter darf keinen Menschen verletzen oder durch Untätigkeit zu Schaden kommen lassen.
2. Ein Roboter muss den Befehlen der Menschen gehorchen – es sei denn, solche Befehle stehen im Widerspruch zum Ersten Gesetz.
3. Ein Roboter muss seine eigene Existenz schützen, solange dieses Handeln nicht dem Ersten oder Zweiten Gesetz widerspricht.

ISAAC ASIMOV (1920-1992)
Ich, der Robot [Asimov 1952]

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Schach dem Weltmeister

Im Jahr 1950 erschien im *Philosophical Magazine* ein bemerkenswerter Artikel von Claude E. Shannon mit dem Titel *„Programming a Computer for Playing Chess”* [Shannon 1950]. Darin diskutiert der amerikanische Mathematiker und Computerwissenschaftler die Möglichkeit, einen Automaten zu programmieren, der einen Menschen im Schach schlagen kann. Diese Idee beschäftigte von diesem Zeitpunkt an Wissenschaftler auf der ganzen Welt. In der Künstlichen Intelligenz entstand die Spiele-Theorie und es wurden mächtige Suchverfahren und Lernstrategien entwickelt. Gleichzeitig konnte auch die Öffentlichkeit Anteil an der stetigen Entwicklung nehmen, wenn beispielsweise ein Schachcomputer – zunächst mit mäßigem Erfolg – an einer Weltmeisterschaft teilnahm. Ihren Höhepunkt erreichte die Disziplin des Computerschach, als 1996, knapp 50 Jahre nach Shannons Aufsatz, zum ersten Mal der amtierende Schachweltmeister von einem Computer besiegt wurde. Garry Kasparov unterlag dem von IBM entwickelten Supercomputer *Deep Blue* [Schaeffer 1997].
Zu diesem Zeitpunkt waren sich allerdings die Forscher auf der ganzen Welt einig, dass das Computerschach zwar die Entwicklungen auf dem Gebiet der Künstlichen Intelligenz beflügelt hatte, aber keinen wirklichen Prüfstein für die Leistungen maschineller Intelligenz darstellte. Die Art, in der ein Computer Schach spielt, nämlich riesige Datenbanken zu durchsuchen und unzählige Zugmöglichkeiten voraus zu berechnen, entspricht keineswegs der menschlichen Herangehensweise an das Spiel. Die deterministische Natur des Schachspiels, durch die man theoretisch jeden Zug im Voraus berechnen kann, erfordert aus heutiger Sicht keine wirkliche Intelligenz, sondern „nur” riesigen Rechenaufwand.

Auf der Suche nach einem neuen Standardproblem für die Künstliche Intelligenz wurde 1995 das Fußballspiel vorgeschlagen [Kitano et al. 1995]. Um den neuesten Entwicklungen der KI Rechnung zu tragen, bei der die Robotik mehr in den Mittelpunkt gerückt ist, sollen autonome Roboter gegeneinander antreten. Bei dieser Art von Problem, dem Handeln in einer realen Umgebung, treten ganz andere Aspekte der Intelligenz in den Vordergrund. Ein Roboter muss in Echtzeit mit unvollständigen Informationen umgehen und auf unvorhergesehene Ereignisse reagieren können. Er muss sich in einer dynamischen Welt zurechtfinden, um erfolgreich sein Ziel zu erreichen. Insbesondere verspricht man sich durch den direkten Vergleich verschiedener Ansätze im Rahmen eines Wettbewerbes Erkenntnisse auf den Gebieten *Lokalisierung*, *Pfadplanung*, *Aktorik*, *Sensorik*, *Sensor-Fusion*, *reaktives Verhalten*, *Lernen*, *Planen* und *Multi-Agenten-Systeme*.

Seit 1997 finden jedes Jahr der *RoboCup*, die Roboter-Fußball-Weltmeisterschaft, und andere Wettbewerbe statt, in der Regel in Verbindung mit Konferenzen über Künstliche Intelligenz [Kitano 1998, Asada und Kitano 1999, Veloso et al. 2000, Stone et al. 2001, Birk et al. 2002]. Erklärtes Ziel der Initiatoren von *RoboCup* ist es, bis zum Jahr 2050 den menschlichen Fußball-Weltmeister zu besiegen [Burkhard et al. 2002]. Diese Zeitspanne entspricht in etwa der zwischen Shannons Aufsatz und der Niederlage von Garry Kasparov gegen Deep Blue.

1.2 RoboCup

Eine RoboCup-Meisterschaft wird in verschiedenen Disziplinen ausgetragen, bei denen unterschiedliche Schwerpunkte gesetzt sind. Die wichtigsten Ligen sind dabei:

- Die SIMULATIONS-LIGA (SIMULATION LEAGUE): Hier treten Software-Agenten auf einem virtuellen Spielfeld gegeneinander an. Da man sich dabei nicht mit Hardware-Problemen und fehlerbehafteter Sensorik und Aktorik beschäftigen muss, kann man sich auf Lernverfahren und Multi-Agenten-Koordination konzentrieren.

- Die LIGA DER KLEINEN ROBOTER (SMALL SIZE LEAGUE): Eine Mannschaft besteht aus vier Robotern mit einer Grundfläche von jeweils maximal 180 cm^2. Die Roboter können über Funk miteinander kommunizieren. Das Spielfeld hat die Größe einer Tischtennisplatte und als Ball dient ein orangefarbener Golfball. Die lokalen Sensoren der Roboter dürfen durch eine gemeinsame Kamera, die über dem Spielfeld angebracht ist, ergänzt werden.

- Die LIGA DER MITTELGROSSEN ROBOTER (MIDDLE SIZE LEAGUE): Die Roboter in dieser Klasse dürfen einen maximalen Durchmesser von 50 cm und eine Höhe von 80 cm haben. Auch hier bestehen die Teams aus jeweils vier Spielern, die über Funk miteinander verbunden sind. Allerdings sind nur lokale Sensoren erlaubt, d. h. alle Sensoren müssen sich auf den Robotern befinden. Das Feld hat eine Größe von 10×5 m^2 und der Ball ist ein handelsüblicher orangefarbener Fußball.

- Die Liga menschenähnlicher Roboter (Humanoid League): Hier spielen zweibeinige Roboter gegeneinander. Dieser Wettbewerb wurde im Jahr 2002 zum ersten Mal ausgetragen. Zur Zeit stellen das Laufen und Schießen die größten Herausforderungen dar.
- Die Sony-Liga (Sony Legged Robot League): Als Hardware dienen hier die AIBO-Roboterhunde der Firma Sony. Da die Hardware festgelegt ist und von den Teams nicht verändert werden darf, kommt es bei diesem Wettbewerb ausschließlich auf die Programmierung an.
- Die Liga der Rettungs-Roboter (Rescue League): Ein mögliches Einsatzgebiet für mobile Roboter ist der Katastrophenschutz. In diesem Wettbewerb wird daher versucht, mit Software-Agenten Rettungseinsätze zu planen (Simulation) und mit Teams von autonomen Robotern ein unwegsames Gelände zu erkunden, um zum Beispiel verschüttete Menschen in Erdbebengebieten zu finden.

Das Interesse an diesen Wettbewerben ist in den letzten Jahren stark gestiegen. Bei der ersten Weltmeisterschaft 1997 in Nagoya, Japan, waren etwa 40 Teams in den drei Wettbewerben Simulation-, Small Size- und Middle Size League am Start. Die meisten davon in der Simulations-Liga, da diese mit dem geringsten technischen Aufwand verbunden ist. Bei der Weltmeisterschaft 2002 in Fukuoka, ebenfalls Japan, waren bereits 188 Teams aus 29 Ländern vertreten.

1.3 Die Middle Size League

Die Klasse der mittelgroßen Roboter wird derzeit von vielen als die Königsklasse des Roboterfußballs angesehen. Eine Mannschaft besteht in der Regel aus drei Feldspielern und einem Torwart. Die Spieler sind in Größe und Gewicht Beschränkungen unterworfen. Sie dürfen maximal eine Grundfläche von 50×50 cm^2 und eine Höhe von 80 cm haben und höchstens 80 kg wiegen. Zur Grundausrüstung eines Fußballroboters gehören in der Regel ein Antriebssystem, ein Schussmechanismus, eine Kamera und ein Computer. Zusätzlich sind beliebige andere Sensoren erlaubt. Außerhalb des Spielfeldes darf sich ein weiterer Computer befinden. Zwischen diesem und den Robotern können über eine Funkverbindung beliebige Daten ausgetauscht werden. Sämtliche Sensoren müssen sich allerdings auf den Robotern befinden. Durch die Kommunikation zwischen den Robotern soll mannschaftliches Zusammenspiel ermöglicht werden.
Ein offizielles RoboCup-Spielfeld ähnelt einem echten Fußballfeld, jedoch ist es deutlich kleiner. Es ist 10 m lang und 5 m breit und durch weiße Linien auf einem grünen Untergrund markiert, wie in Abbildung 1.1 zu sehen. Um den Robotern die Orientierung zu erleichtern, sind weitere Markierungen vorhanden. Die Tore sind massive Torkästen von 2 m Breite. Sie sind gelb bzw. blau gestrichen, damit sie

leicht erkannt werden können. Außerdem befinden sich an den Ecken des Spielfeldes runde, 90 cm hohe Pfosten mit einem Durchmesser von 20 cm. Bis auf das mittlere Drittel haben die Eckpfosten dieselbe Farbe, wie das Tor auf ihrer Seite des Feldes. Das mittlere Drittel hat die Farbe des gegenüberliegenden Tores.
Zu Beginn der RoboCup-Initiative befand sich an Stelle der Außenlinie eine Spielfeldbande. Sie verhinderte, dass der Ball ins Aus geschossen werden konnte. Seit dem Jahr 2002 ist diese Bande jedoch entfernt und durch eine Reihe von Pfosten ersetzt worden, die sich in einem Meter Entfernung um das gesamte Spielfeld zieht.

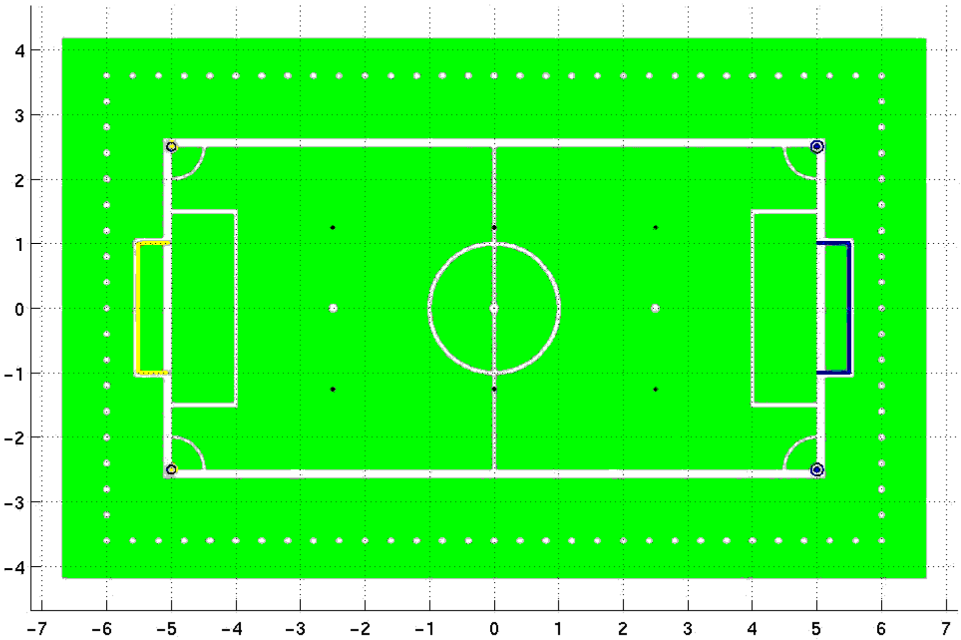


Abbildung 1.1: *Das offizielle Spielfeld der Middle Size League*

Außer der Farbe der Tore und Eckpfosten sind auch für die anderen Objekte auf dem Feld Farben festgelegt. Als Ball wird ein orangefarbener Lederball verwendet, wie er beim echten Fußball im Schnee zum Einsatz kommt. Die Roboter müssen in Schwarz gehalten werden und bekommen farbige Markierungen. Bei einem Team sind diese Marker hellblau, beim anderen magenta. All diese Farben wurden so gewählt, dass sie möglichst große Kontraste bilden, um die Identifizierung der Objekte zu erleichtern.

Diese und weitere Regeln sind von einem Komitee von Robotik-Forschern festgelegt worden [RoboCup 2002]. Im Wesentlichen dienten dabei die Vorschriften des Weltfußballverbandes (FIFA) als Grundlage. In regelmäßigen Abständen werden sie überarbeitet und erweitert. So soll beispielsweise das Spielfeld nach und nach vergrößert werden.

1.4 Das CS Freiburg RoboCup-Team

Das Team *Computer Science Freiburg* (*CS Freiburg*) ist die RoboCup-Mannschaft der Albert-Ludwigs-Universität Freiburg. Sie tritt seit 1998 in der Middle Size League an und hat die beachtliche Bilanz von drei Weltmeistertiteln und einem dritten Platz vorzuweisen, sowie mehrere Deutsche Meistertitel.

1.4.1 Hardware

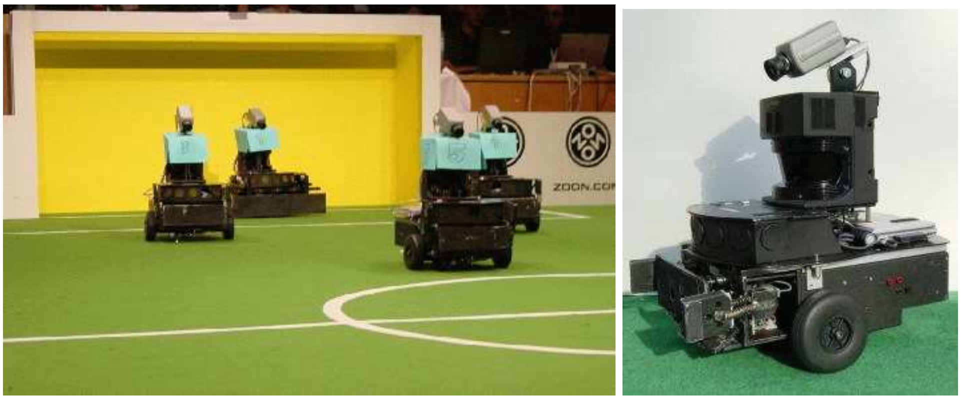


Abbildung 1.2: *Die Spieler des CS Freiburg*

Die Spieler des CS Freiburg sind in Abbildung 1.2 zu sehen. Bei diesen Robotern handelt es sich ursprünglich um *Pioneer I* Roboter, wie sie von der Firma *ActivMedia Robotics* hergestellt werden. Allerdings wurde die Hardware deutlich erweitert und den Bedürfnissen angepasst [Weigel et al. 2002]. Das Controller Board wurde durch ein *Pioneer II* Board ersetzt. Außerdem wurde eine Schussvorrichtung entwickelt, die es nicht nur erlaubt, über das ganze Feld zu schießen, sondern auch ein Führen des Balls möglich macht. Dazu sind an den Seiten Flipper angebracht, die bei Bedarf heruntergeklappt werden.

Die wichtigsten Sensoren der Roboter sind:

- Die ODOMETRIE: Dabei werden die Drehbewegungen der beiden Antriebsräder gemessen, wodurch Rückschlüsse auf die Positionsänderungen des Roboters möglich sind[1]. Dieser Sensor lässt sich zur Lokalisierung, d. h. Positionsbestimmung des Roboters einsetzen. Allerdings sammeln sich dabei mit der Zeit Fehler an, so dass die Odometrie durch andere Sensoren unterstützt werden muss.

[1]Der Pioneer-Roboter besitzt einen *Differentialantrieb*, d. h. er hat zwei unabhängige Antriebsräder. Drehen sich die Räder unterschiedlich schnell, fährt der Roboter auf einer Kreisbahn. Bewegen sie sich in verschiedene Richtungen, kann er sich sogar auf der Stelle drehen.

- Der LASER-SCANNER: Dieser Sensor gehört zur Klasse der sogenannten *Time-Of-Flight-Sensoren*. Das Gerät sendet Lichtstrahlen aus, die von den Objekten in der Umgebung reflektiert werden. Aus der Zeit, die das Licht benötigt, um zum Sensor zurückzukehren, lassen sich die Entfernungen zu den Objekten berechnen. Die vom CS Freiburg verwendeten Laser-Scanner sind vom Typ *SICK LMS200*. Diese liefern Entfernungsmessungen für einen Bereich von 180° vor dem Roboter mit einer Auflösung von 0,5° und einer Genauigkeit von 1 cm. Mit dem Laser-Scanner bestimmt der Roboter seine eigene Position auf dem Feld und misst die Position anderer Roboter.

- Die KAMERA: Diese ist ein unverzichtbarer Sensor für alle Fußball-Roboter. Beim CS Freiburg werden Digitalkameras des Modells *DFW-V500* von Sony verwendet. Sie dienen dazu, den Ball anhand seiner orangenen Farbe zu erkennen.

Eine weitere wichtige Hardwarekomponente der Roboter sind die *Sony Vaio PCG-C1VE*-Notebooks mit einer Prozessorleistung von 600 MHz und 112 MB Hauptspeicher. Auf diesen läuft ein Linux-Betriebssystem, unter dem sich die RoboCup-Software starten lässt. Über eine Firewire-Schnittstelle ist ein solcher Computer direkt mit der Kamera verbunden. Über USB können die Motoren und der Laser-Scanner angesteuert werden. Zur Kommunikation unter den Spielern und mit einem Server außerhalb des Spielfeldes dienen *WaveLan*-Karten von *Lucent-Technologies.*

1.4.2 Software-Architektur

Das RoboCup-System des CS Freiburg ist so ausgelegt, dass jeder Spieler als vollständig autonomer Roboter handeln kann, ohne auf die Kommunikation mit den anderen Spielern angewiesen zu sein. Mittels seiner Sensorinformationen erstellt jeder Spieler sein eigenes lokales Weltmodell, d. h. ausgehend von einem a-priori-Modell des Spielfeldes bestimmt er seine eigene Position, die Position des Balls und die Positionen der anderen Roboter (siehe Abbildung 1.3). Anhand dieser Daten kann er seine Aktionen wählen und ausführen.

Erst durch Kommunikation ist es aber möglich, dass die Spieler ihre Taktik koordinieren. Dazu bestimmt jeder Spieler eine Menge von Rollen, die er in der gegenwärtigen Spielsituation übernehmen kann, z. B. den Ball spielen, sich zum Pass anbieten oder verteidigen; außerdem bewertet er, wie nützlich er in dieser Rolle wäre. Durch Absprache wird bestimmt, welcher Spieler dann tatsächlich welche Rolle übernimmt [Weigel et al. 2002]. Ein weiterer nützlicher Aspekt der Kommunikation ist, dass die Roboter ihre Informationen über die Umwelt miteinander teilen können. Dazu schicken alle Roboter ihre Beobachtungen an einen Server außerhalb des Spielfeldes. Dort werden die Beobachtungen zu einem einheitlichen globalen Weltmodell vereinigt und an die Spieler zurückgeschickt.

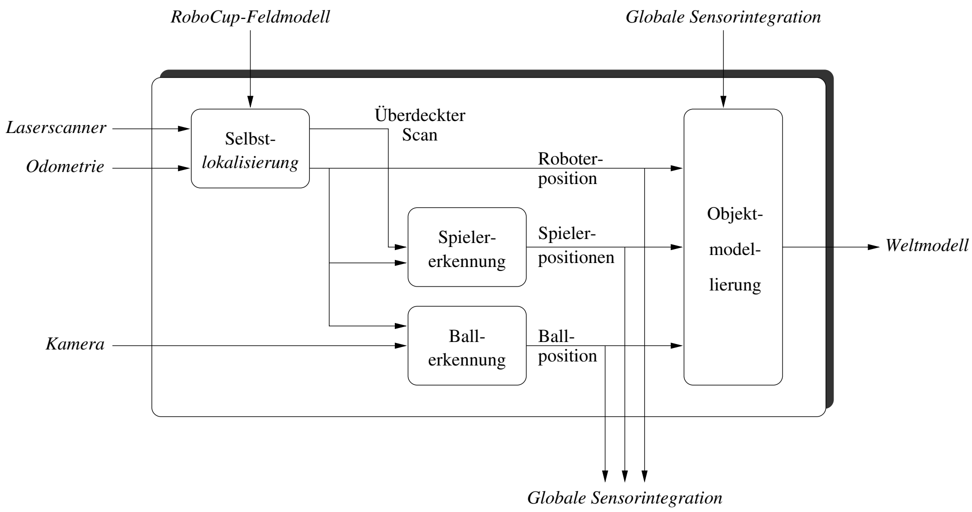


Abbildung 1.3: *Ablauf der Weltmodellierung*

Für die soeben beschriebenen Prozesse und zum Ausführen der jeweiligen Aktionen muss ein einzelner Roboter eine Anzahl an Grundfertigkeiten beherrschen und ein Menge von Teilproblemen lösen können. Dazu gehören unter anderem die Bestimmung der eigenen Position (Selbstlokalisierung), die Objekterkennung, das Planen eines Weges zu einer bestimmten Zielposition (Pfadplanung) und das Schießen des Balls. Für jede dieser Aufgaben sind eigene Module zuständig, die zahlreiche Studien-, Diplom- und Doktorarbeiten umfassen [CS Freiburg 2001].

1.5 Kamera, Merkmalsextraktion und Selbstlokalisierung: die Zielsetzung der Arbeit

Als eine wichtige Voraussetzung für ein erfolgreiches Spiel wird im Allgemeinen angesehen, dass ein Fußball-Roboter seine Position auf dem Spielfeld kennt. Dieses Problem wird in der Robotik als *Selbstlokalisierung* bezeichnet. Mit der Position ist dabei normalerweise der Ort und die Orientierung des Roboters gemeint. Um diese Werte anzugeben, benötigt der Roboter ein Modell der Umwelt, also eine interne Repräsentation der Umgebung, in die ein Koordinatensystem gelegt wird.
Beim System des CS Freiburg besteht dieses *Weltmodell* aus den Linien des Feldes, den Toren und Eckpfosten. Das karthesische Koordinatensystem hat seinen Ursprung im Mittelpunkt des Feldes und die x-Achse zeigt in Richtung des gegnerischen Tores. Die Position eines Roboters wird in diesem Koordinatensystem durch ein Tripel $(x, y, \theta)^T$ angegeben, wobei (x, y) den Ort bezeichnet, an dem er sich befindet und θ der Winkel ist, der seine Ausrichtung beschreibt. Der Selbstlokalisierung kommt beim CS Freiburg eine zentrale Bedeutung zu [Fischer 2002]. Die Erfolge des Teams beruhen zu einem großen Teil darauf, dass die Positionen der Spieler effizient und robust bestimmt werden können. Dazu werden aus einem Laserscan Linien extrahiert, die den Banden des Spielfeldes zugeordnet werden. In Verbindung mit den Daten der Odometrie ist damit eine sehr exakte Positionsschätzung möglich [Weigel 1998].
Infolge einer Regeländerung sind die Spielfeldbanden entfernt worden. Statt dessen umgibt nur noch eine Reihe von Pfosten das Spielfeld (siehe Abschnitt 1.3). Dadurch erschwert sich die Selbstlokalisierung mittels des Laser-Scanners und wird in absehbarer Zukunft unmöglich werden, da geplant ist, auch die Pfosten nach und nach zu entfernen. Aus diesem Grund müssen andere Sensoren zur Positionsbestimmung herangezogen werden. Dafür bietet sich insbesondere die Kamera an, die beim CS Freiburg bisher nur zur Ballerkennung eingesetzt wurde. Mit der Kamera lassen sich die verschiedenen *Landmarken*[2] (Tore, Eckpfosten, Feldlinien) erkennen. Dieser Prozess wird als *Mustererkennung* oder *Merkmalsextraktion* bezeichnet. Aus diesen Daten lässt sich daraufhin eine Positionsschätzung ableiten.
Eine handelsübliche Kamera hat den Nachteil, dass sie nur ein begrenztes Sichtfeld besitzt. Deshalb wird von vielen RoboCup-Teams ein besonderer Typ Kamera eingesetzt, die *omnidirektionale Kamera*. Mit einer solchen Kamera ist es einem Roboter möglich, in alle Richtungen gleichzeitig zu sehen, er erhält also ein 360°-Blickfeld. Das Grundprinzip ist dabei das Folgende: Eine normale Kamera wird senkrecht nach oben gerichtet. Über ihr ist ein konvexer Spiegel befestigt, der die Umgebung des Roboters abbildet. Durch die Wölbung des Spiegels ist der Ausschnitt der Welt, der von ihm reflektiert wird, größer, als es mit einem ebenen Spiegel möglich ist. Allerdings ist das Bild im Allgemeinen stark verzerrt. Um eine Vorstellung von diesem Vorgang zu erhalten, verweisen wir auf Abbildung 1.4.

[2] So werden allgemein Objekte bezeichnet, die eine Positionsbestimmung des Roboters ermöglichen.

Abbildung 1.4: M.C.ESCHER: *Hand mit spiegelnder Kugel,* Lithographie, 1935

Mit dieser Arbeit soll eine omnidirektionale Kamera in das System des CS Freiburg integriert und zur Selbstlokalisierung eines Roboters eingesetzt werden.
Der erste Teil beschäftigt sich mit dem Aufbau einer omnidirektionalen Kamera. Einem Vorschlag von *Marchese und Sorrenti* [Marchese und Sorrenti 2001] folgend, berechnen wir einen Spiegel, der speziell an die Gegebenheiten beim RoboCup angepasst ist und die wichtigsten Bereiche des Spielfeldes mit möglichst geringen Verzerrungen abbildet.
Daran anschließend versuchen wir, wichtige Merkmale der Umgebung im Kamerabild zu erkennen. Neben den Toren und Eckpfosten wurde dabei besondere Sorgfalt auf die Detektion der Spielfeldlinien verwendet, die bei der Selbstlokalisierung ei-

ne wichtige Rolle spielen.
Für die eigentliche Lokalisierung schlagen wir zwei unterschiedliche Verfahren vor. Beim ersten Ansatz werden die aus dem Bild extrahierten Spielfeldlinien mit einem Linienmodell des Feldes in Deckung gebracht. Aus der dazu erforderlichen Drehung und Verschiebung ergibt sich eine Positionsschätzung für den Roboter. Mit Hilfe eines *Kalman-Filters* wird diese Schätzung mit den Odometriedaten abgeglichen. Der zweite Ansatz ist die *Markov-Lokalisierung*. Diese ist eine mächtige und häufig verwendete Methode zur Selbstlokalisierung. Genauer gesagt wird eine Variante der Markov-Lokalisierung eingesetzt und zwar die *Monte-Carlo-Lokalisierung*. Durch Beobachtungen der Tore, Eckpfosten und Spielfeldlinien wird die wahrscheinlichste Roboterposition berechnet.
Im letzten Teil der Arbeit beschreiben wir die Durchführung einiger Tests mit den Lokalisierungsverfahren. Bei der Monte-Carlo-Lokalisierung versuchen wir dabei, Aussagen über den Nutzen verschiedener Beobachtungen zu treffen. Diese sollen eine Weiterentwicklung und Verbesserung unseres Verfahrens ermöglichen.

Kapitel 2

Die omnidirektionale Kamera

Die Kamera ist für einen Fußballroboter ein sehr wichtiger, wenn nicht der wichtigste Sensor. Alle Objekte auf dem Spielfeld lassen sich anhand ihrer Farbe identifizieren. In der Middle Size League sind keine globalen Sensoren erlaubt, d. h. es ist nicht erlaubt, eine Kamera zu benutzen, die sich außerhalb des Spielfeldes befindet. Statt dessen muss man jeden Roboter mit einer Kamera ausstatten, die eine lokale Sicht der Spielsituation bietet. Dazu hat man verschiedene Möglichkeiten, die alle ihre Vor- und Nachteile besitzen [Lima et al. 2001]:

- Eine PERSPEKTIVISCHE KAMERA ist in einer festen Blickrichtung auf dem Roboter angebracht. Sie erfasst einen bestimmten Sichtbereich, der typischerweise einen Öffnungswinkel von ca. 70° hat. Um einen anderen Ausschnitt der Umgebung zu sehen, muss sich der ganze Roboter drehen. Nachteilig ist das enge Sichtfeld, das durch nahegelegene Objekte, z. B. andere Roboter, verdeckt werden kann. Perspektivische Kameras sind in der Robotik weit verbreitet, da sie einfach zu verwenden sind. Auch beim CS Freiburg sind solche Kameras im Einsatz und werden erfolgreich zur Ballerkennung genutzt.

- Eine BEWEGLICHE KAMERA lässt sich durch einen Motor drehen oder hoch und runter bewegen. Dadurch muss sich nicht mehr der gesamte Roboter bewegen, um in eine andere Richtung zu blicken. Schwierig ist dabei die Koordination von Roboter- und Kamerabewegungen. Solche Kameras werden beispielsweise von den *GMD-Robots* eingesetzt [Bredenfeld et al. 2002].

- Durch Verwendung MEHRERER KAMERAS kann man verschiedene Bereiche rund um den Roboter überblicken. Dazu benötigt man aber eine mehrfache Bildverarbeitung und auch der erhöhte Energieverbrauch kann von Nachteil sein. Beim CS Freiburg ist bisher nur der Torwart mit zwei Kameras ausgestattet worden, um sein Sichtfeld zu vergrößern.

- Als KATADIOPTRISCHE KAMERA bezeichnet man eine Kamera, deren Sichtfeld durch einen oder mehrere Spiegel und Linsen erweitert wird. Häufig verwendet werden die sogenannten *omnidirektionalen Kameras*, bei denen die Kamera nach oben auf einen konvexen Spiegel gerichtet ist. Dadurch wird ein Panorama-Blick von 360° ermöglicht. Bereits beim ersten RoboCup 1997 hatten die *„Trackies"* der Universität von Osaka ihren Torwart mit einer solchen Kamera ausgerüstet [Suzuki et al. 1998]. Seitdem wird diese Technik von mehr und mehr RoboCup-Teams genutzt.

Die Vorteile eines 360°-Blickfeldes durch eine omnidirektionale katadioptrische Kamera sind offensichtlich: der Roboter kann in alle Richtungen sehen, ohne sich oder die Kamera zu bewegen. Allerdings hat dieser Ansatz auch einige Nachteile. Zunächst sei hierbei die unvermeidliche Verzerrung des Kamerabildes genannt. Wenn die genaue Form des Spiegels bekannt ist, lässt sich das Bild aber problemlos entzerren. Des Weiteren ist die Genauigkeit eines omnidirektionalen Bildes geringer, da ein viel größerer Ausschnitt der Welt auf die gleiche Anzahl an Pixeln abgebildet wird. Wirkt man dem entgegen und erhöht die Kameraauflösung, steigt wiederum die zu verarbeitende Datenmenge stark an.

In den folgenden Abschnitten dieser Arbeit wird zunächst der prinzipielle Aufbau eines omnidirektionalen Kamerasystems, kurz Omnicam, beschrieben. Anschließend werden die speziellen Anforderungen für den Roboterfußball analysiert und ein Spiegel entworfen, der diese Anforderungen möglichst gut erfüllt.

2.1 Aufbau und Funktionsweise

Für eine omnidirektionale Kamera benötigen wir im Prinzip zwei Dinge: einen konvexen rotationssymmetrischen Spiegel und eine Kamera. Zur Vereinfachung des Modells und der mathematischen Berechnungen nehmen wir an, es handelt sich dabei um eine *Lochkamera*[1]. Die Kamera wird senkrecht nach oben gerichtet und der Spiegel wird so über ihr positioniert, dass die optische Achse der Kamera und die Symmetrieachse des Spiegels übereinstimmen. Dieser Aufbau ist in Abbildung 2.1 skizziert.
Die Lichtstrahlen, die von den Objekten der Umgebung ausgehen, werden vom Spiegel reflektiert und treffen durch das Pinhole auf die Sensorebene der Kamera. Hier werden die Farbanteile und die Intensität des Lichtes gemessen. Dadurch entsteht ein verzerrtes Abbild der Umgebung in dessen Mittelpunkt die Kamera selbst zu sehen ist. Die Verzerrung hängt dabei von der Krümmung des Spiegels ab. Bisher wurden verschiedene Spiegelprofile vorgeschlagen und erprobt, und zwar unter anderem konische, hyperbolische und sphärische Formen. In Abbildung 2.3 ist ein

[1] Eine Lochkamera besitzt statt einem Objektiv ein kleines Loch, das im Englischen als *Pinhole* bezeichnet wird. Alle Lichtstrahlen, die von der Kamera aufgenommen werden, müssen durch diesen Punkt treffen.

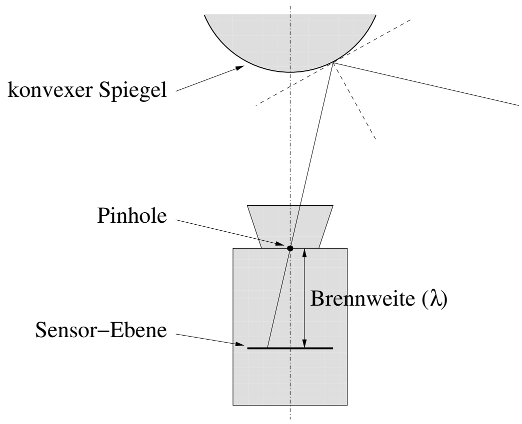


Abbildung 2.1: *Aufbau einer omnidirektionalen katadioptrischen Kamera*

Bild zu sehen, das bei Verwendung eines sphärischen (kugelförmigen) Spiegels entsteht.

Um den Spiegel in seiner Position über der Kamera zu halten, gibt es zwei Möglichkeiten. Die eine ist, den Spiegel an einem Gestänge zu befestigen. Dadurch wird ein Teil des Sichtbereiches verdeckt, wie in Abbildung 2.3 zu erkennen. Die andere Möglichkeit ist, die gesamte Konstruktion mit einem Glas- oder Plexiglas-Zylinder zu umgeben und den Spiegel am oberen Ende des Zylinders zu befestigen. Diese Anordnung ist sehr stabil, hat jedoch den Nachteil, dass das Glas des Zylinders die Schärfe des Bildes beeinträchtigt.

2.2 Anforderungen an die Bildverarbeitung

Wie in Abschnitt 1.4.2 beschrieben, hat ein Roboter verschiedene Teilprobleme zu lösen, um erfolgreich Fußball spielen zu können. Dazu muss er nicht nur die relevanten Objekte anhand ihrer Farbe identifizieren können, sondern auch deren Richtung und Entfernung abschätzen. Diese relative Position geben wir in Polarkoordinaten (d, φ) an, wobei d die Distanz und φ die Richtung bezeichnet, in der sich ein Punkt befindet. Die omnidirektionale Kamera bildet einen solchen Punkt $P_r = (d_r, \varphi_r)$ gemäß einer Funktion τ auf einen Punkt $P_b = (d_b, \varphi_b)$ in der Sensorebene ab. Diese Transformation setzt sich zusammen aus einer Spiegelung an der gekrümmten Fläche des Spiegels und einer Projektion gemäß der Lochkamerageometrie. In Abbildung 2.2 sehen wir dazu eine Skizze.

Nehmen wir an, dass die Symmetrieachse des Spiegels senkrecht zum betrachteten zweidimensionalen Unterraum der Welt – dem Feld – steht, lässt diese Transformation den Winkel unverändert. Die Distanz ändert sich jedoch. Es gilt also:

$$(d_b, \varphi_b) = \tau(d_r, \varphi_r) = (\tau_d(d_r), \varphi_r) \tag{2.1}$$

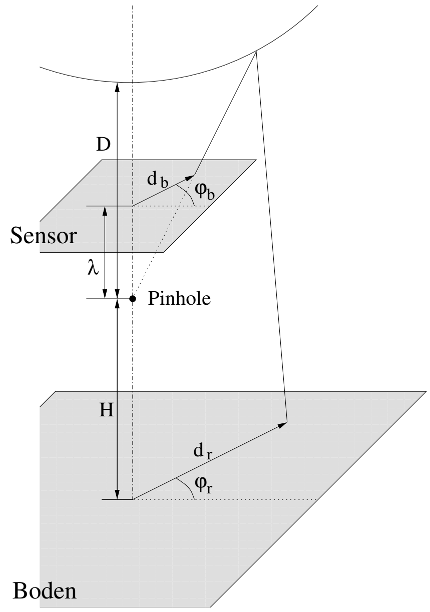


Abbildung 2.2: *Transformation zwischen Welt- und Bildkoordinaten. Die Sensorebene ist aus Gründen der Übersichtlichkeit am Pinhole gespiegelt dargestellt.*

Die Abbildung τ_d nennen wir die *Distanztransformation* der Kamera.
Um nun mit Hilfe der omnidirektionalen Kamera die Position des Punktes $P_r = (d_r, \varphi_r)$ in der Spielfeldebene zu bestimmen, wird die Position des korrespondierenden Bildpunktes $P_b = (d_b, \varphi_b)$ gemessen. Anschließend wird die Distanz d_b durch die inverse Distanztransformation τ_d^{-1} umgerechnet und man erhält:

$$\widetilde{P_r} = (\tau_d^{-1}(d_b), \varphi_b) \tag{2.2}$$

Da die Sensorebene in der Praxis nur diskrete Punkte enthält, ist die Genauigkeit der Messungen begrenzt. Daher kann $\widetilde{P_r}$ nur eine Schätzung für die tatsächliche Position des Punktes P_r sein.

Betrachten wir die Aufgaben eines einzelnen Roboters, ist für diese Schätzungen eine unterschiedliche Genauigkeit erforderlich. Eine Analyse der Teilaufgaben und Erfahrungen aus zurückliegenden Meisterschaften hat im wesentlichen zu folgenden Anforderungen geführt [Lima et al. 2001]:

(A1) Ist ein Objekt sehr nahe am Roboter, muss seine Position sehr genau geschätzt werden können. Das ist beispielsweise notwendig, um den Ball kontrollieren und zielgenau schießen zu können.

(A2) In einem Radius von einigen Metern um den Roboter ist eine gute Richtungs- und Entfernungsschätzung zu den Objekten wünschenswert. Dies gilt insbesondere für die Spielfeldlinien und Tore, da diese zur Selbstlokalisierung

Abbildung 2.3: *Aufnahme einer Omnicam mit kugelförmigem Spiegel. In der Mitte spiegelt sich das Objektiv der Kamera. Deutlich zu sehen sind auch die Stäbe, die den Spiegel in seiner Position halten.*

verwendet werden sollen. Vorteilhaft ist es, wenn der Fehler in den Distanzschätzungen unabhängig von der Entfernung ist, da das die Selbstlokalisierung vereinfacht.

(A3) Für Objekte in größerer Entfernung sollte eine gute Richtungsschätzung möglich sein; die Entfernung ist in diesem Fall weniger wichtig. Ein Beispiel ist das Anfahren eines entfernten Balls. Dazu muss hauptsächlich die Richtung bekannt sein.

(A4) Alle wichtigen Objekte, die sich auf dem Feld befinden, müssen erfasst werden können. Diese können sich nach den Regeln in bis zu 80 cm Höhe befinden. Beispiele sind die Farbmarkierungen der Roboter und die Pfosten an den Spielfeldecken.

2.3 Der Spiegel

Dem Vorschlag von *Marchese und Sorrenti* [Marchese und Sorrenti 2001] folgend, entwerfen wir nun ein Spiegelprofil, das aus drei Teilen zusammengesetzt ist[2]. Jeder dieser Teile ist dazu gedacht, eine oder mehrere der Anforderungen aus Abschnitt 2.2 zu erfüllen. Der erste Teil ist derart geformt, dass er die Spielfeldebene im Umkreis einiger Meter unverzerrt abbildet. Er erfüllt die Anforderung (A2) und soll hauptsächlich zur Selbstlokalisierung benutzt werden. Da die Sichtweite dieses Spiegels begrenzt ist, schließt sich daran ein konstant gekrümmter Spiegel

[2] Solche zusammengesetzten Spiegel, im Englischen als *multi-part-mirrors* bezeichnet, sind nicht neu. Eine einfache, häufig verwendete Form ist die Kombination aus einer Kugel und einem Konus.

an. Durch ihn werden weiter entfernte und hoch gelegene Objekte erfasst ((A3) und (A4)). Die Forderung (A1) nach einer exakten Lokalisierung von Objekten im Nahbereich wird durch den dritten Spiegelteil erfüllt. Um einen Überblick dieser drei Bereiche zu erhalten, verweisen wir auf Abbildung 2.6 auf Seite 19.

2.3.1 Isometrischer Bereich

Um die Entfernung eines Punktes auf dem Spielfeld zu schätzen, wird bei einer omnidirektionalen Kamera der Abstand des entsprechenden Bildpunktes zum Mittelpunkt des Bildes gemessen (Abbildung 2.2). Bei einem System mit einem konventionellen, z. B. sphärischen oder konischen, Spiegel ist die Distanztransformation τ_d nicht-linear, was sich als Verzerrung des Bildes auswirkt (Abbildung 2.3). Wenn das genaue Profil des Spiegels bekannt ist, lässt sich diese Verzerrung berechnen und korrigieren. Das eigentliche Problem ist aber, dass durch die Diskretisierung des Bildes ein Fehler in den Abstandsmessungen zwischen zwei Punkten entsteht, der sich nicht korrigieren lässt. Dieser Fehler ist abhängig vom Maßstab, in dem eine Strecke im Bild dargestellt wird. In einem verzerrten Bild werden weiter entfernte Strecken kleiner dargestellt, wodurch der Fehler größer wird. Für die Selbstlokalisierung ist es von Vorteil, wenn die Messungen mit einem konstanten Fehler vorliegen, damit alle Beobachtungen unabhängig von ihrer Entfernung gleich behandelt werden können.
Diese Überlegungen führen zu der Idee, die Verzerrung optisch zu kompensieren [Hicks und Bajcsy 1999, Marchese und Sorrenti 2001], indem man den Spiegel so formt, dass die Transformation τ_d linear ist. Der erwähnte Fehler ist dann durch eine Konstante beschränkt.
Die Kameratransformation τ ist in diesem Fall eine *Isometrie* zwischen den beiden zweidimensionalen euklidischen Räumen „Boden" und „Sensor", d. h. sie ist eine Abbildung, die die Längenverhältnisse und Winkel erhält. Aus diesem Grund wird ein solcher Spiegel *isometrisch* genannt.
Die Bedingung für das Profil eines isometrischen Spiegels ist in Abbildung 2.4 dargestellt. Die Entfernung zwischen zwei beliebigen Punkten X_i und X_{i+1} wird bei der Abbildung in die Sensorebene mit einem konstanten Faktor k skaliert. Daraus ergibt sich die folgende Differentialgleichung [Lima et al. 2001]:

$$\frac{\frac{X}{Y} + \frac{2Y'}{1-Y'^2}}{1 - \frac{X}{Y}\frac{2Y'}{1-Y'^2}} = \frac{\eta Y - X^2}{X(Y+H)}; \begin{cases} Y(0) = Y_0 \\ Y'(0) = 0 \end{cases} \tag{2.3}$$

wobei $Y' = dY/dX$, $\eta = k\lambda$, λ die Brennweite und H der Abstand des Pinholes vom Boden ist. Mit dieser Gleichung lässt sich das Profil des Spiegels in einem iterativen Prozess mit beliebiger Genauigkeit annähern. Dazu berechnen wir ausgehend vom Punkt $Y(0)$ und der Steigung $Y'(0)$ den nächsten Profilpunkt $Y(1)$ und die Steigung $Y'(1)$. Mit diesen Werten wiederum den Punkt $Y(2)$ usw.

Der isometrische Spiegel bildet alle Objekte in der Spielfeldebene ab, ohne sie zu verzerren. Das gilt insbesondere für die Spielfeldlinien. τ_d reduziert sich auf eine

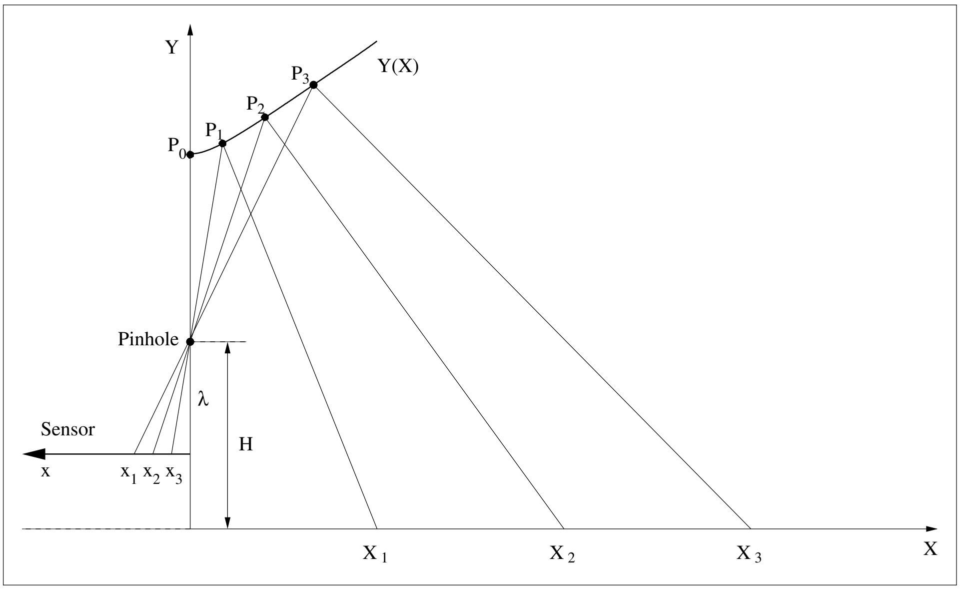


Abbildung 2.4: *Skizze zum Bestimmen der Differentialgleichung für einen isometrischen Spiegel:* $\forall i : (x_{i+1} - x_i) = k(X_{i+1} - X_i)$

einfache Multiplikation mit der Konstanten k, was eine aufwändige Kamerakalibrierung überflüssig macht.
Ein weiterer Effekt ist, dass auch das Bild des Balls, der nur wenig über die Spielfeldebene hinausragt, immer in ähnlicher Größe erscheint. Dies erleichtert die Ballerkennung, da man Fehlmessungen aussortieren kann, die stark von der erwarteten Größe abweichen.
Ideal wäre es, wenn man das gesamte Spielgeschehen mit einem isometrischen Spiegel beobachten könnte. Allerdings ist dies nicht möglich, weil der Abbildungsmaßstab dabei zu klein wäre. Aus diesem Grund legt man einen Wert d_{iso} fest, der die Sichtweite begrenzt. Der isometrische Spiegel kann somit die Zone A in Abbildung 2.5 auf der nächsten Seite erfassen. Der Punkt A und seine Steigung sind durch Formel 2.3 festgelegt.

2.3.2 Bereich mit konstanter Krümmung

Da der Sichtbereich des isometrischen Spiegels begrenzt ist, können weiter entfernt liegende Objekte nicht erfasst werden. Aber auch näher gelegene Objekte, können oben aus dem Sichtbereich herausragen. Aus diesem Grund wird das Spiegelprofil durch einen zweiten Teil fortgesetzt, der zwar kein unverzerrtes Bild mehr liefert, aber ein Erfassen dieser Objekte ermöglicht.
Abbildung 2.5 ist eine Skizze zum Entwurf dieses Spiegelabschnitts.
Zone A ist der Bereich des isometrischen Spiegels. Der zweite Teil des Spiegels soll Zone B erfassen, damit bis zu einer maximalen Entfernung d_{max} alle

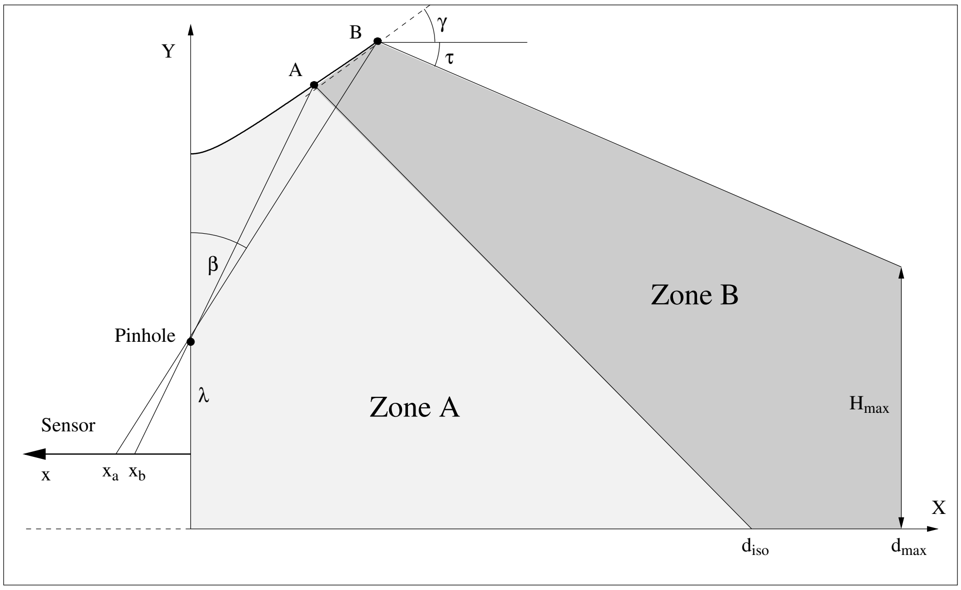


Abbildung 2.5: *Skizze des Spiegels mit zwei Bereichen: Zone A wird vom isometrischen Teil erfasst, in Zone B liegen weiter entfernte oder höher gelegene Objekte, wie beispielsweise die Farbmarkierungen der Roboter.*

Objekte in der Höhe H_{max} gesehen werden können. Um Störungen am Übergang zwischen den Zonen (Punkt A) zu vermeiden, wird verlangt, dass die beiden Spiegelteile in diesem Punkt eine gemeinsame Tangente haben. Eine weitere Bedingung ergibt sich, indem wir die Werte H_{max} und d_{max} festlegen und den Punkt $B = (X_B, Y_B)$ wählen, der die Größe des neuen Spiegelabschnitts definiert. Die Steigung in Punkt B ist durch die folgenden Gleichungen festgelegt [Lima et al. 2001]:

$$\tan(\tau) = \frac{(H_{max} - Y_B)}{(d_{max} - X_{B)}}, \ \tan(\beta) = \frac{x_b}{\lambda}, \ \tan(\gamma) = \tan(\frac{(\beta + \tau - \pi)}{2}). \quad (2.4)$$

Außer den Punkten A und B und den zugehörigen Steigungen sind keine weiteren Bedingungen gegeben. Am einfachsten ist es daher, zwischen diesen Punkten eine gleichmäßige Änderung der Steigung, eine konstante Krümmung, anzunehmen. Damit lassen sich auf die gleiche Weise wie beim isometrischen Spiegel, alle Profilpunkte zwischen A und B berechnen.
Die Entfernungsschätzungen in diesem Spiegelbereich sind ungenauer, doch gute Richtungsschätzungen sind immer noch möglich. Damit sind die Anforderungen (A3) und (A4) erfüllt.

2.3.3 Bereich für Nahsicht

Der Spiegel, der bisher beschrieben wurde, hat einen Nachteil: Wegen der starken Verkleinerung durch den isometrischen Spiegel wird der Ball nur auf wenige Pixel[3] abgebildet und seine Position lässt sich sehr schlecht schätzen. Um aber den Ball während des Spiels genau und stark schießen zu können, muss der Roboter entscheiden können, ob der Ball an der Schussvorrichtung anliegt. Ist er nur wenige Zentimeter zu weit entfernt, geht der Schuss ins Leere.
Aus diesem Grund wird ein dritter Teil an das Spiegelprofil angefügt. Dieser befindet sich am Rand des Spiegels und bildet die direkte Umgebung des Roboters ab. In Abbildung 2.6 ist dies dargestellt: der Bereich zwischen den Punkten B und C spiegelt die Zone C vom Rumpf des Roboters bis zu einer Entfernung d_{nah} wider. Ein kleiner Bereich direkt am Roboter bleibt immer vom Roboterrumpf verdeckt, so dass der Boden erst ab einer Entfernung d_{min} sichtbar ist.
Durch Festlegen des Punktes C und der Entfernungen d_{min} und d_{nah} ergeben sich analog zur Berechnung des zweiten Spiegelabschnitts die Tangenten in den Punkten B und C. Dazwischen wird wiederum eine konstante Variation der Steigung angenommen, um die fehlenden Punkte berechnen zu können.

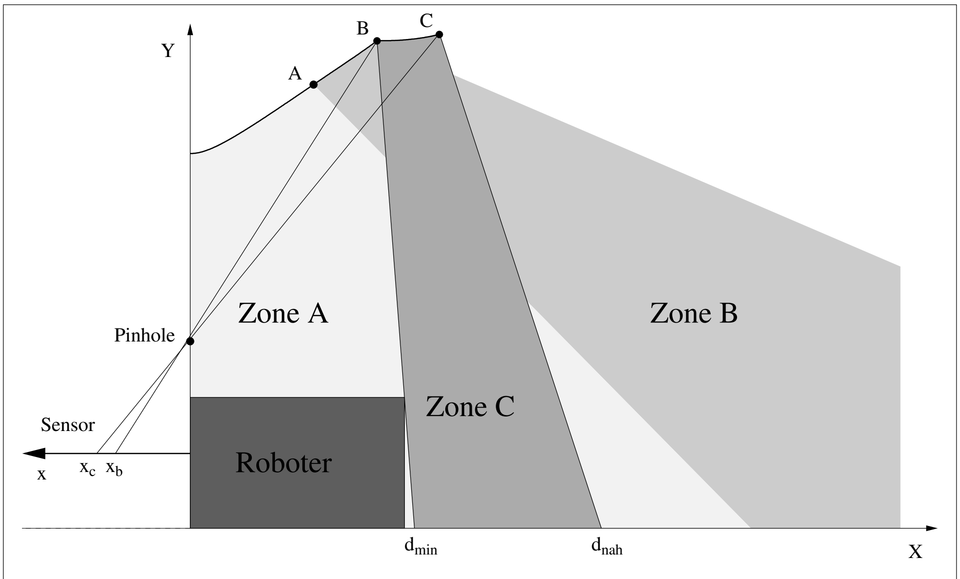
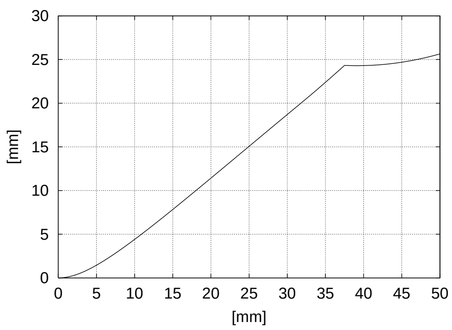


Abbildung 2.6: *Skizze des Spiegels mit drei Bereichen: Der Spiegelbereich zwischen den Punkten B und C erfasst den Bereich vom Roboter bis zu einer Entfernung d_{nah} (Zone C)*

[3] Bei dem nach unseren Spezifikationen gefertigten Spiegel sind es etwa 5x5 Pixel.

2.3.4 Festlegen der Parameter

Bevor wir mit diesen Spezifikationen einen Spiegel herstellen können, müssen wir die verschiedenen Parameter festlegen.
Da wir unsere CCD-Kamera, eine Sony DFW-V500, und das vorhandene Objektiv weiter verwenden wollen, ist die Brennweite auf $\lambda = 8$ mm festgelegt. Die Kamera wird so auf den Roboter montiert, dass sich das Pinhole – in diesem Fall eigentlich das Objektiv – in der Höhe $H = 60$ cm befindet. Der Spiegel wird 14 cm darüber mit einem Zylinder aus Plexiglas befestigt. Aus material-technischen Gründen ist der Radius des Spiegels auf 5 cm begrenzt. Damit ist also der Wert X_C bestimmt.
Im isometrischen Teil sollen Entfernungen bis zu $d_{iso} = 6$ m abgebildet werden. Im Bild reservieren wir für diesen Bereich 65% vom maximalen Radius. Mit Hilfe der Größe des Sensors unserer Kamera lässt sich daraus leicht die Konstante k für den isometrischen Spiegel berechnen und es ergibt sich der Punkt A. Mit dem zweiten Spiegelabschnitt sollen Objekte in einer Höhe von $H_{max} = 0,6$ m in bis zu $d_{max} = 10$ m Entfernung sichtbar sein. Dabei wird der Punkt B so festgelegt, dass dieser Bereich etwa 10% des Bildes belegt. Die äußeren 25% des Bildes werden für den Nahbereich verwendet, der die Umgebung des Roboters zwischen $d_{min} = 20$ cm und $d_{nah} = 80$ cm abbilden soll.
Nach diesen Angaben wurde von uns das Profil berechnet, das in Abbildung 2.7(a) zu sehen ist. Anschließend wurde ein Prototyp des Spiegels in der mechanischen Werkstatt des Instituts für Mikrosystemtechnik an der Universität Freiburg auf einer Ultra-Präzisions-Fräse gefertigt. Als Material wurde die Aluminiumlegierung *AlCuMgPb* verwendet, die sich durch ein geringes Gewicht und gute Reflexionseigenschaften im gesamten Lichtspektrum auszeichnet. Das geringe Gewicht ist deshalb von Bedeutung, weil der Spiegel sehr hoch angebracht ist. Dadurch verlagert sich der Schwerpunkt des Roboters nach oben und er wird bei starken Beschleunigungen instabil. Der fertige Spiegel ist in Abbildung 2.7(b) zu sehen.

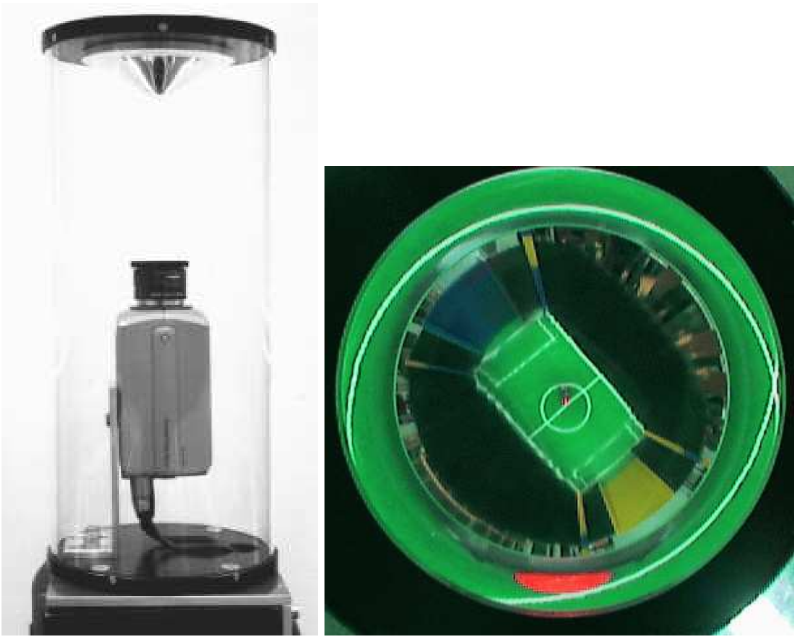


(a)

(b)

Abbildung 2.7: *Das berechnete Spiegelprofil* (a) *und der fertige Spiegel* (b)

Die Halterungen für Spiegel und Kamera wurden ebenfalls in der Werkstatt des Instituts für Mikrosystemtechnik hergestellt. In Abbildung 2.8(a) ist die fertig aufgebaute omnidirektionale Kamera gezeigt. In Teil (b) der Abbildung sieht man eine Aufnahme, die mit dieser Kamera gemacht wurde. Im isometrischen Bereich in der Mitte werden die Spielfeldlinien unverzerrt abgebildet. Die Tore und Eckpfosten, werden allerdings verformt. Im unteren Teil des Bildes sieht man im Nahbereich einen Ball, der sich direkt vor dem Roboter befindet.



(a) (b)

Abbildung 2.8: *Die fertige omnidirektionale Kamera* (a) *und eine Aufnahme durch die Kamera* (b).

Abbildung 2.9 zeigt das gleiche Bild in unserer graphischen Oberfläche. Das Zentrum des Bildes und die verschiedenen Spiegelbereiche sind markiert. Der Abbildungsmaßstab des isometrischen Spiegels ist auf 55 mm/Pixel eingestellt. Das heißt, eine Strecke von einem Meter wird im Kamerabild auf ca. 18,2 Pixel abgebildet. Dies entspricht in etwa unseren Spezifikationen. Durch Veränderungen in der Höhe der Kamera und Ungenauigkeiten im Abstand zwischen Kamera und Spiegel kann dieser Wert jedoch leicht variieren.
Eine Fotografie des Roboters, der mit der omnidirektionalen Kamera ausgerüstet ist, ist in Abbildung 5.1 auf Seite 69 zu sehen.

Abbildung 2.9: *Graphische Oberfläche zum Einstellen der Kameraparameter*

Kapitel 3

Merkmalsextraktion

In diesem Kapitel wird beschrieben, wie aus dem Kamerabild Informationen gewonnen werden, die für ein RoboCup-Spiel relevant sind. Dieser Vorgang wird als Merkmalsextraktion bezeichnet.
Im RoboCup-Regelwerk [RoboCup 2002] ist für alle Objekte auf dem Spielfeld eine Farbe festgelegt. Diese Farben sind in Abbildung 3.2 zusammengefasst. Da sich diese Arbeit mit der Selbstlokalisierung eines Roboters befasst, sind in diesem Fall die statischen Objekte auf dem Spielfeld interessant; dies sind die Tore, die Eckpfosten und die Feldlinien. Um diese Merkmale aus dem Kamerabild zu extrahieren, sind verschiedene Verarbeitungsschritte erforderlich. Zuerst wird jedes Pixel des Bildes einer *Farbklasse* zugeordnet. Anschließend werden Regionen gleicher Farbe zu sogenannten *Blobs* zusammengefasst. Außerdem werden Farbübergänge zwischen grünen und weißen Pixeln gesucht, um den Verlauf der Feldlinien zu bestimmen. In Abbildung 3.1 ist eine Übersicht über den Ablauf dargestellt. Die einzelnen Schritte werden im Folgenden näher erläutert.

3.1 Farb-Segmentierung

Der erste Schritt der Bildverarbeitung ist die Farb-Segmentierung des Bildes. Dabei wird jedem Pixel eine oder mehrere der Farbklassen *Rot*, *Gelb*, *Grün*, *Cyan*, *Blau*, *Magenta*, *Weiß* oder *Schwarz* zugeordnet. Diese Farbklassen ergeben sich direkt aus den Farben, die in den Regeln des RoboCup festgelegt sind. Der Ball ist rot (oder orange), das Feld ist grün mit weißen Linien, die Roboter sind schwarz mit cyan-farbigen Markierungen für das eine und magenta-farbigen Markierungen für das andere Team, die Tore sind gelb bzw. blau und die Eckpfosten sind auf der einen Seite gelb-blau-gelb und auf der gegenüberliegenden Seite blau-gelb-blau (vgl. Abbildung 3.2).
Bevor nun der Vorgang der Klassifizierung der Pixel beschrieben wird, sollen kurz einige Grundlagen der Farbdarstellung erläutert werden.

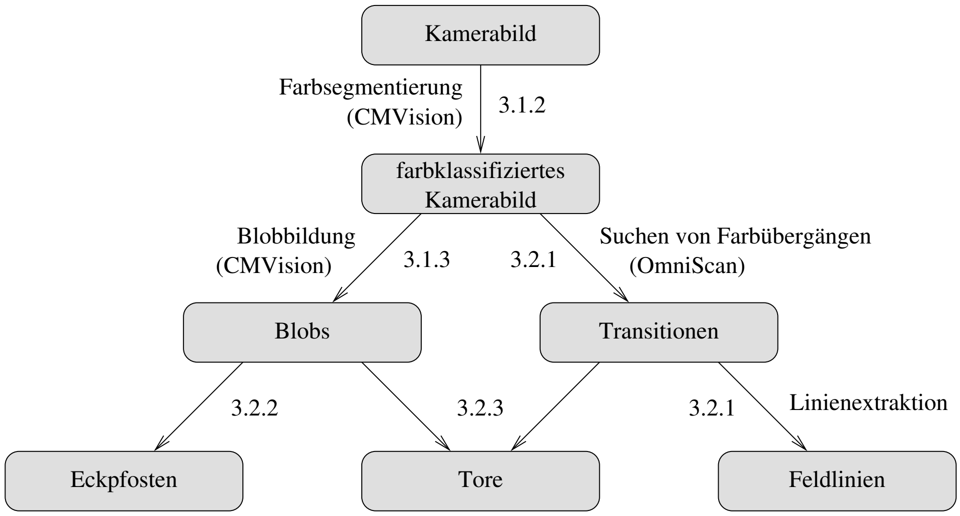


Abbildung 3.1: *Ablauf der Bildverarbeitung. Die Zahlen geben die Abschnitte dieser Arbeit an, in denen der jeweilige Prozess beschrieben ist.*

3.1.1 Farbdarstellungen

Die *Dreifarbentheorie* von *Thomas Young* aus dem Jahr 1802 besagt folgendes:

> Jede sichtbare Farbe lässt sich aus drei Primärfarben durch additive Farbmischung herstellen.

Diese Theorie ist begründet durch die Anatomie des menschlichen Auges. Die Netzhaut enthält verschiedene Rezeptoren. Neben den sogenannten Stäbchen, die für das Hell-Dunkel-Sehen verantwortlich sind, gibt es drei Arten von Zapfen für das Farbsehen. Diese nehmen unterschiedliche Bereiche des Lichtspektrums wahr und haben ihre maximale Empfindlichkeit im roten, grünen bzw. blauen Bereich. Jede für den Menschen wahrnehmbare Farbe lässt sich daher als Kombination dieser drei Signale beschreiben.

Mathematisch ausgedrückt besagt die Dreifarbentheorie, dass sich jede Farbe X als Linearkombination der drei Grundfarben Rot ($\mathcal{R}$), Grün ($\mathcal{G}$) und Blau ($\mathcal{B}$) darstellen lässt:

$$X = R\mathcal{R} + G\mathcal{G} + B\mathcal{B} \tag{3.1}$$

Der Raum der sichtbaren Farben besitzt also drei Dimensionen. Eigentlich sind als Basisvektoren beliebige linear unabhängige Farben zulässig. 1931 wurden von der CIE (Commission International de l'Eclairage) die Standardwerte Rot, Grün und Blau mit bestimmten Wellenlängen festgelegt [Burkhardt 1999].

RGB

Die Beschreibung der Farben durch ihre Rot-, Grün- und Blau-Werte aus Gleichung 3.1 wird als RGB-System bezeichnet. Diese Darstellung wird vor allem bei Monitoren und Fernsehern verwendet, wobei die Werte für R, G und B auf das Intervall $[0; 1]$ beschränkt werden. Dadurch ergibt sich der RGB-Würfel, der in Abbildung 3.2 zu sehen ist. Die für den RoboCup festgelegten Farben entsprechen nicht zufällig den acht Ecken dieses Würfels, denn dadurch sind die Unterschiede zwischen den Farben maximal und sie lassen sich leicht unterscheiden.

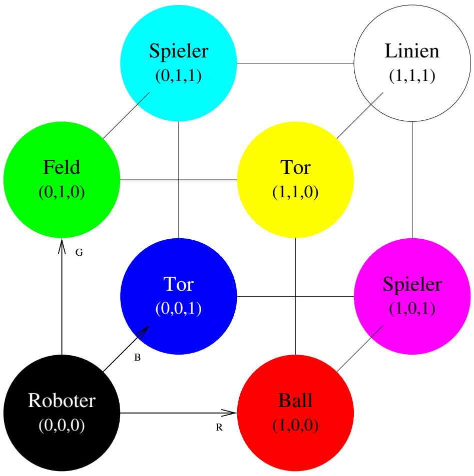


Abbildung 3.2: *Der Farbraum, den die drei Grundfarben Rot, Grün und Blau aufspannen, wird als RGB-Würfel bezeichnet. Die für eine RoboCup-Umgebung festgelegten Farben entsprechen den acht Ecken dieses Würfels.*

Je nach Farbtiefe der Anzeige steht eine unterschiedliche Anzahl an Bits zur Codierung der einzelnen Farbwerte zur Verfügung. Bei einer 16-Bit-Darstellung werden fünf Bit für Rot, sechs Bit für Grün und wiederum fünf Bit für Blau verwendet. Bei 24-Bit stehen jeweils acht Bit zur Verfügung. Bei einer 32-Bit-Farbtiefe werden zusätzlich acht Bit zur Codierung des sogenannten *Alpha-Kanals* verwendet, der die Transparenz der Farbe angibt. Dieser Wert beschreibt keine Eigenschaft der Farbe selbst, sondern kann dazu genutzt werden, durchscheinende Effekte oder Überblendungen auf einem Monitor zu erzeugen. In diesem Fall spricht man von der *RGBA-Darstellung*.
Um ein Bild auf einem Monitor darstellen zu können, muss es in das RGB-Format umgewandelt werden, das der Farbtiefe entspricht, die auf dem jeweiligen System eingestellt ist.

YUV

Eine andere Farbdarstellung, die in der Video-Übertragung eingesetzt wird, ist das *YUV-System*. Hierbei werden die Farben durch die Luminanz Y, die die Helligkeit angibt, und zwei Chrominanz-Werte U und V beschrieben, die die Farbinformation enthalten. Der YUV-Wert einer Farbe ergibt sich aus dem RGB-Wert durch eine lineare Transformation gemäß der folgenden Gleichung:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ -0,148 & -0,289 & 0,437 \\ 0,615 & -0,515 & -0,100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.2}$$

In den Luminanzwert Y gehen die Farbanteile mit unterschiedlicher Gewichtung ein. Die Koeffizienten entsprechen dabei dem Helligkeitsempfinden des Menschen, welches sich zu etwa 30% aus dem Rot-Anteil, zu 59% aus dem Grün-Anteil und zu 11% aus dem Blau-Anteil zusammensetzt.

Das YUV-Format wurde zur Übertragung des Farbfernsehens entwickelt und hat zweierlei Vorteile:

- Der Luminanz-Wert Y beschreibt das Bild in Graustufen. Indem die Chrominanz-Werte U und V in einem anderen Frequenzband übertragen werden, das von S/W-Fernsehern nicht dargestellt wird, erreicht man eine Abwärtskompatibilität des Farbfernsehformats.
- Das menschliche Auge reagiert viel empfindlicher auf Fehler in der Helligkeit als auf Fehler in der Farbe. Die Chrominanz-Werte U und V können daher mit geringerer Bandbreite übertragen werden als die Luminanz Y.

Auch in der digitalen Videotechnik wird davon Gebrauch gemacht, dass der Mensch weniger empfindlich auf Farbunterschiede reagiert, indem man die Farbinformationen in geringerer Auflösung darstellt. Beim Format *YUV 4:1:1* werden für jeden 2x2-Bildausschnitt vier Helligkeitswerte, aber nur jeweils ein Wert für U und V gespeichert. Analog dazu werden beim *YUV 4:2:2-Format* jeweils zwei benachbarte Pixel durch gemeinsame Chrominanz-Werte beschrieben. Dadurch wird die Datenmenge auf 2/3 komprimiert. Ein Bild mit n Bildpunkten wird durch die folgende Bytefolge dargestellt:

U_0	Y_0	V_0	Y_1	U_2	Y_2	V_2	Y_3	$\cdots$	U_{n-2}	Y_{n-2}	V_{n-2}	Y_{n-1}

Dabei beschreiben jeweils vier aufeinander folgende Werte zwei benachbarte Pixel des Bildes.
Bei der Mustererkennung ergibt sich ein weiterer Vorteil des YUV-Systems: Die Farbdarstellung reagiert weniger sensibel auf Beleuchtungsänderungen. Eine Änderung der Helligkeit wirkt sich hauptsächlich auf den Y-Wert aus, wodurch sich die Farbsegmentierung vereinfacht.

Unsere Sony DFW-V500 Kamera kann Bilder in verschiedenen Formaten liefern. Neben Bildern in RGB, YUV 4:1:1 und YUV 4:2:2 gibt es auch einen Monochrom-Modus. Für diese Arbeit haben wir uns für das YUV 4:2:2-Format bei einer Auflösung von 640x480 Pixeln entschieden, als einen guten Kompromiss zwischen Detailtreue und Datenmenge. Im Folgenden bezeichnet der Ausdruck *Farbraum* daher immer den YUV-Raum.

Andere Farbsysteme

Neben den erwähnten RGB- und YUV-Formaten existieren in der Bildverarbeitung noch zahlreiche andere Farbsysteme, zum Beispiel das *Normalfarbensystem XYZ* und der *farbmetrische Raum UVW*. Erwähnenswert ist noch der *HSV-Farbraum*, der sich durch gute Anschaulichkeit auszeichnet und darum in vielen Computeranwendungen zur Farbauswahl verwendet wird. In ihm wird eine Farbe durch die drei Werte Hue (Farbton), Saturation (Sättigung) und Value (Helligkeit) beschrieben [Burkhardt 1999].

3.1.2 CMVision

Zur Farbklassifizierung eines Pixels gibt es verschiedene Ansätze:

Die NEAREST NEIGHBOUR KLASSIFIZIERUNG benutzt eine Menge von zuvor klassifizierten Beispielen. Um ein neues Pixel zu klassifizieren, werden die nächstliegenden Beispiele betrachtet und danach entschieden, zu welcher Klasse das Pixel gehört.

Beim LINEAR THRESHOLDING wird der Farbraum durch lineare Funktionen, in diesem Fall also Ebenen, partitioniert. Ein Pixel wird der Klasse zugeordnet, in deren Partition es fällt. Realisiert wird eine solche Klassifizierung beispielsweise durch ein Künstliches Neuronales Netz.

Das in dieser Arbeit verwendete Verfahren ist das CONSTANT THRESHOLDING. Dabei wird eine Farbklasse durch sechs Schwellenwerte $\mathtt{Y}_{\mathtt{min}}$, $\mathtt{Y}_{\mathtt{max}}$, $\mathtt{U}_{\mathtt{min}}$, $\mathtt{U}_{\mathtt{max}}$, $\mathtt{V}_{\mathtt{min}}$, $\mathtt{V}_{\mathtt{max}}$ beschrieben. Diese definieren einen achsenparallelen Quader im dreidimensionalen Farbraum. Alle Pixel, deren Farbwerte innerhalb des Quaders liegen, werden dieser Klasse zugeordnet. Der Vorteil dieser Methode ist die gute Performanz. Nachteilig ist aber, dass keine Abhängigkeiten zwischen den einzelnen Farbdimensionen modelliert werden können.

Der naive Ansatz zur Klassifizierung eines Pixels `YUV` ist:

```
if ((Y >= Ymin) AND
    (Y <= Ymax) AND
    (U >= Umin) AND
    (U <= Umax) AND
```

```
  (V >= V_min) AND
  (V <= V_max))
 pixel_color = color_class;
```

Diese Code-Sequenz kann bei der Ausführung sechs bedingte Sprünge erfordern. Besonders auf einem Pipeline-Rechner mit Sprungvorhersage ist das unter Umständen sehr ineffizient. Deshalb wird in der *Color Machine Vision* Bibliothek (kurz *CMVision*) von *James Bruce* ein anderer Ansatz benutzt [Bruce 2000]. Die dreidimensionale Farbklasse wird als Produkt von drei eindimensionalen boolwertigen Funktionen dargestellt, wie in Abbildung 3.3 veranschaulicht.

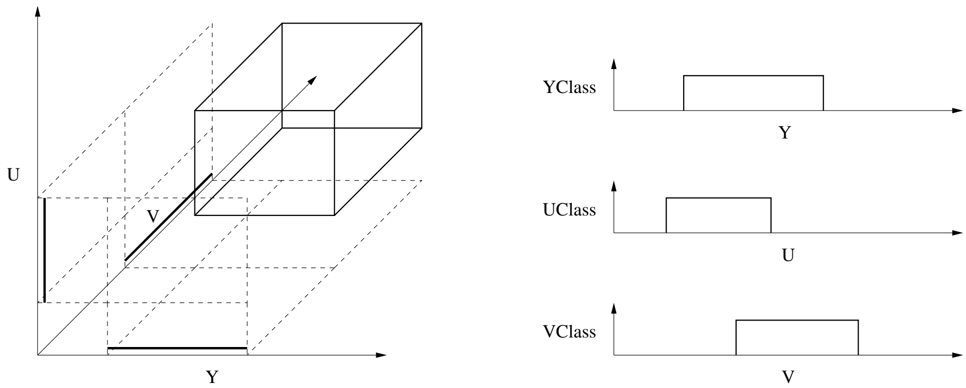


Abbildung 3.3: *Binäre Dekomposition einer Farbklasse: Ein dreidimensionaler Ausschnitt des Farbraums (links) wird durch drei binäre Funktionen repräsentiert (rechts).*

Die Teilfunktionen werden als Arrays gespeichert, gemäß der Diskretisierung des Farbraums. Die Entscheidung, ob ein Pixel zu einer Farbklasse gehört, reduziert sich dann auf das `AND` der Array-Elemente, die durch die Farbwerte des Pixels bestimmt werden:

```
pixel_in_class = YClass[Y] AND
                 UClass[U] AND
                 VClass[V];
```

Ein weiterer nennenswerter Vorteil dieser Herangehensweise ist, dass die Zugehörigkeit eines Pixels zu mehreren Farbklassen gleichzeitig entschieden werden kann. Indem für jedes Array-Element der Typ Integer benutzt wird und an jeder Bit-Position eine andere Farbklasse gespeichert wird, kann auf einem 32-Bit-Rechner mit nur zwei `AND`-Operationen die Zugehörigkeit eines Pixels zu 32 Farbklassen überprüft werden. Für eine genauere Erläuterung und ein Beispiel verweisen wir auf die Arbeit von *Bruce* [Bruce 2000].

Wir benutzen die CMVision-Bibliothek zur Klassifizierung der Kamerabilder. Dabei werden die Pixel einer oder mehreren der Farbklassen *Red*, *Cyan*, *Magenta*, *Yellow*, *Blue*, *Green* und *White* zugeordnet. Zum Einstellen der Schwellenwerte der Farbklassen benutzen wir ein Werkzeug, das bei der CMVision-Bibliothek mitgeliefert wird und von uns mit einer graphischen *QT-Oberfläche* versehen wurde. Das Werkzeug erstellt ein Histogramm über die Pixel des Kamerabildes und projiziert dieses auf die drei Farbebenen *UV*, *YU* und *YV*. Die Schwellenwerte lassen sich durch Rechtecke in diesen drei Ebenen markieren. Dies ist in Abbildung 3.4 für die Farbklasse *Yellow* dargestellt. Bei unterschiedlicher Beleuchtung der Umwelt verändern sich die Farben im Kamerabild. Deshalb müssen bei einer Änderung der Beleuchtung die Schwellenwerte manuell neu eingestellt werden. Dies kann ein sehr mühsamer Vorgang sein. Abbildung 3.5 zeigt eine Aufnahme der omnidirektionalen Kamera und dasselbe Bild nach der Farbsegmentierung durch CMVision.

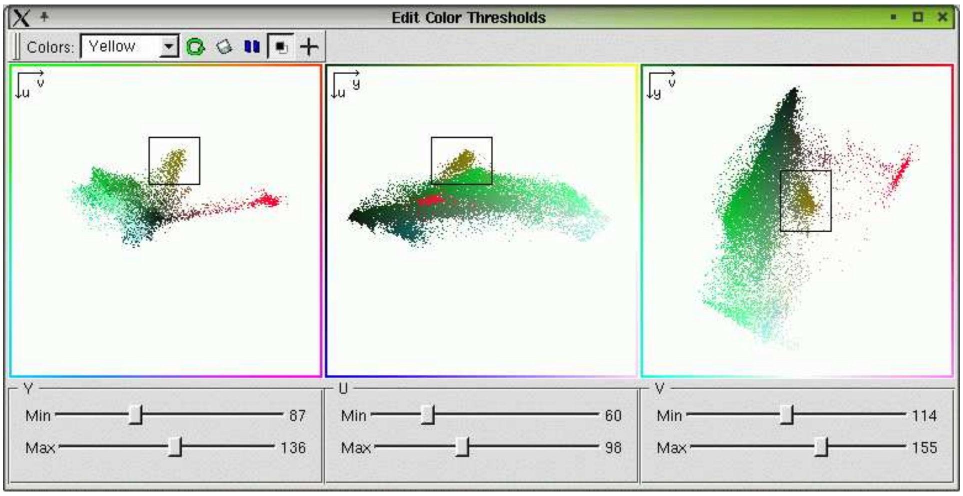


Abbildung 3.4: *Graphische Oberfläche zum Festlegen der Farbklassen: Ein Farb-Histogramm des Kamerabildes wird auf die drei Farbebenen projiziert. Durch Rechtecke werden darin die Schwellenwerte einer Farbklasse festgelegt.*

3.1.3 Blob-Bildung

Um einen Informationsgewinn aus dem farb-klassifizierten Bild zu erhalten, werden in einem weiteren Schritt Regionen von Pixeln gleicher Farbe gebildet, sogenannte *Blobs*. *Gleiche Farbe* bedeutet, dass sie derselben Farbklasse zugeordnet worden sind. Auch für dieses Problem bietet *CMVision* eine effiziente Implementierung, die in zwei Stufen abläuft. Zuerst werden horizontal nebeneinander liegende Pixel gleicher Farbe zu sogenannten *„Runs"* zusammengefasst. Anschließend werden die „Runs", die sich vertikal überlappen, mit Hilfe einer *Union-Find-Struktur* zu Regionen verschmolzen [Bruce 2000].
Das Ergebnis dieses Vorgangs sind Bereiche von Pixeln, die der gleichen Farbklas-

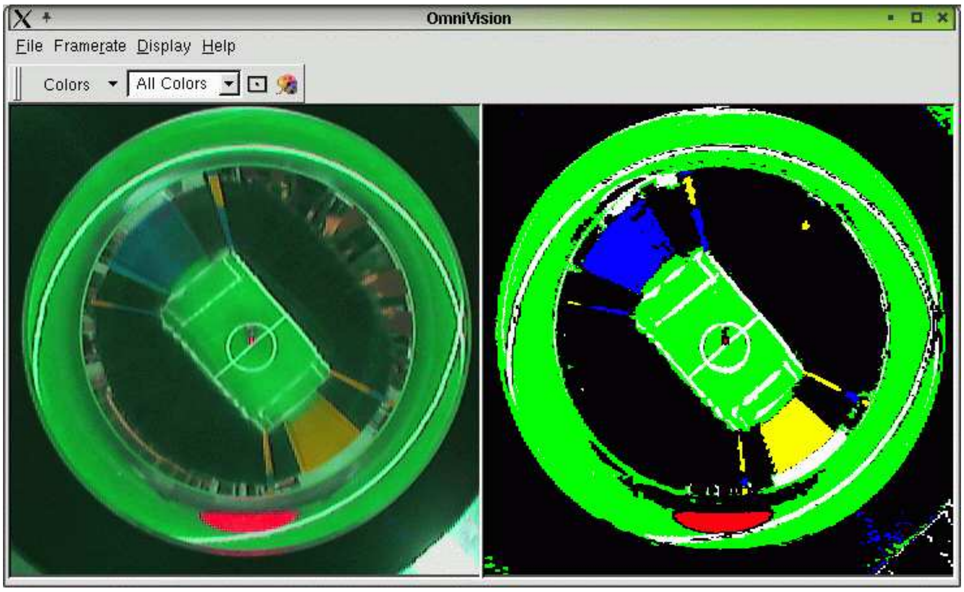


Abbildung 3.5: *Eine Aufnahme der Omnicam (links) und das farbklassifizierte Bild (rechts)*

se zugeordnet sind und unter einer Vierernachbarschaft[1] miteinander verbunden sind. Zu diesen werden nun noch ihre Bounding-Box und ihr Flächen-Schwerpunkt berechnet. Diese Werte werden an die nächste Schicht der Mustererkennung weitergegeben.

3.2 Extraktion der High-Level-Merkmale

In diesem Abschnitt wird beschrieben, wie aus dem durch Farb-Segmentierung aufbereiteten Kamerabild die relevanten Merkmale des Spielfelds extrahiert werden.

3.2.1 Feldlinien

Die weißen Feldlinien stellen wichtige Merkmale zur Standortbestimmung auf einem Fußballfeld dar. Da sie über das ganze Feld verteilt sind, ist fast immer eine Linie zur Orientierung in der Nähe und es ist unwahrscheinlich, dass alle Linien von Hindernissen verdeckt sind, was bei anderen Merkmalen durchaus vorstellbar ist. Da außerdem das Profil unseres Omnicam-Spiegels darauf ausgelegt ist, die Linien unverzerrt und mit konstantem Fehler abzubilden (vgl. Abschnitte 2.2 und

[1] *Vierernachbarschaft* heißt, diagonal nebeneinanderliegende Pixel werden nicht als benachbart angesehen. Im Gegensatz dazu gibt es die *Achternachbarschaft*, bei der dies der Fall ist.

2.3), wurde besonderer Wert auf eine effiziente und zuverlässige Erkennung der Feldlinien gelegt.

Das Erkennen der Feldlinien läuft in mehreren Schritten ab. Zuerst wird eine Menge $\mathcal{T}$ von Punkten erzeugt, die möglicherweise auf einer Feldlinie liegen. Gute Kandidaten für solche Punkte sind Pixel, in denen ein Farbübergang zwischen Grün und Weiß stattfindet. Diese Punkte bezeichnen wir als *Transitionen*. Als nächstes wird aus der Menge $\mathcal{T}$ eine Menge $\mathcal{L}$ von Linien extrahiert. Diese Linien können anschließend noch einem Plausibilitätstest unterzogen werden.

Erster Ansatz: Kantendetektion und Hough-Transformation

Die KANTENDETEKTION ist ein klassisches Verfahren der Bildanalyse. Aus dem Kontrast zwischen benachbarten Pixeln werden Grenzen zwischen Bildobjekten bestimmt. Üblicherweise wird die Kantendetektion mit Hilfe von *linearen Lokaloperatoren* realisiert. Am bekanntesten sind dabei die *Sobel-Operatoren* [Burkhardt 1999]:

$$\mathrm{SOB}_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \mathrm{SOB}_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Zum Detektieren einer horizontalen Kante wird die Matrix SOB_X über einem Bildpunkt zentriert. Die umliegenden Pixel werden mit den entsprechenden Gewichten der Matrix multipliziert und aufsummiert. Dadurch ergibt sich der Wert Δ_X. Entsprechend wird der Wert Δ_Y mit Hilfe der Matrix SOB_Y für vertikale Kanten berechnet. Anschließend werden diese beiden Werte verknüpft, zum Beispiel durch die Summe der Beträge: $\Delta = |\Delta_X| + |\Delta_Y|$. Liegt Δ über einem festgelegten Schwellenwert, wird die betrachtete Stelle als Kante markiert.

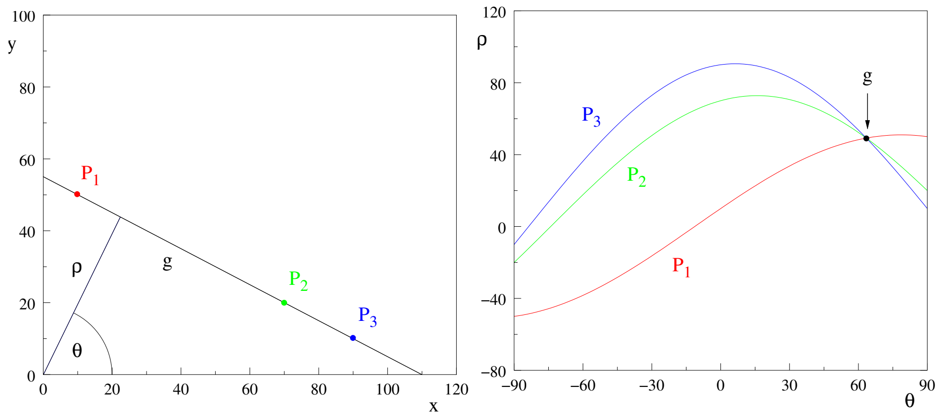
Um Rechenzeit zu sparen, bietet es sich beim YUV-Format an, zunächst nur die Y-Komponente zur Kantendetektion zu verwenden, da diese den größten Informationsgehalt besitzt. In Abbildung 3.7(a) ist ein Bild zu sehen, das in dieser Weise gefiltert wurde. Die Feldlinien treten deutlich hervor, allerdings auch noch viele andere Objektkanten, da noch keinerlei Farbinformationen verarbeitet wurden. Um die Punktmenge weiter zu reduzieren, müssen noch die Farbwerte der umliegenden Punkte betrachtet werden.

Aus der so erzeugten Transitionsmenge $\mathcal{T}$ sollen nun gerade Linien extrahiert werden. Dazu bietet sich die HOUGH-TRANSFORMATION an [Jähne 1997]. Grundlage der Hough-Transformation ist es, dass sich jede beliebige Gerade des Datenraumes $\mathcal{D} \subseteq \mathbb{R}^2$ in ihrer Normalenform darstellen lässt:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{3.3}$$

Dabei ist θ der Normalenwinkel und ρ der rechtwinklige Abstand vom Ursprung (siehe Abbildung 3.6(a)). Der zweidimensionale Raum $\mathcal{M}$, der durch ρ und θ aufgespannt wird, ist der *Modellraum* der Hough-Transformation. In ihm wird durch

jeden Punkt $(\rho, \theta) \in \mathcal{M}$ eine Gerade im Datenraum definiert. Umgekehrt wird durch jeden Punkt $(x, y) \in \mathcal{D}$ eine Menge von Punkten im Modellraum definiert, nämlich die Menge aller Geraden, die durch den Punkt (x, y) gehen. Diese Abbildung von Punkten des Datenraums in den Modellraum wird als Hough-Transformation bezeichnet.
Liegen verschiedene Punkte im Datenraum auf einer Geraden, so schneiden sich ihre Transformierten im Modellraum in einem Punkt (siehe Abbildung 3.6).

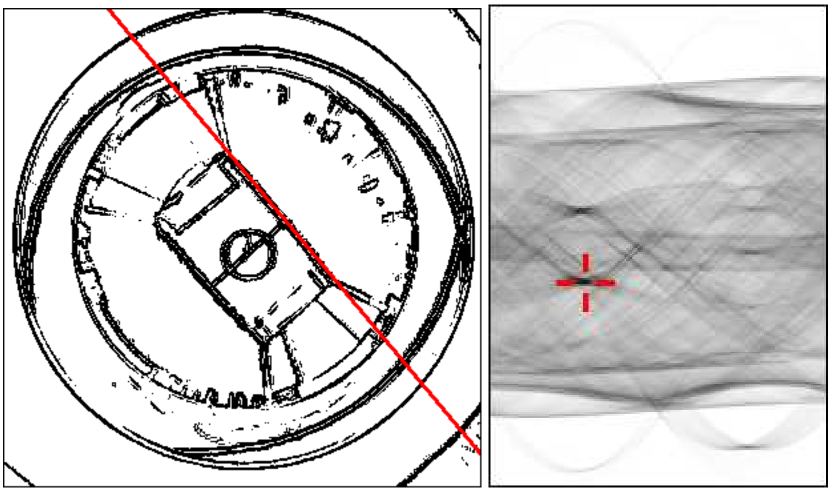


(a) (b)

Abbildung 3.6: *Der Datenraum* (a) *und der Modellraum* (b) *der Hough-Transformation.*

Diese Tatsache kann man zur Extraktion von Linien aus einer Punktmenge $\mathcal{T}$ ausnutzen:
Zunächst initialisiert man eine diskrete Version des Modellraums, den sogenannten *Akkumulator*. Anschließend wird für jeden Punkt $P_i \in \mathcal{T}$ die Hough-Transformation berechnet und die entsprechenden Zellen des Akkumulators werden inkrementiert. Danach werden die Maximalstellen im Akkumulator bestimmt. An diesen Stellen schneiden sich die meisten Hough-Transformierten, das heißt, sie entsprechen Geraden im Datenraum auf denen besonders viele Punkte der Menge $\mathcal{T}$ liegen.

In Abbildung 3.7 ist der Vorgang beispielhaft dargestellt. Das Kamerabild wurde mit dem Sobel-Operator gefiltert (links) und anschließend in den Hough-Modellraum transformiert (rechts). Das Maximum des Akkumulators (siehe Markierung) definiert die eingezeichnete Gerade im Kamerabild.
Die Hough-Transformation lässt sich in ähnlicher Weise auch zur Detektion von anderen geometrischen Formen einsetzen. Sie ist also auch ein geeignetes Mittel zum Erkennen des Mittelkreises.



(a) (b)

Abbildung 3.7: *Ergebnis der Kantendetektion mit dem Sobel-Operator* (a) *und die zugehörige Hough-Transformation* (b).

Allerdings haben wir diesen Ansatz zur Bestimmung der Feldlinien, die Kombination aus Kantendetektion und Hough-Transformation verworfen. Beide Algorithmen benötigen in der Praxis lange Berechnungszeiten und die Genauigkeit der Hough-Transformation ist durch die Auflösung des Akkumulators begrenzt.

Zweiter Ansatz: OmniScan

Die Idee des zweiten Ansatzes ist es, bestimmte Transitionen zwischen grünen und weißen Pixeln im farb-klassifizierten Bild zu suchen. Auf Grund des Vorgehens beim Suchen erhält man auf den Transitionen eine Ordnung: sie sind gemäß den Winkeln ihrer Polarkoordinaten-Darstellung sortiert. Dadurch kann eine effiziente Linienextraktion benutzt werden, die sich schon zum Detektieren von Liniensegmenten in den Daten eines Laser-Scanners bewährt hat [Gutmann 2000]. Auf Grund der Verwendung dieses Verfahrens für unsere omnidirektionale Kamera und den Parallelen zu Laser-Scanner-Anwendungen nennen wir sowohl den zugehörigen Algorithmus als auch die entstehende Punktmenge einen *OmniScan*.

Bestimmung der Transitionen: Die Suche nach den Transitionen läuft wie folgt ab: Vom Zentrum des Bildes – der Position des Roboters – werden Linien in alle Richtungen gezogen. Entlang dieser Linien werden die Farben der einzelnen Pixel betrachtet. Folgt eine bestimmte Anzahl grüner und weißer Pixel aufeinander, ist eine Transition gefunden, ihre Polarkoordinaten werden gespeichert und es wird in der nächsten Richtung weitergesucht. Zwischen den grünen und weißen Pixeln darf nur eine maximale Anzahl andersfarbiger Pixel liegen, die z. B. durch Rauschen oder Unschärfe entstanden sein können. Dieser Ablauf ist in Algorithmus 3.1 verdeutlicht.

Algorithmus 3.1 OmniScan($\mathcal{B}$)

Eingabe: farbklassifiziertes Kamerabild $\mathcal{B}$
Ausgabe: Liste von Transitionen in Polarkoordinaten $\mathcal{T}$
Ablauf:

```
T := empty
for φ := 0 to 359 do
    transitionFound := false
    counter_green := 0; counter_white := 0; counter_other := 0
    pixel := CenterOfImage(B)
    while pixel ∈ IsometricPart(B) and ¬transitionFound do
        if Color(pixel) = GREEN then counter_green++
        else if Color(pixel) = WHITE then counter_white++
            else counter_other++
        endif
        if counter_green ≥ MIN_green and counter_white ≥ MIN_white then
            transitionFound := true
            T := T ∪ (d(pixel), φ)
        endif
        if counter_other > MAX_other then
            counter_green := 0; counter_white := 0; counter_other := 0
        endif
        pixel := NextPixel(pixel, φ)
    endwhile
endfor
return T
```

Die Schrittweite, in der die Suchrichtung ϕ einen Kreis durchläuft, kann natürlich variabel gestaltet werden. Die Werte MIN_{green} und MIN_{white} geben die Mindestzahl an grünen und weißen Pixeln an, die aufeinander folgen müssen, um als Transition zu gelten. Werden mehr als MAX_{other} andersfarbige Pixel gefunden, werden alle Zähler auf 0 gesetzt und die Suche wird fortgesetzt. Die Funktion $NextPixel(pixel, \phi)$ wählt dabei den nächsten Bildpunkt aus, der auf einer radialen Linie in Richtung ϕ liegt. Die Bedingung der while-Schleife: $pixel \in IsometricPart(\mathcal{B})$ stellt sicher, dass nur der isometrische Bereich des Bildes durchsucht wird. Dort sind die Feldlinien von besonderem Interesse (siehe Abschnitt 2.2).

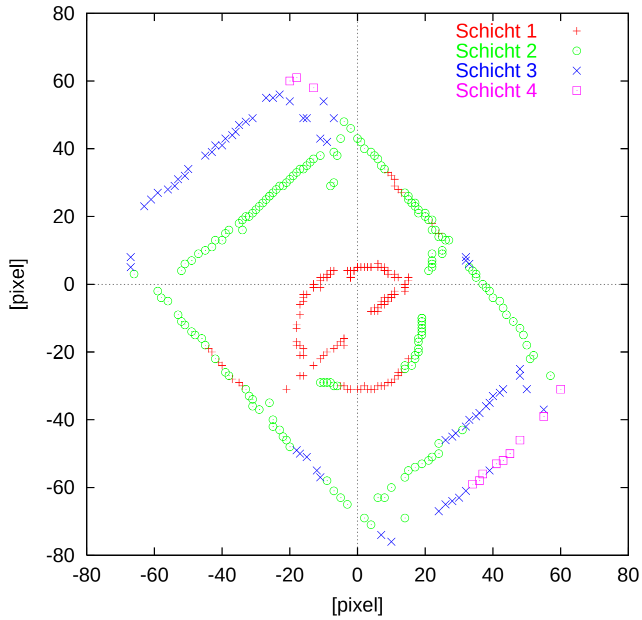
Um den Algorithmus zu beschleunigen, berechnen wir für jede Richtung ϕ im Voraus die Positionen der Pixel, die betrachtet werden müssen. Diese werden in einem Array gespeichert, das durch -1 terminiert wird. Die Funktion $NextPixel(pixel, \phi)$ muss damit nur auf den nächsten Array-Eintrag zugreifen und das Überprüfen der Abbruchbedingung $pixel \in IsometricPart(\mathcal{B})$ reduziert sich auf einen Vergleich mit -1.

Dadurch, dass die Suche in einer Richtung abgebrochen wird, wenn eine Transition gefunden ist, stellen die Punkte in $\mathcal{T}$ ein Abstandsprofil der Feldlinien dar, die am nächsten zum Roboter liegen. Leider werden dabei aber Linien, die weiter entfernt sind, nicht erfasst. Durch eine leichte Modifikation von Algorithmus 3.1 lässt sich das aber erreichen: Wird in Richtung ϕ eine Transition gefunden, wird diese wie zuvor in eine Liste $\mathcal{T}_1$ eingetragen. Danach wird die Suche allerdings entlang der gleichen Linie fortgesetzt. Die nächste Transition wird in eine Liste $\mathcal{T}_2$ eingetragen usw. Dadurch erhalten wir mehrere Abstandsprofile $\mathcal{T}_1, \dots, \mathcal{T}_n$, die wir die *Schichten* des OmniScans $\mathcal{T}$ nennen und die getrennt weiterverarbeitet werden können.
Ein Beispiel für einen solchen OmniScan ist in Abbildung 3.8 zu sehen. Auf das farb-klassifizierte Bild aus Abbildung 3.4 wurde der beschriebene Algorithmus angewendet. Dadurch erhält man die dargestellten Transitionen, die in vier Schichten angeordnet sind.

Extraktion von Linien: Der Algorithmus 3.2 auf Seite 37 kann dazu benutzt werden, aus einem OmniScan Liniensegmente zu bilden. Dieses Verfahren wird auch zur Extraktion von Linien aus Laserscans verwendet [Gutmann 2000]. Die Punkte werden gruppiert, um dann für jede Gruppe mittels der Funktion *Split* Liniensegmente zu extrahieren.
Die Gruppierung wird durch die Konstante *MAX-DISTANCE* gesteuert. Sie gibt an, wie weit zwei aufeinander folgende Punkte voneinander entfernt sein dürfen, um zur selben Gruppe zu gehören. Für unsere Anwendung liefert ein Wert von 10 Pixeln gute Ergebnisse. Die eigentliche Aufgabe der Linien-Extraktion wird durch die Funktion *Split* (Algorithmus 3.3, Seite 38) durchgeführt.

Die Funktion *Split* ist eine rekursive Funktion und folgt dem *Divide-and-Conquer*-Prinzip. Zunächst wird durch die Gruppe eine Ausgleichsgerade (*make-line*) gelegt, die die gesamte Punktmenge optimal annähert. *Optimal* heißt in diesem Fall, dass die Summe der Abstandsquadrate minimiert wird[2]. Enthält diese Linie genügend Punkte und ist die Standardabweichung σ klein genug, wird die Linie in die Lösungsmenge aufgenommen und zurückgesprungen (Basisfall der Rekursion). Ist jedoch die Standardabweichung zu groß, wird die Punktmenge an der Stelle i_{split} in zwei Untergruppen aufgeteilt (Divide) und die Funktion für beide Untergruppen rekursiv aufgerufen (Conquer). Die so erzeugten Teil-Lösungsmengen werden anschließend zur Gesamtlösung vereinigt (Merge). Um die Stelle i_{split} zu bestimmen, wird mittels der Funktion *distance-to-line* für jeden Punkt der Abstand zu

[2]Für Details zur Berechnung der Ausgleichsgeraden wird auf [Gutmann 2000] verwiesen.



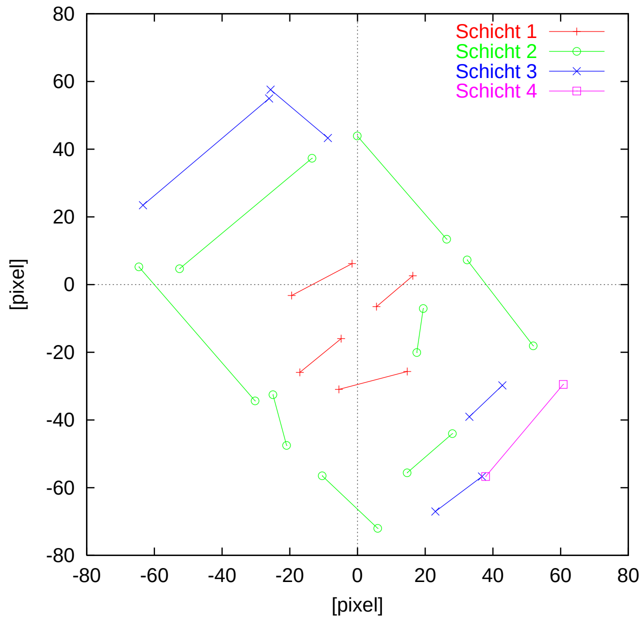


Abbildung 3.8: *Ein OmniScan mit vier Schichten (oben) und die Linien, die sich durch Linien-Extraktion daraus bilden lassen (unten).*

einer Geraden durch Start- und End-Punkt der Gruppe berechnet. Der Punkt, der den maximalen Abstand hat, definiert die Stelle, an der die Punktmenge aufgeteilt wird. Die mittlere Laufzeit dieses Verfahrens ist, wie bei den meisten Divide-and-Conquer-Verfahren $O(n \log n)$, wenn n die Anzahl an Punkten ist.

Algorithmus 3.2 Linien-Extraktion($\mathcal{T}$) [Gutmann 2000]

Eingabe: einzelne Schicht eines OmniScans $\mathcal{T}$
Ausgabe: Menge von Liniensegmenten $\mathcal{L}$
Ablauf:

$\mathcal{L} := empty$
$start := 0$
for $i := 1$ **to** $numpoints(\mathcal{T}) - 1$ **do**
  $p_1 :=$*n-th-point*$(\mathcal{T}, i-1)$
  $p_2 :=$*n-th-point*$(\mathcal{T}, i)$
  if $distance(p_1, p_2) >$*MAX-DISTANCE* **then**
    $\mathcal{L} := \mathcal{L} \cup Split(\mathcal{T}, start, i-1)$
    $start := i$
  endif
endfor
$\mathcal{L} := \mathcal{L} \cup Split(\mathcal{T}, start, numpoints(\mathcal{T}) - 1)$
return $\mathcal{L}$

Durch die Konstanten *MIN-POINTS-ON-LINE* und *MAX-SIGMA* werden die Anzahl und Qualität der erzeugten Linien beeinflusst. Sie müssen in Abhängigkeit vom Rauschen in den Datenpunkten gewählt werden. Sie sind bei uns auf 10 Punkte bzw. 30 Pixel festgelegt worden.

Abbildung 3.9 verdeutlicht, wie der Algorithmus einer gegebenen Punktmenge arbeitet. Das oberste Bild zeigt, wie durch die gesamte Punktmenge eine Ausgleichsgerade gelegt wird (durchgezogene Linie). Da für diese die Standardabweichung aber sehr groß ist, wird die Punktmenge an der mit einem Pfeil markierten Stelle aufgeteilt. Im mittleren Bild sind für beide Hälften die Ausgleichsgeraden eingezeichnet. Für die linke Seite stellt diese bereits eine Lösung dar. Die rechte Hälfte muss aber ein weiteres Mal an der markierten Stelle aufgeteilt werden, bevor die im unteren Bild dargestellte Linienmenge gefunden ist.

Wird der Algorithmus 3.2 zur Linien-Extraktion auf die einzelnen Schichten eines OmniScans angewendet, erhält man eine Menge von Linien, wie sie in Abbildung 3.8 auf der vorherigen Seite zu sehen ist.

Filterung der Daten: Bei der Linien-Extraktion werden auch aus Transitionen, die auf dem Mittelkreis liegen, Linien anstatt Kreissegmente extrahiert. Da außerdem die einzelnen Schichten des OmniScans unabhängig voneinander verarbeitet werden müssen, kann man verschiedene Liniensegmente erhalten, die eigentlich zur gleichen Feldlinie gehören (vgl. Abbildung 3.8). Auf Grund dieser Beobachtung verwenden wir noch einige Filter, die derartige Störungen beseitigen sollen.

Algorithmus 3.3 Split($\mathcal{T}, start, end$) [Gutmann 2000]

Eingabe: Gruppe von Punkten, die durch $\mathcal{T}$, $start$ und end festgelegt ist
Ausgabe: Menge von Liniensegmenten $\mathcal{L}$
Ablauf:

$\mathcal{L} := empty$
$line$:=*make-line*$(\mathcal{L}, start, end)$
if $numpoints(line)$ $\geq$*MIN-POINTS-ON-LINE* **then**
  if $\sigma(line)$ <*MAX-SIGMA* **then**
    $\mathcal{L} := \mathcal{L} \cup \{line\}$
  else
    p_{start} :=*n-th-point*$(\mathcal{T}, start)$
    p_{end} :=*n-th-point*$(\mathcal{T}, end)$
    $i_{split} := start$
    $d := 0$
    for $i := start + 1$ **to** $end - 1$ **do**
      p :=*n-th-point*$(\mathcal{T}, i)$
      if *distance-to-line*$(p, p_{start}, p_{end}) > d$ **then**
        $i_{split} := i$
        d :=*distance-to-line*(p, p_{start}, p_{end})
      endif
    endfor
    $\mathcal{L} := \mathcal{L} \cup Split(\mathcal{T}, start, i_{split})$
    $\mathcal{L} := \mathcal{L} \cup Split(\mathcal{T}, i_{split}, end)$
  endif
endif
return $\mathcal{L}$

- REDUKTIONSFILTER: Der Reduktionsfilter wird auf die Transitionsmenge angewendet, bevor die Linien-Extraktion ausgeführt wird. Er fasst Punktwolken zu einzelnen Punkten zusammen. Besonders in der Nähe des Bildmittelpunktes liegen die radialen Scanlinien und damit die gefundenen Transitionen sehr nahe beieinander. Solche Transitionen können durch einen Punkt ersetzt werden, um die Datenmenge zu reduzieren. Eine Beschreibung dieses Verfahrens für Laserscans findet sich in der Arbeit von *Gutmann* [Gutmann 2000].

- MEDIANFILTER: Da ein Kamerabild aus diskreten Pixeln besteht, ist die Genauigkeit der Transitionen durch die Auflösung des Bildes begrenzt und es entsteht ein Rauschen in den OmnisScan-Punkten. Durch den Medianfilter wird dieses Rauschen verringert und das Abstandsprofil des OmniScans geglättet. Dadurch kann man die Linienextraktion verbessern. Der Medianfil-

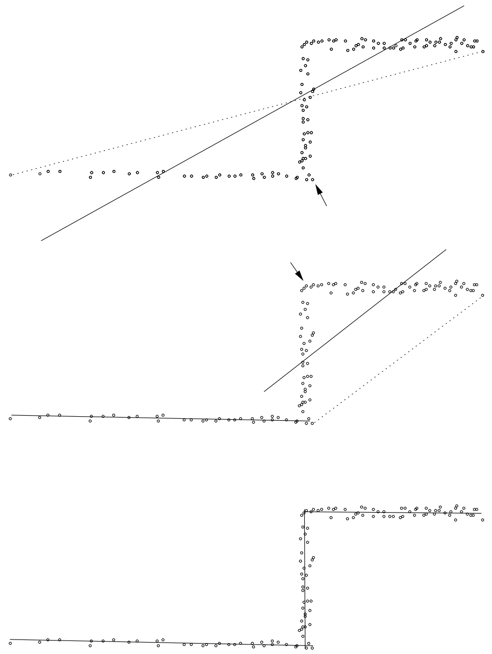


Abbildung 3.9: *Extraktion von Linien aus einer Menge von Punkten*

ter betrachtet zu jedem Punkt $P = (d, \varphi) \in \mathcal{T}$ eine bestimmte Anzahl linker und rechter Nachbarn und berechnet den Median ihrer Entfernungen zum Bildmittelpunkt. Dann wird die Distanz d des Punktes auf diesen Wert gesetzt. Das gleiche Verfahren wird auch zur Glättung von Laserscans benutzt [Weigel 1998].

- ORTHOGONALITÄTSFILTER: Die Feldlinien des Fußballfeldes stehen alle rechtwinklig zueinander. Eigentlich können die extrahierten Linien also nur in einer Hauptrichtung Ω oder orthogonal dazu in der Richtung $\Omega' = \Omega + 90°$ vorkommen. Weicht eine Linie stark von dieser Bedingung ab (wir sagen dann, sie ist „schief"), ist sie entweder durch Fehlmessungen entstanden oder sie ist Teil des Mittelkreises. In beiden Fällen ist es wünschenswert, die Linie zu entfernen. Genau dies tut der Orthogonalitätsfilter.
  Bevor schiefe Linien identifiziert werden können, muss natürlich die Hauptrichtung Ω bestimmt werden. Ein Kernpunkt ist dabei, längere Linien stärker zu gewichten, da sie in der Regel exakter sind, als kurze. In Algorithmus 3.4 auf Seite 42 wird ein Histogramm über die Winkel der Linien modulo 90° angelegt. Im Histogramm werden dabei nicht die Anzahl der Linien, sondern ihre Längen aufsummiert und zwar nicht nur an der Stelle des tatsächlichen Winkels, sondern auch in einer Umgebung, die durch TOL^{line}_{angle} festgelegt wird. Die Maximalstelle des Histogramms ist dann der gesuchte Winkel. In Abbildung 3.10 ist ein solches Histogramm zu sehen; die Maximalstelle ist durch eine gestrichelte Linie markiert. Jetzt kann man einfach alle Linien

entfernen, deren Abweichung von den Hauptrichtungen den Schwellenwert TOL_{angle}^{line} überschreitet. Bei uns wurde dieser Schwellenwert auf 10° festgelegt.
Gelegentlich kommt es vor, dass aus dem Mittelkreis Liniensegmente extrahiert werden, die parallel zu den Hauptachsen sind. Diese werden durch den Filter nicht entfernt. Die später vorgestellten Selbstlokalisierungsverfahren können aber mit solchen Linien umgehen.
Wird der Orthogonalitätsfilter auf die Linienmenge in Abbildung 3.11(a) angewendet, werden die durch gestrichelte Linien dargestellten Hauptachsen gefunden. Nach dem Entfernen der schiefen Linien bleiben die Linien in Abbildung 3.11(b) übrig.

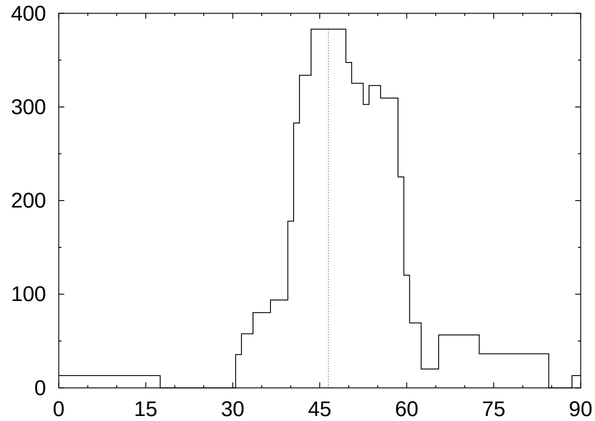


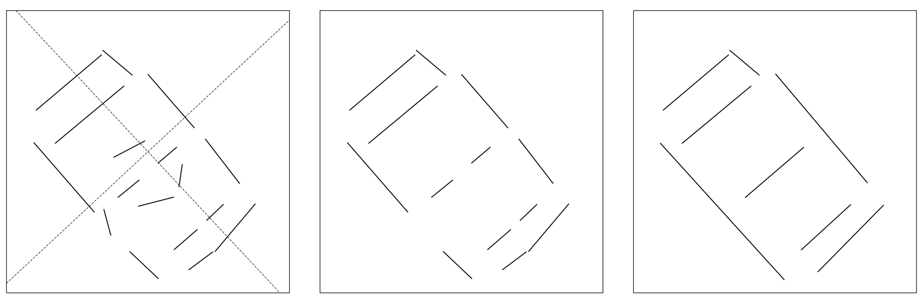
Abbildung 3.10: *Winkel-Histogramm zur Bestimmung der Hauptrichtung Ω einer Linienmenge*

- VERSCHMELZUNG: Liniensegmente, die auf einer Geraden liegen, stammen wahrscheinlich von ein und derselben Feldlinie und können zu einer Linie vereinigt werden. Der naive Ansatz, alle Linien paarweise zu betrachten und gegebenenfalls zu verschmelzen, benötigt quadratische Laufzeit in der Anzahl der Linien. Unsere Implementierung nutzt die Tatsache aus, dass nach der Anwendung des Orthogonalitätsfilters nur Linien in den Richtungen Ω und Ω' vorkommen (mit der erwähnten Toleranz). Anhand dieser Richtungen werden die Linien in zwei Mengen $\mathcal{L}_\Omega$ und $\mathcal{L}_{\Omega'}$ aufgeteilt. Die Linien jeder Menge werden nun mittels Algorithmus 3.5 auf Seite 42 verschmolzen. Dabei bezeichne $line_i$ die i-te Linie der Liste $\mathcal{L}$. Die Funktion $d(line_i, \Omega)$ berechnet die Distanz des Mittelpunktes der Linie $line_i$ von einer Geraden durch den Ursprung mit Richtung Ω, mit anderen Worten also den Abstand der Linie vom Roboter, wenn sie exakt die Ausrichtung Ω hätte. $SortLinesByDistance$ sortiert die Linien gemäß diesem Wert.
Der Algorithmus kann als Sweep-Line-Verfahren aufgefasst werden. Die

Sweep-Line hat dabei die Richtung Ω und wandert durch die Mittelpunkte der Linien. Alle Linien, die in einem Streifen der Breite $TOL_{distance}^{line}$ liegen, werden durch die Funktion *make-line* zu einer Linie verschmolzen. Dabei wird – analog zur Funktion *make-line* aus Algorithmus 3.3 – eine Ausgleichsgerade durch die Start- und Endpunkte der Linien $line_{start}$ bis $line_{end}$ gelegt.
Die Laufzeit des Verfahrens wird durch den Sortieraufwand dominiert; der eigentliche Verschmelzungsprozess erfolgt in linearer Zeit. Es wird also bei n Linien im Mittel $O(n \log n)$ viel Zeit benötigt.
Mit einem Wert von 3 Pixeln für $TOL_{distance}^{line}$ liefert das Verfahren gute Ergebnisse. In Abbildung 3.11(c) wurden die Linien aus Abbildung 3.11(b) auf die beschriebene Weise verschmolzen.

- Es sind noch viele andere Möglichkeiten der Linienfilterung denkbar. So kann man beispielsweise Linien, die eine bestimmte Mindestlänge unterschreiten (LÄNGENFILTER) oder einen maximalen Abstand vom Roboter überschreiten (DISTANZFILTER), entfernen.

Abschließend werden die Feldlinien, die immer noch in Bildkoordinaten angegeben sind, in relative Spielfeldkoordinaten umgerechnet. Dies geschieht durch Division aller Koordinaten durch die Proportionalitätskonstante k der Omnicam (siehe Abschnitt 2.3.1).



(a) (b) (c)

Abbildung 3.11: *Filterung der Linien:*
(a) *Ungefilterte Linienmenge (die gestrichelten Linien sind die Hauptachsen)*
(b) *Dieselbe Linienmenge nach Anwendung des Orthogonalitätsfilters und*
(c) *nach Verschmelzung von Segmenten, die auf einer Geraden liegen.*

Algorithmus 3.4 Hauptrichtung($\mathcal{L}$)

Eingabe: Menge von Liniensegmenten $\mathcal{L}$
Ausgabe: Winkel Ω, zu dem die meisten Linien parallel oder senkrecht sind
Ablauf:
  init $histogram[90]$
  for each line $l \in \mathcal{L}$ **do**
    for $\alpha := angle(l) - TOL^{line}_{angle}$ **to** $angle(l) + TOL^{line}_{angle}$ **do**
      $histogram[\alpha\ modulo\ 90°] \mathrel{+}= length(l)$
    endfor
  endfor
  $\Omega := argmax(histogram)$
  return Ω

Algorithmus 3.5 Linien-Verschmelzung($\mathcal{L}, \Omega$)

Eingabe: Liste von „quasi"-parallelen Liniensegmenten $\mathcal{L}$
Ausrichtung Ω der Linien
Ausgabe: Menge der verschmolzenen Liniensegmente $\mathcal{L}'$
Ablauf:
  $\mathcal{L}' := empty$
  $\mathcal{L} := SortLinesByDistance(\mathcal{L}, \Omega)$
  $start := 0$
  while $start < numLines(\mathcal{L})$ **do**
    $end := start$
    while $|d(line_{start}, \Omega) - d(line_{end}, \Omega)| < TOL^{line}_{distance}$ **do**
      end++
    endwhile
    $\mathcal{L}' := \mathcal{L}' \cup$ *make-line*$(\mathcal{L}, start, end - 1)$
    $start := end$
  endwhile
  return $\mathcal{L}'$

3.2.2 Eckpfosten

Nach den Regeln des RoboCup befindet sich in jeder Ecke des Spielfeldes ein senkrechter Pfosten mit 20cm Durchmesser und einer Höhe von 90cm. Wir nennen diese Eckpfosten auch Poles und unterscheiden im Folgenden drei Arten:

- GELBE POLES befinden sich an den Ecken rechts und links vom gelben Tor. Sie bestehen aus drei Teilen, die jeweils 1/3 der Höhe ausmachen und verschiedene Farben haben. Das obere Drittel ist gelb, das mittlere blau und das untere wieder gelb.

- BLAUE POLES befinden sich auf der anderen Spielfeldseite, rechts und links vom blauen Tor. Bei ihnen ist die Farbmarkierung allerdings andersherum, also blau-gelb-blau.

- UNBEKANNTE POLES nennen wir Eckpfosten, die wir zwar als solche erkennen, aber nicht entscheiden können, ob sie gelb oder blau sind. Das kann beispielsweise passieren, wenn wir die Farbfolge gelb-blau erkennen, aber nicht wissen, ob das untere Drittel eines gelben Poles verdeckt ist oder das obere Drittel eines blauen Poles.

Zum Detektieren der Poles im Kamerabild benutzen wir die Farbklekse (Blobs), die wir in Abschnitt 3.1.3 beschrieben haben. Insbesondere benötigen wir zu jedem gelben und blauen Blob die Anzahl an Pixeln und die relative Position des Flächen-Schwerpunktes. Blobs, deren Pixelzahl nicht innerhalb eines bestimmten Bereichs liegt (bei uns etwa 4 bis 600 Pixel), werden dabei nicht weiter betrachtet, ebenso wie Blobs, die außerhalb des isometrischen und konstant gekrümmten Spiegelbereichs liegen.
Weiterhin nutzen wir eine wichtige Eigenschaft der omnidirektionalen Kamera aus: senkrechte Linien der Umwelt werden auf radiale Linien in der Bildebene abgebildet. Wir suchen also im farb-segmentierten Kamerabild nach Blobs, deren Schwerpunkte auf der gleichen radialen Linie liegen und die Farbkombination gelb-blau-gelb oder blau-gelb-blau bilden.

Der Algorithmus geht nun wie folgt vor: Zuerst werden alle Paare von gelben und blauen Blobs betrachtet. Wird ein Paar gefunden, bei dem die Mittelpunkte ungefähr in der selben Richtung $\varphi \pm TOL_{angle}^{pole}$ liegen, ist ein unbekannter Pole gefunden. In diesem Fall wird versucht, den Typ zu bestimmen. Dazu werden alle übrigen Blobs geprüft, ob sie ebenfalls innerhalb der Toleranz um φ liegen und ob sich mit ihnen die Farbkombination gelb-blau-gelb oder blau-gelb-blau ergibt.
Die Kameratransformation lässt die Winkel von Objekten unverändert. Daher kann die Richtung des Poles geschätzt werden, indem das gewichtete Mittel der Winkel aller beteiligten Blobs gebildet wird. Als Gewicht wird dabei die Größe eines Blobs, also die Anzahl seiner Pixel, benutzt:

$$\varphi_{pole} = \frac{1}{S} \sum_{i=0}^{n-1} \left(size\left(blob_i\right) \varphi\left(blob_i\right) \right), \tag{3.4}$$

mit $S = \sum_{i=0}^{n-1} size\left(blob_i\right)$. Zu beachten ist eine geeignete Normierung der Winkel, um Probleme am Sprung zwischen 360° und 0° zu vermeiden.
Eine Entfernungsschätzung zu den Poles ist schwierig und würde nur ungenaue Ergebnisse liefern. Deshalb wurde in dieser Arbeit darauf verzichtet.

Aus den gelben und blauen Blobs in Abbildung 3.12 werden auf diese Weise vier Eckpfosten erkannt: Zwei gelbe Poles (unten rechts im Bild), ein blauer (oben) und ein unbekannter (links).

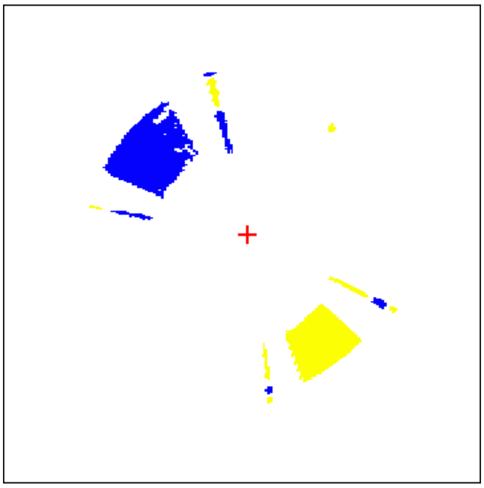


Abbildung 3.12: *Die blauen und gelben Blobs einer Kameraaufnahme. Sie werden zur Detektion der Eckpfosten und Tore benutzt.*

3.2.3 Tore

Beim Erkennen der Tore werden die beiden Werte *Richtung* und *Entfernung* unterschiedlich behandelt.

Die Richtungsschätzung geschieht – ähnlich, wie bei den Eckpfosten – basierend auf den Farbkleksen. Als erste Hypothese wird der größte Blob der entsprechenden Farbe ausgewählt, dessen Pixelzahl in einem bestimmten Bereich liegt (bei uns 300 bis 5000 Pixel). Diese Hypothese kann aber sehr schlecht sein. Man stelle sich zum Beispiel vor, im Tor stehe ein Torwart. Dieser trennt das Farbige Tor in zwei Teile, so dass zwei getrennte Blobs entstehen. Diese liegen aber relativ nahe beieinander und man kann versuchen, sie zu vereinigen. Dies geschieht wie folgt: ein Blob wird zur Hypothese hinzugefügt, wenn er groß genug ist und einen bestimmten Maximalabstand nicht überschreitet. Die neue Hypothese ist der Mittelwert der beiden Flächen-Schwerpunkte. Dadurch erhält man eine bessere Richtungsschätzung. Probleme können dabei die Blobs der Eckpfosten bereiten. Sie haben die gleiche Farbe, wie die Tore, neben denen sie stehen und können unter Umständen mit diesen vereinigt werden. Da wir aber Blobs, die bereits einem Pole zugeordnet wurden, zuvor entfernen, kommt dies nur selten vor.

Die Exaktheit der Schätzung leidet darunter, dass auch die Seitenwände des Torkastens farbig sind. Besonders bei einem spitzen Winkel zum Tor verschiebt sich dadurch der Flächen-Schwerpunkt. Die Methode ist also noch verbesserungswürdig.

Bei der Entfernungsschätzung benutzen wird das gleiche Verfahren wie zur Detektion der Feldlinien. Diesmal wandeln wir aber den Algorithmus 3.1 so ab, dass er Transitionen zwischen grünen und gelben bzw. grünen und blauen Pixeln findet. Aus diesen Punkten kann eine Linie extrahiert werden, die die untere Kante des Tores bildet. Wir nennen diese Unterkante den „Fuß" des Tores.
Schneiden wir die Linie mit der radialen Linie der Richtungsschätzung, erhalten wir einen Punkt, der eine Schätzung für die Mitte des Torfußes darstellt. Da sich dieser Punkt in der Spielfeldebene befindet, kann er wie die Feldlinien durch die inverse Kameratransformation in Weltkoordinaten umgerechnet werden und liefert eine Entfernungsschätzung zum Tor.
Leider gelingt das Extrahieren einer Linie aus den Transitionen nicht immer. Befindet sich ein Torwart im Tor, ist der sichtbare Bereich des Fußes zu klein. In diesem Fall muss die Richtung ausreichen. Zudem ist das Ergebnis oft ungenau, wenn der Torkasten schlecht ausgeleuchtet ist. Die vielen Schatten verhindern eine klare Abzeichnung des Farbübergangs zwischen Feld und Tor.

3.3 Zusammenfassung

In diesem Kapitel haben wir uns mit der Merkmalsextraktion aus einem Kamerabild beschäftigt. Wir haben zunächst die interne Darstellung einer solchen Aufnahme, das YUV-Format, erläutert. Dann wurde beschrieben, wie die für diese Arbeit relevanten Objekte detektiert werden.
Zuerst werden die Bildpunkte verschiedenen Farbklassen zugeordnet und zu Farbkleksen, den Blobs, zusammengefasst. Außerdem werden bestimmte Farbübergänge ermittelt, die Transitionen. Diese Daten besitzen bereits einen deutlich höheren Informationsgehalt als das ursprüngliche Bild, werden aber zur Objekterkennung noch weiter verarbeitet.
Die Blobs werden dazu benutzt, die Richtungen zu den Eckpfosten und Toren zu erkennen. Darüber hinaus können sie auch zur Ballerkennung oder zur Identifizierung von Mit- und Gegenspielern nützlich sein.
Aus den Transitionen werden Linien extrahiert, um die Spielfeldlinien zu bestimmen. Dabei wurde viel Wert auf eine robuste und effiziente Erkennung gelegt. Auch werden die Transitionen benutzt, um den Fuß der Tore zu detektieren und damit die Torentfernung zu schätzen. Dieses Vorgehen ist aber nicht sehr exakt.
Für einen Überblick über den Ablauf der Merkmalsextraktion sei noch einmal auf Abbildung 3.1 auf Seite 24 verwiesen. Der gesamte Vorgang wurde auf einem Vaio-Notebook unserer Roboter getestet (siehe Abschnitt 1.4.2). Es wurde dabei eine Zeit von ca. 31 Millisekunden (ms) pro Kamerabild benötigt. Damit ist also die Bildverarbeitung in Echtzeit bei einer Framerate von 30 Bildern pro Sekunde möglich.

Kapitel 4

Lokalisierung

Mit *Selbstlokalisierung* oder kurz *Lokalisierung* wird der Vorgang bezeichnet, mit dem ein mobiler Roboter seine Position in der Umwelt schätzt. Dies wird im Allgemeinen als eines der grundlegenden Probleme der Robotik angesehen. Mit der Position eines Roboters ist stets ein Tripel $(x, y, \theta)^T$ gemeint, wobei (x, y) die Koordinaten sind, an denen er sich befindet und θ seine Ausrichtung. Da die Positionsbestimmung immer mit Sensoren geschieht, deren Messungen Fehlern unterworfen sind, kann man nie eine absolute Aussage über die Roboterposition treffen. Statt dessen wird die wahrscheinlichste Position geschätzt.
Ein typisches Schema für den Lokalisierungsvorgang, das auch in dieser Arbeit zur Anwendung kommt, ist in Abbildung 4.1 dargestellt. Die Prädiktion liefert eine Vorhersage der Roboterposition auf Basis der letzten Positionsschätzung und der Odometrie-Messungen des Roboters. Dieser Vorgang ist Ungenauigkeiten unterworfen, weshalb sich die Unsicherheit der Positionsschätzung in der Regel erhöht. Im Korrekturschritt wird eine neue Schätzung ermittelt auf Basis der Vorhersage und einer Sensormessung.

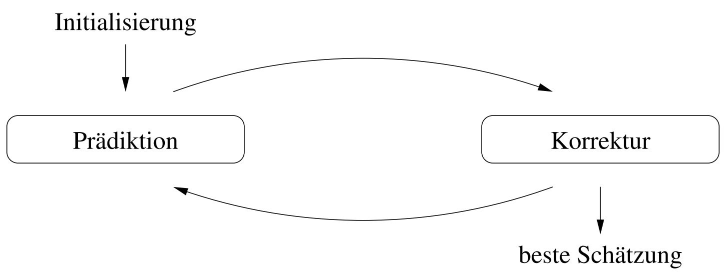


Abbildung 4.1: *Grundprinzip der Selbstlokalisierung*

Üblicherweise unterteilt man die Lokalisierungsverfahren in *lokale* und *globale* Verfahren:

- Bei der LOKALEN POSITIONSSCHÄTZUNG, auch RELATIVE LOKALISIE-

RUNG oder TRACKING genannt, wird, ausgehend von einer bekannten Position, die Bahn des Roboters verfolgt. Meist geschieht das, indem man versucht, Fehler in der Odometrie durch andere Sensormessungen zu korrigieren. Ist die Position allerdings zu Beginn unbekannt oder geht sie während der Fahrt verloren, ist es dem Roboter damit im Allgemeinen nicht möglich, sich zu lokalisieren.

- GLOBALE oder ABSOLUTE VERFAHREN ermöglichen die Lokalisierung ohne Vorwissen über die Position. Dadurch sind sie auch in der Lage, das sogenannte *„Kidnapped-Robot-Problem"* zu lösen. Bei diesem Problem werden dem Roboter sozusagen die Augen verbunden, er wird ohne Sensorinformationen an eine beliebige Position der Umgebung getragen und muss sich erneut lokalisieren. Dieser Test verdeutlicht die Fähigkeit eines Lokalisierungsverfahrens, mit extremen Positionierungsfehlern umgehen zu können. Im Allgemeinen sind globale Lokalisierungsverfahren mächtiger als lokale Verfahren, erkaufen sich dies jedoch durch höheren Rechenaufwand und eine geringere Genauigkeit.

Je nach Einsatz und Aufgabe eines Roboters können verschiedene Anforderungen an die Genauigkeit der Positionsschätzung gestellt werden. Beim CS Freiburg gehen wir davon aus, dass eine exakte Lokalisierung der Roboter beim Fußballspiel Vorteile bei Taktik, Kooperation und globaler Sensorintegration bietet [Gutmann 2000, Fischer 2002]. Bisher bestimmen unsere Fußball-Roboter ihre Positionen mit Hilfe ihres Laser-Scanners. Dieser liefert ein Abstandsprofil zu den Banden, die das Spielfeld umgeben. Aus diesem Abstandsprofil werden Linien gebildet, die dann mit einem entsprechenden Modell des Feldes in Deckung gebracht werden, um die Position zu bestimmen. Mit diesem Algorithmus, *LineMatch* genannt [Weigel 1998], lässt sich eine schnelle und sehr genaue Positionsschätzung mit einem mittleren Fehler von etwa 2 cm bzw. 1° erreichen. In Folge einer Regeländerung wurden die Spielfeldbanden entfernt und durch eine Reihe von Pfosten ersetzt [RoboCup 2002]. Die LineMatch-Lokalisierung mittels Laser-Scanner ist derzeit immer noch funktionstüchtig, indem aus den Pfostenreihen Linien extrahiert werden [Weigel et al. 2002]. Da allerdings der Abstand der Pfosten in Zukunft nach und nach vergrößert werden soll, bis sie ganz verschwunden sind, müssen andere Sensoren und Lokalisierungsverfahren eingesetzt werden.
In diesem Kapitel werden zwei Methoden der Selbstlokalisierung mit einer omnidirektionalen Kamera vorgestellt. Das erste Verfahren ist eine Adaption des LineMatch-Algorithmus: die erkannten Spielfeldlinien werden auf ein Modell des Feldes abgebildet. Das zweite Verfahren ist eine Variante der *Markov-Lokalisierung*. Diese ist eine in vielen Szenarien eingesetzte Methode zur globalen Lokalisierung [Fox 1998].

4.1 LineMatch

Auf Grund der guten Erfahrungen, die mit der *LineMatch-Lokalisierung* gemacht worden sind, stellt sich die Frage, ob sich das Verfahren auch zur Lokalisierung mit einer omnidirektionalen Kamera eignet. Da die Kameradaten fehleranfälliger als die Daten eines Laser-Scanners sind, muss der Algorithmus leicht überarbeitet und erweitert werden, indem sogenannte Ausreißerlinien eingeführt werden. Das sind Linien, die keiner Modelllinie zugeordnet werden, da sie stark verrauscht oder durch Falschmessung entstanden sind. Das Verfahren bestimmt eine Menge von globalen Positionshypothesen. Anschließend wird die plausibelste Hypothese unter Verwendung eines Kalman-Filters mit den Informationen der Odometrie fusioniert. Auf diese Weise ist eine robuste Selbstlokalisierung in der RoboCup-Umgebung möglich.

4.1.1 Kalman-Filter

Eine Beobachtung wird oft von verschiedenen Sensoren erfasst. Diese liefern eine Messgröße mit unterschiedlicher Zuverlässigkeit. Mit einem *Kalman-Filter* lassen sich diese Messungen in optimaler Weise kombinieren. Voraussetzung für die Anwendbarkeit eines Kalman-Filters ist, dass zwei Messwerte z_1 und z_2 mit einem normalverteilten Fehler vorliegen, d. h. sie lassen sich in der Form $z_i \sim N(\mu_i, \Sigma_i)$ angeben, wobei μ_i der Erwartungswert und Σ_i die Kovarianzmatrix der Normalverteilung ist. Letztere beschreibt die Zuverlässigkeit der Messung.
Der Kalman-Filter berechnet daraus eine neue Normalverteilung $N(\mu, \Sigma)$, deren Unsicherheit geringer ist als die Unsicherheiten der beiden Messungen. Der neue Erwartungswert μ ergibt sich als gewichtetes arithmetisches Mittel der ursprünglichen Erwartungswerte. Die Gewichte sind dabei umgekehrt proportional zu den Varianzen, wodurch zuverlässigere Messungen stärker gewichtet werden:

$$\mu := (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \cdot (\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) \tag{4.1}$$

$$\Sigma := (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \tag{4.2}$$

Ist die Annahme eines normalverteilten Fehlers gerechtfertigt und ist das Rauschen in den Messdaten „weiß"[1], so ist dies eine optimale Schätzung für den tatsächlichen Zustand. Eine ausführliche Beschreibung befindet sich auf der Internet-Seite von *Greg Welch* [Welch und Bishop 2002] und ein anschauliches Beispiel in der Arbeit von *Maybeck* [Maybeck 1990].

Kalman-Filterung wird auch zur lokalen Selbstlokalisierung von Robotern eingesetzt. Dazu müssen die Positionsschätzungen in Form von Normalverteilungen gegeben sein, d. h. durch einen Erwartungswert μ und eine Kovarianzmatrix Σ, die

[1] Weißes Rauschen ist nicht über die Zeit korreliert, d. h. das Rauschen ist zu jedem Zeitpunkt unabhängig vom Rauschen zu anderen Zeitpunkten.

die Genauigkeit der Schätzung angibt. Das gilt insbesondere für die Vorhersagen des Prädiktionsschrittes und die Sensormessungen, denn diese werden im Korrekturschritt durch den Kalman-Filter zu einer optimalen Positionsschätzung fusioniert.

4.1.2 Koppelnavigation

Im Prädiktionsschritt wird durch *Koppelnavigation* eine Vorhersage über die aktuelle Roboterposition getroffen. Das Prinzip ist dabei das Folgende: Die Odometrie liefert Daten über die Drehbewegungen der Antriebsräder. Da die Größe und der Abstand der Räder bekannt ist, sind dadurch Rückschlüsse über die zurückgelegte Strecke und die Richtungsänderung des Roboters möglich. Diese Positionsänderung wird zur letzten Positionsschätzung hinzuaddiert. Auf kurzen Strecken liefert die Koppelnavigation gute Ergebnisse. Auf Dauer machen sich jedoch Fehler bemerkbar, die durch Ungenauigkeiten in den Odometriemessungen verursacht werden. Deshalb wird bei der Positionsvorhersage die Unsicherheit in Form der Varianzen erhöht. Gründe für Ungenauigkeiten in den Odometrie-Daten sind im Wesentlichen das Rutschen der Räder (Schlupf), ungenaue Geradeausfahrt (Drift) und Fehler bei den Drehbewegungen (Rotationsfehler), die zum Beispiel durch Unebenheiten im Boden verursacht werden. Details zu diesem Thema und zur Abschätzung des entstehenden Fehlers sind in der Doktorarbeit von *Steffen Gutmann* zu finden [Gutmann 2000].

4.1.3 Linienüberdeckung

In diesem Abschnitt wird beschrieben, wie wir aus den Kameradaten globale Positionshypothesen erzeugen. Dazu wird ein Modell des Feldes mit den detektierten Spielfeldlinien überdeckt. Voraussetzung für diese Linienüberdeckungen ist eine Menge $\mathcal{L}$ von Feldlinien, die auf die in Abschnitt 3.2.1 beschriebene Weise gefiltert wurden. Zur Erinnerung: „schiefe" Linien wurden entfernt, so dass alle Linien paarweise betrachtet – innerhalb einer Toleranz – parallel oder orthogonal zueinander stehen.
Weiter wird ein Linienmodell des Spielfeldes benötigt, wie es in Abbildung 4.2 zu sehen ist. Dabei ist der Mittelkreis, der einen Durchmesser von 2 m hat, durch ein Quadrat mit Seitenlänge 1,6 m ersetzt. Dies soll eine Zuordnung von Linien, die aus dem Mittelkreis extrahiert und durch den Orthogonalitätsfilter nicht entfernt wurden, zu Modell-Linien ermöglichen. Zusätzlich definieren wir noch eine Modelllinie $m_{outlier}$, die keiner physikalischen Linie entspricht. Ihr werden Linien aus $\mathcal{L}$ zugeordnet, die durch Fehlmessung entstanden sind und sich daher keiner anderen Modelllinie zuordnen lassen. Solche Ausreißerlinien können zum Beispiel durch Gegenstände außerhalb des Feldes verursacht werden. Die Menge der Linien des Linienmodells zusammen mit der Linie $m_{outlier}$ bezeichnen wir mit $\mathcal{M}$.

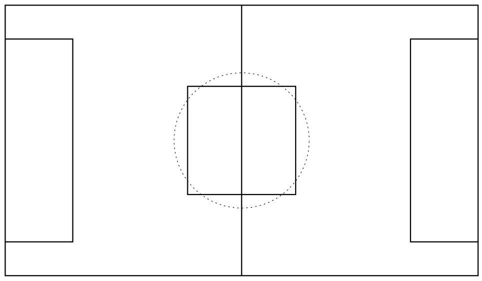


Abbildung 4.2: *Linienmodell des RoboCup-Spielfeldes in der KI-Abteilung der Universität Freiburg. Das Feld ist mit 7×4 m^2 etwas kleiner als ein reguläres Spielfeld.*

Analog zur ursprünglichen Arbeit über den LineMatch-Algorithmus [Weigel 1998] betrachten wir nun Abbildungen h, die jeder Linie $l \in \mathcal{L}$ eine Modelllinie $m \in \mathcal{M}$ zuordnen:

$$h : \mathcal{L} \to \mathcal{M} \tag{4.3}$$

Zu jeder solchen Zuordnung definieren wir die beiden Funktionen $outliers$ und $error$ wie folgt:

$$outliers(h) := |\{l \in \mathcal{L} \mid h(l) = m_{outlier}\}| \tag{4.4}$$

$$error(h) := \sum_{l \in \mathcal{L}} d(l, h(l))^2 \tag{4.5}$$

Die Funktion $outliers$ gibt dabei die Anzahl der Linien an, die als Ausreißer markiert wurden und $error$ definiert den Fehler in der Überdeckung der Modell-Linien durch die Zuordnung. Es sei $d(l, m)$ für $m \neq m_{outlier}$ der Abstand zwischen der Linie und der Modelllinie – etwa als die Summe der Abstände zwischen den Endpunkten von l und der Geraden, auf der m liegt – und $d(l, m_{outlier}) := 0$. Dabei sei die Linienmenge so gedreht und verschoben, dass $error$ minimal ist.

Die Menge der gültigen Zuordnungen, die Hypothesenmenge $\mathcal{H}$, wird nun definiert als:

$$\mathcal{H} := \{h \mid error(h) < MAX_{error} \wedge outliers(h) < MAX_{outliers}\} \tag{4.6}$$

Jede solche Hypothese impliziert eine Verdrehung und Verschiebung der Linienmenge und damit eine mögliche Aufnahmeposition.

Abbildung 4.3 zeigt, wie eine solche Hypothese aussehen kann. Die detektierten Linien sind den Modelllinien zugeordnet und dem entsprechend gedreht und verschoben worden. Damit ergibt sich die eingezeichnete Aufnahmeposition. Die Linie ganz rechts wurde vom Algorithmus als Ausreißerlinie markiert.

Die Suche nach den gültigen Hypothesen kann man sich als Suche in einem Baum vorstellen, bei dem in jedem Knoten die Zuweisung einer Linie zu einer Modelllinie geschieht. Durch den Pfad von der Wurzel zu einem Knoten ist dann eine

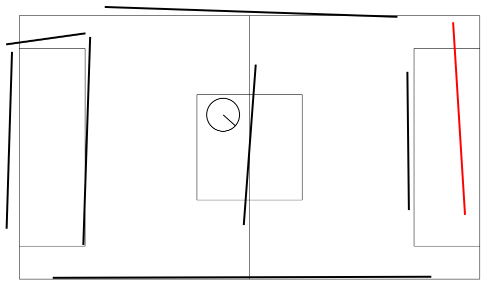


Abbildung 4.3: *Ergebnis des LineMatch-Algorithmus: Die Linie ganz rechts ist eine Ausreißerlinie.*

Teilzuordnung h' über dem Definitionsbereich $\mathcal{L}' \subseteq \mathcal{L}$ definiert. An den Blättern des Baumes befinden sich die vollständigen Zuordnungen h. Da die Funktionen $error$ und $outliers$ entlang eines Pfades monoton steigen, braucht ein Teilbaum nicht weiter durchsucht zu werden, wenn an seiner Wurzel eine der Bedingungen aus Formel 4.6 verletzt ist. Der Suchvorgang lässt sich rekursiv durch Algorithmus 4.1 beschreiben.

Algorithmus 4.1 LineMatch($\mathcal{L}, \mathcal{M}, \mathcal{P}$) [Gutmann 2000]

Eingabe: Linienmenge $\mathcal{L}$, Modell-Linienmenge $\mathcal{M}$, Paarungen $\mathcal{P}$
Ausgabe: Satz von Positionshypothesen $\mathcal{H}$
Ablauf:
  if $|\mathcal{P}| = |\mathcal{L}|$ **then**
    $\mathcal{H} := \{\mathcal{P}\}$
  else
    $\mathcal{H} := \emptyset$
    $l := SelectLine(\mathcal{L}, \mathcal{P})$
    for all $m \in \mathcal{M}$ **do**
      if $VerifyMatch(\mathcal{L}, \mathcal{M}, \mathcal{P} \cup \{(l, m)\})$ **then**
        $\mathcal{H} := \mathcal{H} \cup LineMatch(\mathcal{L}, \mathcal{M}, \mathcal{P} \cup \{(l, m)\})$
      endif
    endfor
  endif
  return $\mathcal{H}$

SelectLine wählt die nächste zu betrachtende Linie aus und *VerifyMatch* überprüft, ob durch Hinzunahme der neuen Zuordnung die Bedingungen aus Formel 4.6 erfüllt bleiben. Um den $error$-Wert zu berechnen, muss dabei eine Verdrehung und Verschiebung der Linienmenge berechnet werden. Im Folgenden stellen wir kurz dar, wie dies bei uns geschieht.

Verdrehung: Um die Verdrehung der Linienmenge zu bestimmen, wird bei uns das gewichtete Mittel der Linienwinkel berechnet:

$$\alpha = \frac{1}{Length(\mathcal{L})} \sum_{l \in \mathcal{L}} \Big[length(l) \big(\alpha(l) \ modulo \ 90^\circ \big) \Big] \tag{4.7}$$

Dabei seien $length(l)$ und $\alpha(l)$ die Länge bzw. Richtung der Linie l und $Length(\mathcal{L})$ die Gesamtlänge aller Linien in $\mathcal{L}$. Die Linien werden mit ihrer Länge gewichtet, da kurze Linien störanfälliger sind. Dieser Wert ist der Erwartungswert der Linienrichtungen modulo 90°. Bei der Berechnung ist Vorsicht geboten wegen der Diskontinuität bei 90°. Da die Winkel durch den Orthogonalitätsfilter nur geringfügig voneinander abweichen (maximal $2 \cdot TOL_{angle}^{line}$, siehe Abschnitt 3.2.1) ist die Berechnung eindeutig. Gleichzeitig bestimmen wir auch noch die Varianz der Winkel:

$$\sigma_\alpha^2 = \frac{1}{Length(\mathcal{L})} \sum_{l \in \mathcal{L}} \Big[length(l) \big((\alpha(l) \ modulo \ 90^\circ) - \alpha \big)^2 \Big] \tag{4.8}$$

Dieser Wert dient später als Varianz der geschätzten Roboterausrichtung.
Der Wert α wird bei uns vor dem eigentlichen LineMatch bestimmt. Die Berechnung muss daher nur einmal ausgeführt werden. Danach gibt es vier mögliche Verdrehungen der Linienmenge, die sich um jeweils 90° unterscheiden. Sind keine anderen Informationen vorhanden, wird nacheinander mit allen vier Ausrichtungen der LineMatch-Algorithmus gestartet, wobei dann nur noch Verschiebungen der Linienmenge betrachtet werden. Ist allerdings zusätzlich zu den Linien die Richtung zu einem der Tore bekannt, lässt sich leicht eine eindeutige Verdrehung identifizieren und nur diese muss weiter betrachtet werden.

Verschiebung: Wir benutzen bei der Bestimmung der Verschiebung eine leichte Variation der $error$-Funktion 4.5, um wiederum längeren Linien mehr Einfluss zu geben:

$$error^* := \frac{1}{Length(\mathcal{L})} \sum_{l \in \mathcal{L}} \Big[length(l) \ d(l, h(l))^2 \Big] \tag{4.9}$$

Die Verschiebung der Linien wird so gewählt, dass $error^*$ minimiert wird. Dadurch ergeben sich – ähnlich wie bei der Bestimmung der Verdrehung – die Verschiebungen dx und dy mit den Varianzen σ_x^2 und σ_y^2.

Aufnahmeposition: Jede gültige Zuordnungshypothese $h \in \mathcal{H}$ definiert eine mögliche Aufnahmeposition $pos_h = (\mu_h, \Sigma_h)$ für die Linienmenge. Da alle Linien relativ zur Kameraposition angegeben sind, ergibt sich diese direkt aus der beschriebenen Verdrehung und Verschiebung zu:

$$\mu_h = (-dx, -dy, -\alpha)^T \quad (4.10)$$

$$\Sigma_h = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\alpha^2 \end{pmatrix} \quad (4.11)$$

4.1.4 Selbstlokalisierung

Das beschriebene Verfahren zur Linienüberdeckung liefert im Idealfall nur eine einzige Hypothese h. Sind die Linien aber verrauscht, Teile des Feldes verdeckt oder die Tore nicht sichtbar, so treten Mehrdeutigkeiten auf und wir erhalten mehrere mögliche Aufnahmepositionen. Diese ergeben sich aus der Symmetrie des Feldes, aber auch aus anderen Ähnlichkeiten, zum Beispiel zwischen der Tiefe des Straframes und dem „Mittelkreis". Um eine einzige globale Position zu bestimmen, haben wir versucht, den einzelnen Hypothesen eine Qualität zuzuordnen. Als Qualitätsmaß wurden verschiedene Werte und deren Kombination in Betracht gezogen. Dazu gehören die Varianzen, der Fehlerwert und der zu erwartende Abstand bzw. Winkel zu den Toren. Dieses Vorgehen hat allerdings keine zuverlässige Identifizierung der besten Position ermöglicht. Deshalb wurde ein anderer Weg gewählt.

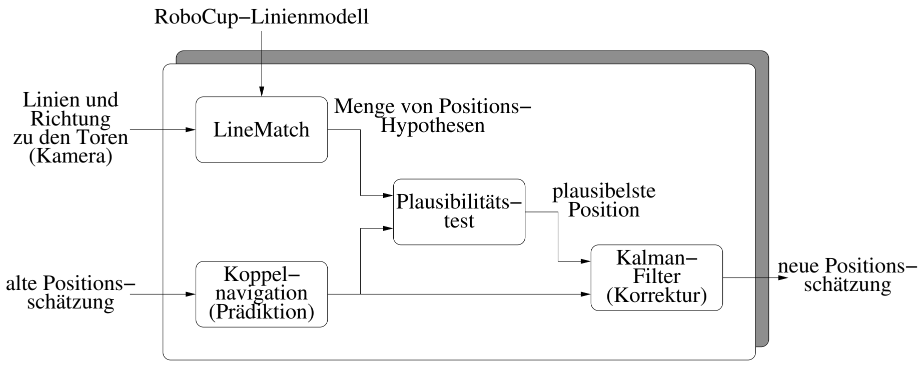


Abbildung 4.4: *Das Modul zur LineMatch-Lokalisierung*

Abbildung 4.4 zeigt das Schema der Selbstlokalisierung. Die durch den Line-Match-Algorithmus erzeugten Positionshypothesen werden einem Plausibilitätstest unterzogen. Bei diesem wird die erwartete Position einbezogen, die durch die Prädiktion gegeben ist. Als beste Position wird die Hypothese $h^* \in \mathcal{H}$ ausgewählt, deren zugehörige Aufnahmeposition μ_{h^*} den geringsten *quadrierten Mahalanobis-Abstand* zur erwarteten Position $\overline{pos} = (\overline{\mu}, \overline{\Sigma})$ hat:

$$h^* := \underset{h}{argmin}\big((\mu_h - \overline{\mu})^T \overline{\Sigma}^{-1} (\mu_h - \overline{\mu})\big) \quad (4.12)$$

Bei diesem Kriterium wird die Varianz in den einzelnen Dimensionen berücksichtigt. Ist sich der Roboter beispielsweise in x-Richtung seiner Position sehr sicher, wird eher eine Position ausgewählt, die in dieser Dimension nur leicht abweicht. Die Prädiktion $\overline{pos}$ und die beste Messung pos_{h^*} werden mit dem Kalman-Filter fusioniert und ergeben die neue Positionsschätzung.
Liefert *LineMatch* keine Hypothese, findet keine Kalman-Fusion statt, sondern der Wert $\overline{pos}$ wird direkt als neue Positionsschätzung übernommen. Das Gleiche geschieht, wenn der Mahalanobis-Abstand zwischen $\overline{pos}$ und pos_{h^*} einen bestimmten Grenzwert überschreitet. In diesem Fall gehen wir davon aus, dass die Hypothese h^* falsch ist.

Dieses Verfahren erlaubt eine zuverlässige lokale Lokalisierung, wenn die ungefähre Roboterposition bekannt ist.
Sind die Störeinflüsse gering, findet der LineMatch-Algorithmus oft nur eine Hypothese. In diesem Fall ist zu Beginn des Verfahrens sogar eine globale Lokalisierung möglich (siehe Kapitel 5). Dazu wird der Roboter auf eine beliebige Position des Spielfeldes gestellt und die Roboterposition mit den Werten

$$\mu_{start} = (0,0,0)^T \tag{4.13}$$

$$\Sigma_{start} = \begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix} \tag{4.14}$$

initialisiert. Die Kovarianzmatrix Σ_{start} definiert die größtmögliche Unsicherheit, wodurch die Position μ_{start} bei der ersten Ausführung des Kalman-Filters das Gewicht Null erhält.

4.1.5 Komplexitätsbetrachtung

Eine erste Abschätzung der Laufzeit des LineMatch-Verfahrens (Algorithmus 4.1 auf Seite 52) ergibt eine Komplexität von $O(|\mathcal{M}|^{|\mathcal{L}|})$, da jede Linie aus $\mathcal{L}$ jeder Modelllinie zugeordnet werden kann. Sind keine Ausreißerlinien zugelassen, lässt sich die Laufzeit allerdings durch eine genauere Betrachtung mit $O(|\mathcal{M}|^3\,|\mathcal{L}|^2)$ abschätzen [Weigel 1998, Gutmann 2000]. Das entscheidende Argument ist dabei, dass nur in den ersten beiden Rekursionsstufen wirklich alle möglichen Zuordnungen betrachtet werden müssen, da dann sämtliche Freiheitsgrade für Rotation und Translation festgelegt sind.
Wird eine Linie als Ausreißer markiert, geschieht durch diese Zuordnung keine Reduktion der Freiheitsgrade. Erlaubt man eine maximale Anzahl von k Ausreißerlinien, so sind im schlimmsten Fall erst nach $k + 2$ Stufen die Freiheitsgrade festgelegt. In diesen Rekursionsstufen gibt es daher $|\mathcal{M}|^{k+2}$ viele Paare. In der Folge müssen für jedes solche Paar weitere $|\mathcal{M}|\,|\mathcal{L}|$ Zuordnungen betrachtet werden. Da die Funktion $VerifyMatch$ $O(|\mathcal{L}|)$ viel Zeit benötigt, ergibt sich eine Gesamtlaufzeit von $O(|\mathcal{M}|^{k+3}\,|\mathcal{L}|^2)$.

In der Praxis besteht das Modell eines Robocup-Feldes aus 15 Linien. Wir lassen außerdem in Abhängigkeit von der Anzahl der erkannten Linien maximal 2 Ausreißerlinien zu. Mit einem strengen Fehlerkriterium MAX_{error} erreichen wir auf der in Abschnitt 1.4.1 beschriebenen Hardware eine durchschnittliche Laufzeit von etwa 3,5 ms. Durch das strenge Abbruchkriterium erhalten wir allerdings in ca. 8% der Fälle überhaupt keine Lösung.

4.2 Markov-Lokalisierung

Das zweite Verfahren, das wir in diesem Kapitel beschreiben wollen, ist die *Markov-Lokalisierung*, die für die globale Selbstlokalisierung in einer bekannten Umgebung ausgelegt ist [Fox 1998]. Bei ihr wird die Aufenthaltswahrscheinlichkeit des Roboters zu einem Zeitpunkt t durch eine beliebige Wahrscheinlichkeitsverteilung über dem Raum L aller möglichen Positionen beschrieben. Diese Verteilung wird der *Belief* des Roboters zum Zeitpunkt t genannt und mit $Bel(L_t)$ bezeichnet. Mit jeder neuen Sensormessung wird diese Verteilung aktualisiert und die globale Maximalstelle als aktuelle Positionsschätzung ermittelt. Dadurch, dass der Belief eine beliebige Verteilung ist, können mehrere (lokale) Maximalstellen repräsentiert werden. Wird durch weitere Sensorinformationen eine andere Stelle des Beliefs wahrscheinlicher, kann auf diese Positionsschätzung umgeschwenkt werden. Der Roboter behält sozusagen auch andere Positionshypothesen „im Hinterkopf”, anstatt, wie bei der Kalman-Filterung, nur eine einzige Hypothese zu verfolgen. Darauf beruht die Fähigkeit dieser Methode zur globalen Lokalisierung. Ein gutes Beispiel findet sich in der Arbeit von *Fox et al.* [Fox et al. 1999].

Die Wahrscheinlichkeit für den Roboter, zum Zeitpunkt t an Position l zu sein, ist abhängig von den bis dahin erhaltenen Sensordaten d_i:

$$Bel(L_t = l \mid d_0, \ldots, d_t) \tag{4.15}$$

Um den Lokalisierungs-Algorithmus zu präzisieren, muss festgelegt werden, wie sich dieser Wert mit einer neuen Sensormessung d_{t+1} ändert. Dabei folgt auch die Markov-Lokalisierung dem in Abbildung 4.1 auf Seite 47 dargestellten Prinzip. Je nachdem, ob es sich bei d_{t+1} um eine Odometrie- oder Umgebungsmessung handelt, wird zwischen Prädiktion und Korrektur unterschieden.
Um die zugehörigen Aktualisierungsgleichungen angeben zu können, wird die sogenannte *Markov-Annahme* benötigt:

> Die Wahrscheinlichkeit, an einer Position l eine bestimmte Sensormessung d_t zu erhalten, ist unabhängig von den vorherigen Messungen: $P(d_t \mid L_t = l, d_0, \ldots, d_{t-1}) = P(d_t \mid L_t = l)$

Zusammen mit dem *Satz von Bayes* und dem *Satz von der vollständigen Wahrscheinlichkeit* erhalten wir damit die folgenden rekursiven Update-Formeln [Fox et al. 1999]:

1. Fall: neue Umgebungsmessung s_t (Korrektur):

$$Bel(L_t = l) = \alpha_t \, P(s_t \mid l) \, Bel(L_{t-1} = l) \tag{4.16}$$

Dabei ist α_t ist eine Normalisierungskonstante, die sicherstellt, dass $\int Bel(L_t = l) \, dl = 1$ gilt. Die bedingte Wahrscheinlichkeit $P(s \mid l)$ nennen wir das *Sensormodell*, da sie die Zuverlässigkeit des Sensors beschreibt.

2. Fall: neue Odometriemessung a_t (Prädiktion):

$$Bel(L_t = l) = \int P(l \mid a_t, l')\, Bel(L_{t-1} = l')\, dl' \tag{4.17}$$

Hierbei wird $P(l \mid a, l')$ als *Bewegungsmodell* bezeichnet. Der Wert gibt die Wahrscheinlichkeit dafür an, dass sich der Roboter an Position l befindet, nachdem er Aktion a an Position l' ausgeführt hat. Dadurch lässt sich die Unsicherheit in den Roboterbewegungen modellieren.

Diese beiden Gleichungen bilden das Herzstück der Markov-Lokalisierung. Der vollständige Ablauf ist in Algorithmus 4.2 dargestellt.

Algorithmus 4.2 MarkovLocalization() [Fox et al. 1999]

for each location l **do** /* Initialisierung des Beliefs */
  $Bel(L_0 = l) \longleftarrow P(L_0 = l)$
endfor
forever do
  if sensory input s_t is received **do**
    $\alpha_t \longleftarrow 0$
    for each location l **do** /* Anwendung des Sensormodells */
      $\widehat{Bel}(L_t = l) \longleftarrow P(s_t \mid l) \cdot Bel(L_{t-1} = l)$
      $\alpha_t \longleftarrow \alpha_t + \widehat{Bel}(L_t = l)$
    endfor
    for each location l **do** /* Normalisierung des Beliefs */
      $Bel(L_t = l) \longleftarrow \alpha_t^{-1} \cdot \widehat{Bel}(L_t = l)$
    endfor
  endif
  if odometry reading a_t is received **do**
    for each location l **do** /* Anwendung des Bewegungsmodells */
      $Bel(L_t = l) \longleftarrow \int P(l \mid a_t, l') \cdot Bel(L_{t-1} = l')\, dl'$
    endfor
  endif
endforever

Bei dieser allgemeinen Formulierung der Markov-Lokalisierung ist offen gelassen, wie der Belief $Bel(L)$ repräsentiert wird. Die Verwendung von stückweise konstanten Funktionen, also Tabellen, ist oft zu langsam. Eine schnellere Variante ist die Verwendung eines *Sampling-Verfahrens*, wie es auch in unserer Implementierung geschieht. In diesem Fall spricht man von *Monte-Carlo-Lokalisierung*.

4.2.1 Monte-Carlo-Lokalisierung

Die *Monte-Carlo-Lokalisierung* (*MCL*) ist eine Variante der Markov-Lokalisierung. Bei ihr wird die Wahrscheinlichkeitsverteilung $Bel(L_t)$ durch eine Menge $\mathcal{S}_t = \{z_0, \ldots, z_n\}$ von Stichproben, sogenannten *Samples,* repräsentiert. Jedes Sample ist dabei ein Tupel $z_i = (l_i, g_i)$, wobei l_i eine Roboterposition und g_i das Gewicht des Samples ist. Die Wahrscheinlichkeit, dass ein Sample $z = (l, g)$ in der Menge $\mathcal{S}_t$ enthalten ist, sei proportional zu $Bel(L_t = l)$. Auf diese Weise ist die Sample-Menge eine diskrete Approximation des Beliefs. In Bereichen, in denen die Wahrscheinlichkeit hoch ist, liegen viele Samples, in Bereichen niedriger Wahrscheinlichkeit nur wenige. Zur Aktualisierung des Beliefs muss nicht mehr der gesamte Zustandsraum betrachtet werden, sondern „nur” die Sample-Menge in korrekter Weise aktualisiert werden. Derartige Verfahren werden im Allgemeinen als *Monte-Carlo-Verfahren* oder *Partikel-Filter* bezeichnet.
Das Grundprinzip bei der Aktualisierung der Sample-Menge ist ein Evolutionärer Algorithmus nach dem Prinzip *„Survival of the Fittest”*. Im Korrekturschritt wird jedes Sample $z_i \in \mathcal{S}_{t-1}$ gemäß dem Sensormodell $P(s_t \mid l_i)$ gewichtet:

$$g_i \longleftarrow P(s_t \mid l_i) \tag{4.18}$$

Im Prädiktionsschritt wird eine neue Sample-Generation $\mathcal{S}_t$ erzeugt. Dazu wird n-mal ein zufälliges Sample $z^* = (l^*, g^*) \in \mathcal{S}_{t-1}$ gezogen und zwar mit einer Wahrscheinlichkeit, die seinem Gewicht entspricht. Es haben also solche Samples eine größere Chance zu „überleben”, die sich durch ein großes Gewicht auszeichnen. Die neue Position des Samples z^* wird gemäß dem Bewegungsmodell $P(l \mid a_t, l^*)$ gewählt.

Ein typisches Szenario der globalen Lokalisierung mit MCL sieht wie folgt aus: Man startet mit einer zufälligen Sample-Menge $\mathcal{S}_0$. Alle Samples sind gleichmäßig über den Zustandsraum verteilt. Fängt der Roboter an, sich zu bewegen und die Umgebung zu messen, beginnen die Samples sofort, sich an den wahrscheinlichen Positionen zu sammeln. Werden durch weitere Sensormessungen die Mehrdeutigkeiten aufgelöst, bleibt schließlich nur ein einziger dichter Haufen von Samples übrig, der der Bahn des Roboters folgt.
Diese Art der Lokalisierung ist schneller und benötigt weniger Speicherplatz als die ursprüngliche Markov-Lokalisierung, liefert aber vergleichbare Ergebnisse.

In Rahmen einer Diplomarbeit von *Björn Fischer* ist die Monte-Carlo-Lokalisierung in das RoboCup-System des CS-Freiburg integriert worden [Fischer 2002]. Dabei werden die Daten des Laser-Scanners zur Korrektur des Beliefs benutzt. Wir wollen nun dieses Verfahren so erweitern, dass es auch die Merkmale benutzen kann, die wir durch die omnidirektionale Kamera erhalten.

4.2.2 Sensormodelle für die omnidirektionale Kamera

Die omnidirektionale Kamera liefert zu einem Zeitpunkt verschiedene Beobachtungen der Umgebung, die in unterschiedlichen Einheiten und mit unterschiedlicher Genauigkeit gemessen werden (siehe Kapitel 3). Man kann sie als Messungen mehrerer virtueller Sensoren betrachten. Wir unterscheiden die folgenden Beobachtungstypen:

- Entfernung eines Tores: D_{goal}
- Winkel zu einem Tor: Φ_{goal}
- Winkel zu einem Eckpfosten: Φ_{pole}
- Feldlinien: $Line$

Zur Gewichtung der Samples in Formel 4.18 wird das Sensormodell $P(s \mid l)$ benötigt. Wir gehen davon aus, dass die Messung s ein Vektor ist, der aus den einzelnen Beobachtungen $s_1, \dots, s_n$ eines Zeitpunktes besteht. Weiter machen wir die Annahme, dass alle diese Beobachtungen voneinander unabhängig sind. Damit ergibt sich:

$$P(s \mid l) = P(s_1, \dots, s_n \mid l) = P(s_1 \mid l) \cdot \ldots \cdot P(s_n \mid l) \tag{4.19}$$

Mit diesem Produkt werden bei der Monte-Carlo-Lokalisierung in Formel 4.18 die Samples gewichtet. Es sei aber darauf hingewiesen, dass es sich bei den Gewichten der Samples keineswegs um diese Wahrscheinlichkeiten handeln muss; vielmehr ist eine beliebige Bewertung der Sampleposition anhand einer Sensormessung möglich.

Es ist klar, dass der Wert $P(s_i \mid l)$ vom Typ der Beobachtung s_i abhängt. Daher muss für jeden Beobachtungstyp F ein eigenes Sensormodell $P_F(s \mid l)$ angegeben werden. Um dieses Modell zu vereinfachen, stellen wir folgende Überlegung an: Eine Messung s vom Typ F hängt nur von den statischen Objekten der Umwelt ab, die ihrem Typ entsprechen. An jeder Position ist die relative Position dieser Objekte eindeutig festgelegt und lässt sich anhand eines Umgebungsmodells berechnen. Damit ergibt sich also für eine Position l ein Menge von Erwartungswerten für Beobachtungen des Typs F. Diese Menge bezeichnen wir mit $\mathcal{E}_F(l)$. Betrachten wir zum Beispiel die gelben Eckpfosten, würde sie die beiden tatsächlichen Winkel enthalten, unter denen diese Eckpfosten von Position l aus zu sehen sind. Durch unsere Annahme, dass alle Beobachtungen voneinander unabhängig sind, können wir aber nicht wissen, welches Objekt (z. B. welcher Eckpfosten) unsere Messung verursacht hat. Daher vergleichen wir die Messung s nacheinander mit allen Erwartungswerten $e \in \mathcal{E}_F(l)$ und gehen davon aus, dass der richtige Erwartungswert derjenige ist, der die Wahrscheinlichkeit der Messung maximiert. Mit dieser Überlegung definieren wir:

$$P_F(s \mid l) := \max_{e \in \mathcal{E}_F(l)} P_F(s \mid e) \tag{4.20}$$

Der Wert $P_F(s \mid e)$ muss für die verschiedenen Beobachtungstypen F angegeben werden. Wir verabschieden uns aber an dieser Stelle von der Auffassung, dass es sich dabei um eine Wahrscheinlichkeit handeln muss, da wir – wie bereits erwähnt – bei der Gewichtung der Samples nicht darauf angewiesen sind. Eine Möglichkeit, die Funktion P_F zu definieren ist es, Heuristiken in Form von Dreiecksfunktionen zu benutzen, die an der Stelle e ihr Maximum haben [Enderle et al. 2000].
Wir versuchen im Folgenden einen anderen Weg zu gehen. Zunächst beschreiben wir das Modell für die eindimensionalen Messwerte vom Typ D_{goal}, Φ_{goal} und Φ_{pole}; anschließend erweitern wir dieses auf die mehrdimensionalen Messungen der Linien.

Entfernung zum Tor: D_{goal}
Winkel zum Tor: Φ_{goal}
Winkel zu einem Eckpfosten: Φ_{pole}

Das hier vorgestellte Fehlermodell orientiert sich grob an der entsprechenden Modellierung für Laser-Scanner [Fischer 2002]. Es soll die Kombination verschiedener Messungen ermöglichen.
Zunächst modellieren wir, dass die Messungen verrauscht sind. Dabei gehen wir von der üblichen Annahme aus, dass die Wahrscheinlichkeit einer Messung s normalverteilt um den erwarteten Wert e liegt:

$$p(s) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{s-e}{\sigma}\right)^2\right) \tag{4.21}$$

Den Faktor $\frac{1}{\sigma\sqrt{2\pi}}$ werden wir aber auf Grund der folgenden Überlegung streichen: Die Beobachtungstypen haben unterschiedliche Wertebereiche. Die Winkel werden in Grad im Intervall $[-180; 180]$ angegeben, Entfernungen in Millimetern zwischen 0 und einigen Tausend. Aus diesem Grund sind die entsprechenden Standardabweichungen σ sehr unterschiedlich und lassen sich nur schwer vergleichen. Beispielsweise ist $\sigma_D = 100mm$ ein guter Wert, wohingegen $\sigma_\Phi = 100°$ eine katastrophale Winkelgenauigkeit beschreibt. Nimmt man dagegen eine realistischere Standardabweichung von $\sigma_\Phi = 10°$, wird eine Winkelmessung zehn mal so stark gewichtet, wie eine Distanzmessung. Durch Streichen des Faktors erreichen wir eine Normalisierung der Gaußverteilung, so dass ihr Maximum unabhängig von der Standardabweichung 1 ergibt. Dass die resultierende Funktion keiner Wahrscheinlichkeitsverteilung mehr entspricht, soll uns nicht weiter stören, da wir – wie erwähnt – bei der Gewichtung der Samples nicht darauf angewiesen sind.
Als weitere Fehlerquelle nehmen wir Messungen an, bei denen ein falsches Objekt detektiert wird. Ein Beispiel hierfür ist, dass ein Zuschauer mit blauer Hose und gelbem T-Shirt als Eckpfosten erkannt wird. Um zu verhindern, dass in Folge einer Falschmessung eine Position unterbewertet wird, modellieren wir diesen Sachverhalt durch eine konstante Verteilung über dem Messraum: $p'(s) = c$

Diese beiden Verteilungen werden wie folgt zur Funktion $P_F(s \mid e)$ kombiniert:

$$P_F(s \mid e) = (1 - c) \exp\left(-\frac{1}{2}\left(\frac{s-e}{\sigma}\right)^2\right) + c \tag{4.22}$$

In Abbildung 4.5 ist diese Funktion in Abhängigkeit der Messung s dargestellt.

Da man anhand eines Umgebungsmodells alle erwarteten Messungen berechnen kann, erlaubt eine Messung eine Bewertung aller Roboterpositionen durch die Funktion 4.20. In Abbildung 4.6 sind diese Bewertungen bei konstanter Roboterausrichtung für drei unterschiedliche Messungen zu sehen. Wahrscheinlichere Positionen sind dunkler dargestellt. Von links nach rechts wurden ein Winkel zum Tor, eine Entfernung zum Tor und ein Eckpfosten erkannt. Da es zwei identische Eckpfosten gibt, bilden sich in diesem Fall zwei Kegel, in denen die Aufenthaltswahrscheinlichkeit hoch ist. Wäre die Farbe des Eckpfostens nicht erkannt worden, hätte man sogar vier Kegel.

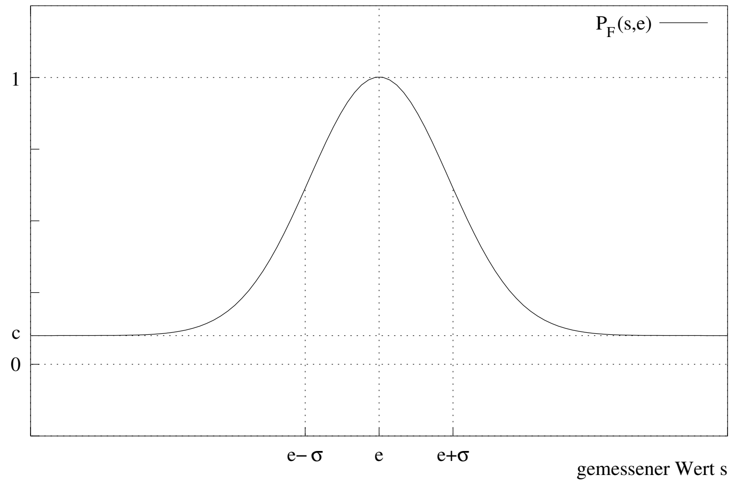
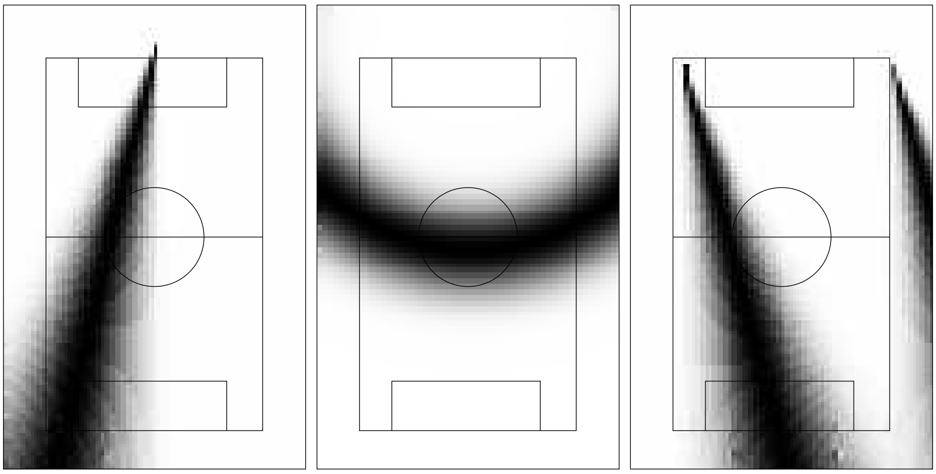


Abbildung 4.5: *Gewichtungsfunktion der Messung s bei erwartetem Wert e.*

Feldlinien: $Line$

Da sich die Linien nicht durch einen einzelnen Wert beschreiben lassen, werden sie getrennt behandelt. Wir betrachten zur Vereinfachung alle Liniensegmente als Geraden, wodurch sich jede Beobachtung s vom Typ $Line$ durch die zwei Werte Entfernung d_s und Richtung α_s beschreiben lässt.
Die Bestimmung des Gewichts $P_{Line}(s \mid e)$ findet in zwei Schritten statt. Mit Hilfe der Ausrichtung α_s bestimmen wir, wie gut eine gemessene Linie s zu einer erwarteten Linienmessung e passt und führen die eigentliche Gewichtung anhand des gemessenen Abstandes d_s und des erwarteten Abstandes d_e durch.



(a) Winkel zum Tor (b) Entfernung zum Tor (c) Winkel zu einem Eckpfosten

Abbildung 4.6: *Bewertung der Positionen bei konstanter Roboterausrichtung und einer einzelnen Messung.*

Konkret bedeutet das, dass wir das Gewicht $P(d_s \mid d_e)$ nach Formel 4.22 berechnen. Außerdem bestimmen wir eine Maßzahl q wie folgt:

$$q(s,e) = \begin{cases} 1 - \frac{\|\alpha_s - \alpha_e\|}{TOL_\Phi} & \text{falls } \|\alpha_s - \alpha_e\| \leq TOL_\Phi, \\ 0 & \text{sonst} \end{cases} \tag{4.23}$$

Das ist eine Dreiecksfunktion, wie sie in Abbildung 4.7 zu sehen ist. Mit diesem Faktor wird das Gewicht multipliziert und wir erhalten:

$$P_{Line}(s \mid e) = q(s,e)P(d_s \mid d_e) \tag{4.24}$$

Auf diese Weise werden Zuordnungen zwischen Linien und Modelllinien aussortiert, die mehr als TOL_Φ voneinander abweichen und nur Positionen, in denen die Richtungen von gemessener und erwarteter Linie übereinstimmen, erhalten das volle Gewicht. Als einen guten Wert für die Toleranzgrenze benutzen wir $TOL_\Phi = 10°$.

Mit diesem Vorgehen lassen sich auch mit Linienmessungen alle Positionen des Zustandsraumes nach Formel 4.20 bewerten. In den Beispielen in Abbildung 4.8 ist dabei die Roboterposition auf 0° fixiert, d. h. der Roboter blickt im Bild nach oben. Im ersten Fall beobachtet er etwa 2 m vor seiner Position eine Linie. Da er nicht wissen kann, welche Linie er sieht, bilden sich die drei eingezeichneten Streifen mit hoher Aufenthaltswahrscheinlichkeit. Im zweiten Beispiel liegt die detektierte Linie etwa 3 m rechts von der Roboterposition. Da die Linien als Geraden betrachtet werden, kann das entweder die rechte Seitenauslinie sein oder eine

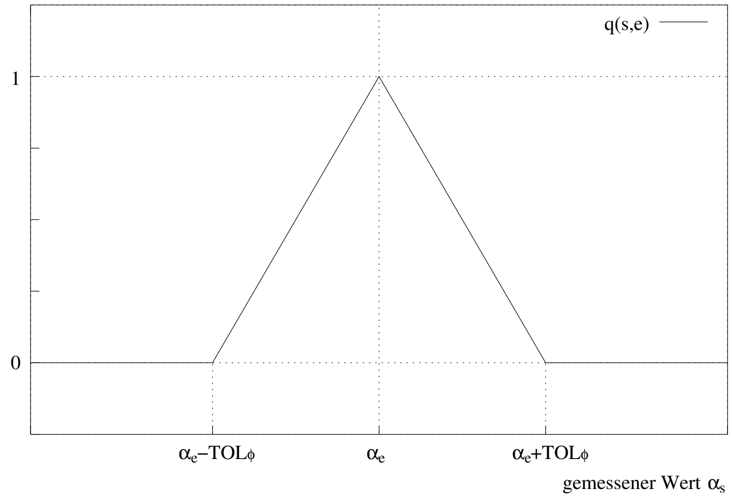
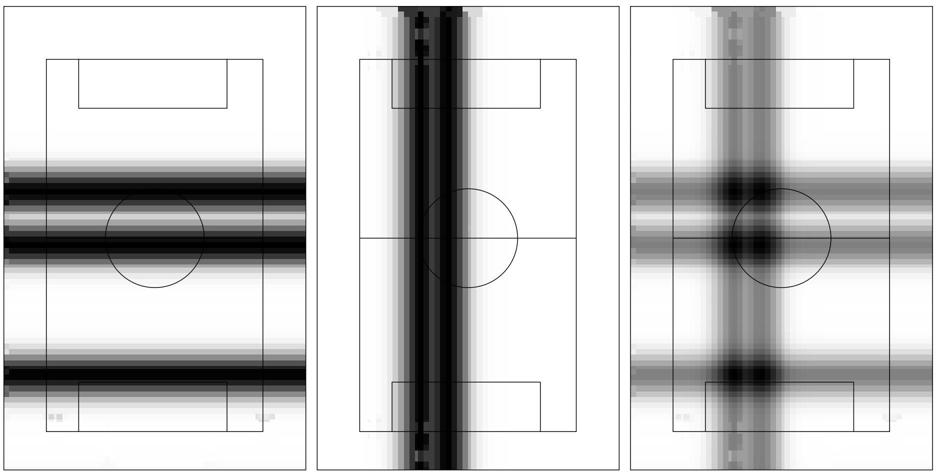


Abbildung 4.7: *Heuristik zur Gewichtung der Zuordnung einer Linie l zu einer Modelllinie e anhand der Ausrichtungen α der Linien.*

der rechten Strafraumbegrenzungen. Dadurch ergeben sich zwei dicht beieinander liegende senkrechte Streifen in der Positionsbewertung. Das dritte Bild zeigt die Kombination beider Beobachtungen.



(a) Eine Linie 2m vor dem Roboter

(b) Eine Linie 3m rechts vom Roboter

(c) Kombination der beiden Beobachtungen

Abbildung 4.8: *Bewertung der Positionen bei konstanter Roboterausrichtung und verschiedenen Messungen. Der Roboter blickt nach oben.*

4.2.3 Kombination von Beobachtungen

In Formel 4.19 werden die Gewichte aller Messungen eines Zeitpunktes durch Multiplikation verknüpft. Wir benutzen jedoch ein anderes Maß für die Gewichtung der Messungen, das wir mit $G_F(s \mid l)$ bezeichnen und verweisen dabei auf die Arbeit von Björn Fischer [Fischer 2002]:

$$G_F(s \mid l) \sim \ln(1 + P_F(s \mid l)) \tag{4.25}$$

Durch die Verwendung des Logarithmus müssen diese Gewichte nicht multipliziert werden, sondern können durch Addition verknüpft werden, was deutlich effizienter ist:

$$G(s \mid l) = G_{F_1}(s_1 \mid l) + \ldots + G_{F_n}(s_n \mid l) \tag{4.26}$$

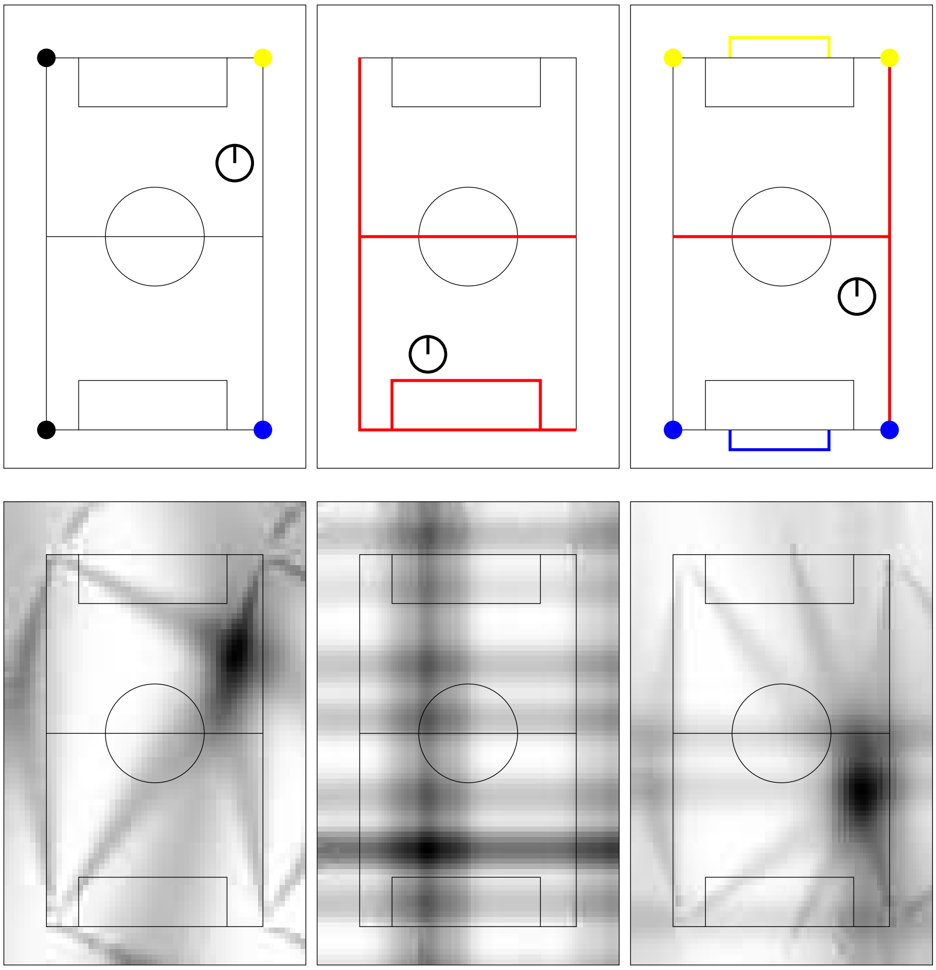
Eine Ausnahmebehandlung erfahren die Spielfeldlinien, die durch ihre große Anzahl die anderen Messungen überlagern. Die Gewichte der Linienmessungen werden durch ihre Anzahl geteilt, wodurch ihr Einfluss verringert wird. Eine andere Lösung für dieses Problem ist es, nicht alle Linienmessungen in die Bewertung mit einfließen zu lassen, sondern z. B. nur die beiden am nächsten zum Roboter gelegenen Linien zu verwenden.

Einige Beispiele für Positionswahrscheinlichkeiten, die sich dadurch ergeben, sind in Abbildung 4.9 zu sehen. Links sind die Beobachtungen von vier Eckpfosten kombiniert. Ein gelber und ein blauer Eckpfosten sind identifiziert worden, bei den anderen beiden ist die Farbe unbekannt. In der Mitte sind sechs verschiedene Linienmessungen kombiniert. Es gibt nur eine Position, in der zu jeder Beobachtung ein Erwartungswert gefunden wird, aber auch andere Positionen, in denen nur vier oder fünf Beobachtungen zutreffen, sind sehr wahrscheinlich. Im letzten Beispiel sind Beobachtungen verschiedenen Typs zusammengefasst. In diesem Fall bildet sich eine deutliche Positionshypothese im Zustandsraum heraus.
Bei all diesen Beispielen ist allerdings zu beachten, dass sich in der dritten Dimension, also in einer anderen Roboterausrichtung, weitere wahrscheinliche Positionen befinden können. Besonders im Fall der Linien kann das für die um 90° gedrehten Positionen gelten.

Mit diesem Vorgehen ist zwar eine globale Selbstlokalisierung möglich, doch sind die Ergebnisse noch nicht zufriedenstellend (siehe Kapitel 5).
Eine bessere Sensormodellierung und ein anderes Vorgehen bei der Kombination der Messungen kann in Zukunft zu besseren Ergebnissen führen. Eine denkbare Möglichkeit ist die Verwendung einer gewichteten Summe:

$$G(s \mid l) = \sum_{i=0}^{n} g(s_i) G(s_i \mid l) \tag{4.27}$$

wobei der Wert von g vom Typ und der Genauigkeit der Messung abhängt, etwa von der Länge einer Linie. Dadurch kann auch der Nutzen verschiedener Beobach-



(a) Vier Eckpfosten, bei zwei davon wurde die Farbe nicht erkannt

(b) Kombination vieler Linienbeobachtungen

(c) Zwei Linien und die Richtung zu den Toren und Eckpfosten

Abbildung 4.9: *Positionswahrscheinlichkeiten bei Kombination mehrerer Merkmale. Die Roboterausrichtung ist wieder konstant. Oben sind die tatsächliche Roboterposition und die Beobachtungen eingezeichnet.*

tungstypen berücksichtigt werden, was allerdings genauere Analysen erforderlich machen und den Rahmen dieser Arbeit sprengen würde.

4.2.4 Implementierung

Zur Beschleunigung der Gewichtung der Samples berechnen wir bei unserer Implementierung die Erwartungswerte und die Sensormodelle im Voraus.

Bei den Erwartungswerten wird dazu ein zweidimensionales Gitter mit einer Auflösung von 10 cm über die Umgebung gelegt. Für jede Gitterzelle speichern wir ein Array mit den zu erwartenden Messungen, wenn der Roboter sich in der Mitte der Gitterzelle mit Ausrichtung 0° befindet. Dieses Array nennen wir das *Erwartungsmodell*.
Die Sensormodelle $G(s \mid e)$ werden ebenfalls in zweidimensionalen Arrays gespeichert. Eine Dimension entspricht dabei dem gemessenen, die andere dem erwarteten Wert.
Zur Gewichtung eines Samples bei einer Messung s werden die erwarteten Werte $e_1, \ldots, e_n$ an der entsprechenden Stelle des Erwartungsmodells nachgeschlagen. Im Fall einer Winkelmessung muss von diesen Werten noch die Ausrichtung θ des Samples abgezogen werden. Für jedes Paar (s, e_i) wird daraufhin im Sensormodell das entsprechende Gewicht $G(s \mid e_i)$ bestimmt. Das Maximum dieser Gewichte wird zum Gewicht des Samples hinzu addiert und man kann mit der nächsten Messung fortfahren.
Dieses Prinzip entspricht dem Vorgehen bei der MCL mit dem Laser-Scanner [Fischer 2002]. Durch die Diskretisierung des Zustandsraumes und der Sensormodelle werden die Ergebnisse zwar ungenauer, der Zeitgewinn ist aber enorm.
Weitere Details der Implementierung bezüglich der Auswahl und Prädiktion der Samples finden sich in der Arbeit von *Björn Fischer* [Fischer 2002].

4.2.5 Komplexitätsbetrachtung

Die Laufzeit der Monte-Carlo-Lokalisierung hängt von der Größe der verwendeten Samplemenge $\mathcal{S}$ ab, durch die der Belief des Roboters repräsentiert wird. Für jedes Sample aus $\mathcal{S}$ muss die Gewichtung durchgeführt werden. Dabei muss wiederum jede Beobachtung mit jeder erwarteten Beobachtung verglichen werden. Wir bezeichnen die Menge der Beobachtungen mit $\mathcal{L}$ und die Menge der erwarteten Beobachtungen mit $\mathcal{M}$. Man beachte hierbei die Analogie zum LineMatch-Verfahren, bei dem mit $\mathcal{L}$ die Menge der beobachteten Linien und mit $\mathcal{M}$ die Menge der Modelllinien, also der zu erwartenden Messungen bezeichnet wurde (siehe Abschnitt 4.1.3).
Für die Gewichtung der Samplemenge $\mathcal{S}$ im Korrekturschritt der Monte-Carlo-Lokalisierung ergibt sich somit die Komplexität $O(|\mathcal{S}|\,|\mathcal{L}|\,|\mathcal{M}|)$. Im Prädiktionsschritt werden die Samples anhand ihres Gewichts ausgewählt und gemäß der Bewegung des Roboters verschoben. Dieser Schritt kann in $O(|\mathcal{S}|)$ durchgeführt werden. Ein Algorithmus dazu findet sich beispielsweise in der Arbeit von *Fischer* [Fischer 2002].
Die Gesamtlaufzeit des Verfahrens beträgt also $O(|\mathcal{S}|\,|\mathcal{L}|\,|\mathcal{M}|)$.

Kapitel 5

Ergebnisse und Diskussion

In diesem Kapitel beschreiben wir die Durchführung einiger Tests, die das Verhalten der beiden beschriebenen Lokalisierungsverfahren – LineMatch und Monte-Carlo-Lokalisierung – in der Praxis überprüfen sollen.
Der verwendete Roboter ist einer der Spieler des CS Freiburg, wie sie in Abschnitt 1.4.1 beschrieben sind. Nur die Kamera wurde durch die in Kapitel 2 entwickelte omnidirektionale Kamera ersetzt. Dieser Roboter ist auf der linken Seite von Abbildung 5.1 zu sehen. Die Tests fanden auf dem RoboCup-Spielfeld der Universität Freiburg statt. Dieses Spielfeld ist mit etwa 6,90 m auf 3,90 m kleiner als ein offizielles Spielfeld, die Größe des Mittelkreises und der Strafräume entsprechen aber den Regeln. Dieses Feld ist in Abbildung 5.1 gezeigt.

Abbildung 5.1: *Der Roboter mit der Omnicam (links) und das Spielfeld für unsere Tests (rechts)*

Um das Feld herum befindet sich in etwa 10 cm Abstand die Spielfeldbande, die bei Meisterschaftsspielen nicht mehr erlaubt ist. Sie ermöglicht es, das alte Lokalisierungsverfahren mit dem Laser-Scanner zu benutzen. Dieses Verfahren bietet eine globale Lokalisierung mit sehr exakten Positionsschätzungen, die im Mittel nur um 2 cm bzw. 1° von der tatsächlichen Position abweichen [Weigel et al. 2002]. Bei unseren Tests lief parallel stets dieses Lokalisierungsverfahren. Es liefert Positionsschätzungen, die als *Referenzpositionen* verwendet und mit den Ergebnissen der anderen Verfahren verglichen werden können. Dieses Vorgehen ermöglicht zwar keine absoluten Aussagen über die Genauigkeiten, aber dafür einen guten Vergleich zwischen unseren Verfahren unter realen Bedingungen.

Bei den Tests wurde den folgenden Fragen nachgegangen:

- Eignet sich das Verfahren zur globalen Lokalisierung?
  Dazu speziell bei der Monte-Carlo-Lokalisierung: Welche Merkmale sind für die globale Lokalisierung besonders wichtig?
- Wie gut ist die Lokalisierung?
  Bei der Monte-Carlo-Lokalisierung: Welche Merkmale führen zu größerer Genauigkeit?

Die Schwellenwerte zur Farbklassifizierung (siehe Abschnitt 3.1.2) wurden zu Beginn unserer Tests eingestellt und dann nicht mehr verändert, um für alle Versuche gleiche Bedingungen zu schaffen. Durch die Spielfeldbande werden einige Störeinflüsse durch schlecht eingestellte Farbklassen beseitigt. Auf der Bahn, die unser Roboter bei den Tests fuhr, konnte die Kamera jedoch über die Bande hinwegsehen. Gelegentlich kam es daher zu Falschmessungen, bei denen aus der Oberkante der Spielfeldbande eine Feldlinie extrahiert wurde oder Gegenstände im Regal (siehe Abbildung 5.1) als Eckpfosten erkannt wurden. Diese Fehler waren bei allen Tests gleich.

Bei der Monte-Carlo-Lokalisierung werden zu den Merkmalen verschiedene Werte für die Sensormodelle benötigt (siehe Abschnitt 4.2.2). Da wir noch keine statistischen Schätzungen über die tatsächlichen Genauigkeiten anfertigen konnten, wurden die Werte anhand einiger Testaufnahmen großzügig geschätzt und auf die Werte in Tabelle 5.1 gesetzt.

Beobachtung	σ	c	TOL_{Φ}
Richtung zu einem Tor: Φ_{goal}	8°	0,1	–
Entfernung zu einem Tor: D_{goal}	50 cm	0,1	–
Richtung zu einem Eckpfosten: Φ_{pole}	8°	0,1	–
Spielfeldlinien: $Line$	10 cm	0,05	10°

Tabelle 5.1: *Geschätzte Werte für die Sensormodelle, die wir in unseren Tests verwendet haben.*

Außerdem wurde die Anzahl der Samples für die MCL bei allen Versuchen auf 1000 Stück festgelegt. Diese Anzahl wählten wir auf Grund der Tests von *Björn Fischer* [Fischer 2002] als Kompromiss zwischen Geschwindigkeit und Genauigkeit der Positionsschätzung auf dem RoboCup-Feld. Auf die Untersuchung des Verhaltens der MCL bei verschiedenen Samplezahlen wurde in unseren Tests verzichtet. Nach Meinung des Autors sind beim derzeitigen Entwicklungsstand solche Tests noch nicht sinnvoll. Wichtiger ist es, den Einfluss der unterschiedlichen Merkmale auf die Genauigkeit der Positionsschätzungen zu untersuchen.

5.1 Laufzeiten

In den Abschnitten 4.1.5 und 4.2.5 haben wir die Komplexität der beiden Lokalisierungsverfahren abgeschätzt. Beim LineMatch-Verfahren kamen wir dabei auf $O(|\mathcal{M}|^{k+3}\,|\mathcal{L}|^2)$ und bei der Monte-Carlo-Lokalisierung auf $O(|\mathcal{S}|\,|\mathcal{L}|\,|\mathcal{M}|)$, wobei $|\mathcal{L}|$ die Anzahl der Beobachtungen, $|\mathcal{M}|$ die Anzahl der erwarteten Messungen, $|\mathcal{S}|$ die Anzahl der Samples und k die Anzahl der Ausreißerlinien ist. Aus dieser Sicht sieht die Laufzeit der MCL deutlich besser aus.
Wie bereits erwähnt, benötigen wir beim LineMatch-Verfahren eine durchschnittliche CPU-Zeit von 3,5 ms. Der dabei verwendete Computer besitzt 600 MHz und 112 MB Hauptspeicher.
Unsere MCL ist in dieser Hinsicht noch nicht ausreichend optimiert. Durch die Vorberechnung der Erwartungswerte und der Sensormodelle sparen wir zwar Rechenzeit (siehe Abschnitt 4.2.4), aber an anderen Stellen muss noch viel verbessert werden. Derzeit benötigen wir zur Gewichtung von 1000 Samples zwischen 30 und 120 ms CPU-Zeit. Diese Zeit hängt von der Art und Anzahl der verwendeten Merkmale ab. Die Beobachtung eines Tores muss nur mit einem einzigen Erwartungswert verglichen werden, eine Feldlinie jedoch mit jeder Modelllinie.
Diese langen Berechnungszeiten beeinträchtigen die Genauigkeit der Prädiktion und damit auch die Ergebnisse der folgenden Tests.

5.2 Globale Lokalisierung

Unser Test zur globalen Lokalisierung sieht wie folgt aus: Der Roboter wird an eine beliebige Stelle des Feldes gestellt und eingeschaltet. Von diesem Moment an läuft die Selbstlokalisierung. Die Positionsschätzung wird mit der Referenzposition verglichen und die Zeit gestoppt, bis eine globale Lokalisierung statt gefunden hat[1]. Das Kriterium für eine globale Lokalisierung ist dabei, dass die Abweichung zwischen Schätzung und Referenzposition maximal 50 cm bzw. 10° beträgt und diese Schranken auch in den darauf folgenden 10 Sekunden eingehalten werden. Dieses Kriterium mag recht locker erscheinen. Wie wir jedoch in Abschnitt 5.3 sehen werden, entstehen bei der Monte-Carlo-Lokalisierung oft einzelne Ausreißer in der Positionsschätzung. Diese würden eine enger gesetzte Schranke überspringen, allerdings ohne dass dabei die globale Positionshypothese verloren geht.

[1] Dieser Versuchsaufbau wird auch als das *„Wake-Up-Problem"* bezeichnet.

5.2.1 Monte-Carlo-Lokalisierung

An dieser Stelle sei angemerkt, dass die Implementierung der Monte-Carlo-Lokalisierung noch nicht ausgereift ist. Sie besitzt einige Mängel, die behoben werden müssen. Das Hauptproblem sind die langen Aktualisierungszeiten und eine geeignete Prädiktion der Samples, wenn sich der Roboter nicht bewegt. Im Rahmen der Möglichkeiten haben wir dennoch versucht, einige Aussagen über die Fähigkeiten des Verfahrens zu treffen.
Die Monte-Carlo-Lokalisierung startet mit einer gleichmäßigen Verteilung der Samples über dem gesamten dreidimensionalen Zustandsraum der möglichen Roboterpositionen. Bei unserer derzeitigen Implementierung findet keine ausreichende Streuung der Samples statt, wenn der Roboter nicht fährt. Mit anderen Worten, der Roboter muss sich bewegen, damit eine globale Lokalisierung möglich ist. Außerdem lassen sich manche Mehrdeutigkeiten im Belief des Roboters im Stand nicht auflösen, da sich unter diesen Umständen die Markov-Annahme nicht rechtfertigen lässt: Der Roboter erhält immer die gleichen Beobachtungen mit sehr ähnlichen Fehlern durch schlechte Beleuchtung oder Objekte außerhalb des Feldes. Ein Beispiel für eine Mehrdeutigkeit, die sich nicht auflösen lässt, ist in Abbildung 4.8(c) zu sehen. Misst der Roboter immer wieder die beiden gleichen Linien, kann er nie entscheiden, welche der Hypothesen die richtige ist. Deshalb lassen wir bei unseren Versuchen den Roboter nach dem Start langsam über das Feld fahren. Die Zeitmessung beginnt in dem Moment, in dem er losfährt.
Um zu testen, welche Merkmale sich besser zur globalen Lokalisierung eignen, haben wir verschiedene Kombinationen ausprobiert. Dabei hat sich schnell herausgestellt, dass die Entfernungsmessungen zu den Toren – zumindest in ihrer derzeitigen Genauigkeit und Verwendung – wenig hilfreich sind. Klar ist außerdem, dass mit den Feldlinien alleine, auf Grund der Symmetrie des Feldes, keine globale Lokalisierung möglich ist. Als sinnvolle Tests haben wir uns daher für die folgenden vier Kombinationen entschieden:

1. Die Richtungen zu den beiden Toren,
2. die Richtungen zu den Eckpfosten,
3. die beiden längsten sichtbaren Linien *und*, um globale Lokalisierung zu ermöglichen, die Richtungen zu den Toren,
4. alle Linien, die erkannt werden *und* die Richtungen zu den Toren.

Mit jeder dieser Kombinationen wurden zehn Versuche unternommen. Die globale Lokalisierung war jedesmal erfolgreich, allerdings wurde zum Teil sehr viel Zeit benötigt. Die gemessenen Zeiten sind in Abbildung 5.2 zusammengefasst. Die Werte sind sortiert und die durchschnittlich benötigte Zeit ist durch ein Linie markiert.

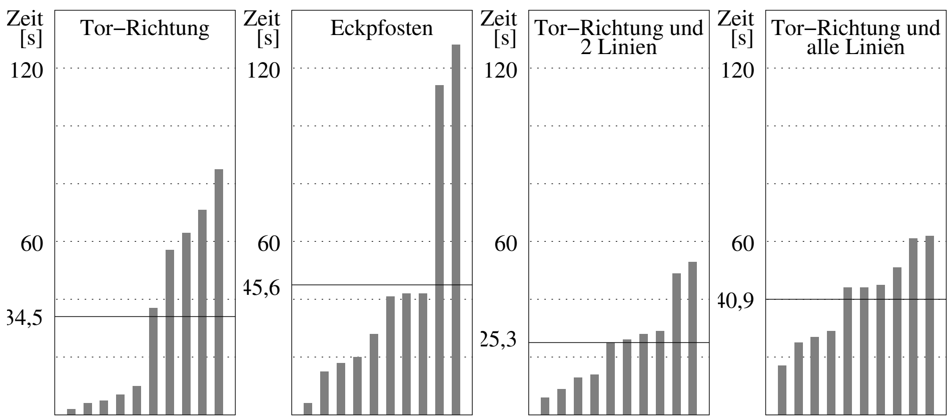


Abbildung 5.2: *Zeitdauer zur globalen Lokalisierung*

Erstaunlich ist, dass allein mit den Richtungen zu den Toren eine Lokalisierung möglich ist. Dies gilt allerdings nicht immer. In der Mitte des Feldes liefern die Winkel nämlich keinerlei Informationen über die Position des Roboters in x-Richtung. Je näher sich der Roboter am Rand des Feldes befindet, desto eindeutiger wird die Positionsschätzung. Mit dieser Tatsache lassen sich vermutlich die starken Schwankungen in der benötigten Zeit erklären. Sehr schnell wurde bei diesen Versuchen die Ausrichtung des Roboters gefunden, da die Tore zuverlässig erkannt wurden und sich die gemessenen Werte generell eindeutig den erwarteten Winkeln zuordnen lassen.
Die Eckpfosten lieferten bei unseren Versuchen keine guten Ergebnisse. Das kann zum einen daran liegen, dass gelegentlich falsche Eckpfosten in den Objekten außerhalb des Spielfeldes erkannt wurden und zum anderen daran, dass oft keine Identifizierung ihrer Farbe möglich war. Da wir bei unserem Vorgehen jeden solchen „unbekannten” Eckpfosten allen vier tatsächlichen Pfosten zuordnen können, entstehen viele Positionshypothesen, auch in verschiedenen Orientierungen des Roboters.
Die besten Ergebnisse wurden durch die beiden längsten Linien in Verbindung mit den Toren erzielt. Offenbar wird durch die Tore sehr schnell die Ausrichtung des Roboters und seine ungefähre Position bestimmt. Die Linien ermöglichen dann eine genauere Positionsbestimmung in x- und y-Richtung.
Durch Hinzunahme der übrigen Linien verschlechtern sich die Ergebnisse wieder, wie im vierten Diagramm von Abbildung 5.2 zu erkennen. Da wir keine Korrelationen zwischen den Linien betrachten, kann jede Linie jeder Feldlinie zugeordnet werden und es entstehen viele wahrscheinliche Stellen im Zustandsraum. Bis sich aus diesen Hypothesen durch die Tore die richtige Position herausbildet, wird einige Zeit benötigt.

Prinzipiell ist mit unserem Verfahren also eine globale Lokalisierung möglich, die benötigten Zeiten sind aber nicht akzeptabel. Allerdings war dabei auch die Anzahl der verwendeten Samples mit 1000 Stück gering. Bei einer größeren Samplezahl erzielten wir aber auch keine besseren Ergebnisse, da sich dann die Update-Zeiten verlängerten und die Streuung der Samples im Prädiktionsschritt ungenauer wurde. In dieser Richtung lässt sich in Zukunft sicher noch einiges verbessern.

5.2.2 LineMatch

Mit dem LineMatch-Verfahren wurde eine entsprechende Testreihe durchgeführt. Der Roboter wurde 20 mal an verschiedenen Positionen des Feldes aufgestellt und gestartet. Die Unkenntnis der Startposition wurde dabei durch eine Fehlerkovarianzmatrix modelliert, in der unendliche Varianzen stehen (siehe Abschnitt 4.1.4). Bei allen Versuchen wurde innerhalb kürzester Zeit die richtige globale Position gefunden. Meist geschah dies mit der ersten Hypothese des LineMatch-Algorithmus. Erwähnenswert sind nur zwei Ausreißer, bei denen der LineMatch-Algorithmus zu Beginn eine falsche Hypothese lieferte. Einmal wurden daher 1,3 Sekunden benötigt, ein anderes Mal 24 Sekunden bis die Position durch die darauf folgenden Messungen korrigiert war.
Wird das Sichtfeld des Roboters stark eingeschränkt oder nimmt man ihm die Information der Tore, ist eine globale Lokalisierung nicht mehr möglich.

5.2.3 Diskussion

Ein Vergleich der beiden Verfahren bezüglich ihrer Fähigkeit zur globalen Lokalisierung ist derzeit nicht angebracht, denn unsere Implementierung der MCL ist in dieser Beziehung – wie bereits erwähnt – noch nicht ausgereift. Was sich aber erkennen lässt, ist, dass die farbigen Tore bei der globalen Lokalisierung eine wichtige Rolle spielen, da durch sie die Symmetrie des Feldes gebrochen wird. Die Eckpfosten sind in dieser Beziehung weniger hilfreich, da sie sehr fehleranfällig sind und ihre eindeutige Identifizierung oft fehlschlägt.
Bei beiden Verfahren muss in Zukunft noch genauer untersucht werden, inwieweit sich eine Fehleinschätzung der Position korrigieren lässt. Dazu bietet sich das *Kidnapped-Robot-Problem* (siehe Seite 48) an. Bei ersten Versuchen mit dem LineMatch-Verfahren gelang es dem Roboter nach wenigen Sekunden, sich zu relokalisieren, während er sich bewegte. Versuche mit der Monte-Carlo-Lokalisierung wurden dazu nicht durchgeführt, da diese ihre derzeitigen Mängel schon beim Wake-Up-Problem offenbarte. Das Kidnapped-Robot-Problem stellt eine deutlich schwierigere Herausforderung dar.

5.3 Lokale Lokalisierung

Zum Testen der Genauigkeit der Selbstlokalisierung benutzen wir folgendes Experiment: Der Roboter fährt mehrmals mit mäßiger Geschwindigkeit ein Rechteck

auf dem Feld ab. Dabei lokalisiert er sich mit dem Laser-Line-Match-Verfahren und schreibt die Referenzpositionen in eine Datei. Gleichzeitig wird auch mit der zu testenden Lokalisierungsmethode die Position geschätzt und in einer Datei gespeichert. Diese Dateien können wir anschließend auswerten und die Unterschiede in den Schätzungen bestimmen. Die Aufzeichnung der Positionen beginnt dabei erst, wenn die globale Lokalisierung abgeschlossen ist. Das heißt insbesondere, dass wir bei der Monte-Carlo-Lokalisierung so lange warten, bis die ungefähre Roboterposition bestimmt ist. Danach findet nur noch ein Tracking der Position statt.
Eine typische Bahn – auch Trajektorie genannt – des Roboters während eines solchen Experiments ist im oberen Teil von Abbildung 5.3 dargestellt. Die grüne (gestrichelte) Linie zeigt die Trajektorie, die dabei von den Odometriesensoren gemessen wurde. Auf Grund von Odometriefehlern weicht sie immer mehr von der tatsächlichen Bahn ab. Dies verdeutlicht die Messfehler, mit denen die Selbstlokalisierungsverfahren umgehen müssen.
Darunter sind die Bahnen dargestellt, die zur gleichen Zeit durch die anderen Lokalisierungsverfahren geschätzt wurden: in der Mitte durch das OmniCam-LineMatch-Verfahren (im Folgenden zur Unterscheidung vom Laser-LineMatch-Verfahren auch kurz OLM genannt) und unten durch die Monte-Carlo-Lokalisierung (MCL). Bei der MCL wurden dabei die Tore, die Eckpfosten und die beiden längsten sichtbaren Linien verwendet – eine Kombination, die, wie wir sehen werden, nicht die besten Ergebnisse liefert.
Diese Darstellungen geben nur einen groben Überblick der Ergebnisse der Lokalisierung. Zur genaueren Analyse betrachten wir die Abweichungen von der Referenzposition. Dabei unterscheiden wir zwischen dem Distanzfehler Δd in der (x,y)-Ebene und dem Ausrichtungsfehler $\Delta\theta$. Der Verlauf dieser Fehler bei dem in Abbildung 5.3 dargestellten Experiment ist in Abbildung 5.4 gezeigt. Die Abweichung zwischen der von der Odometrie geschätzten Position und der Referenzposition steigt während der zehn gefahrenen Runden stetig an, da sich die Messfehler akkumulieren. Deutlich zu erkennen ist außerdem, dass der Distanzfehler – zumindest in diesem Fall – beim OLM-Verfahren kleiner ist, als bei der MCL. Bei den Ausrichtungen dagegen sind die Unterschiede nicht so signifikant, die MCL scheint aber im Mittel bessere Ergebnisse zu liefern. Dies werden wir aber im Folgenden genauer untersuchen.

5.3.1 Monte-Carlo-Lokalisierung

Wie bei der globalen Lokalisierung stellt sich auch hier die Frage, welche Merkmale der Umgebung sich besser zur lokalen Lokalisierung eignen. Deshalb haben wir wieder verschiedene Merkmale und Kombinationen getestet. Jedesmal fuhr der Roboter fünf Rechtecke auf dem Feld ab. Wir haben zu den Distanz- und Ausrichtungsfehlern jeweils den Mittelwert μ, die Standardabweichung σ und den minimalen und maximalen Fehler bestimmt. Diese Ergebnisse sind in Tabelle 5.2 zu-

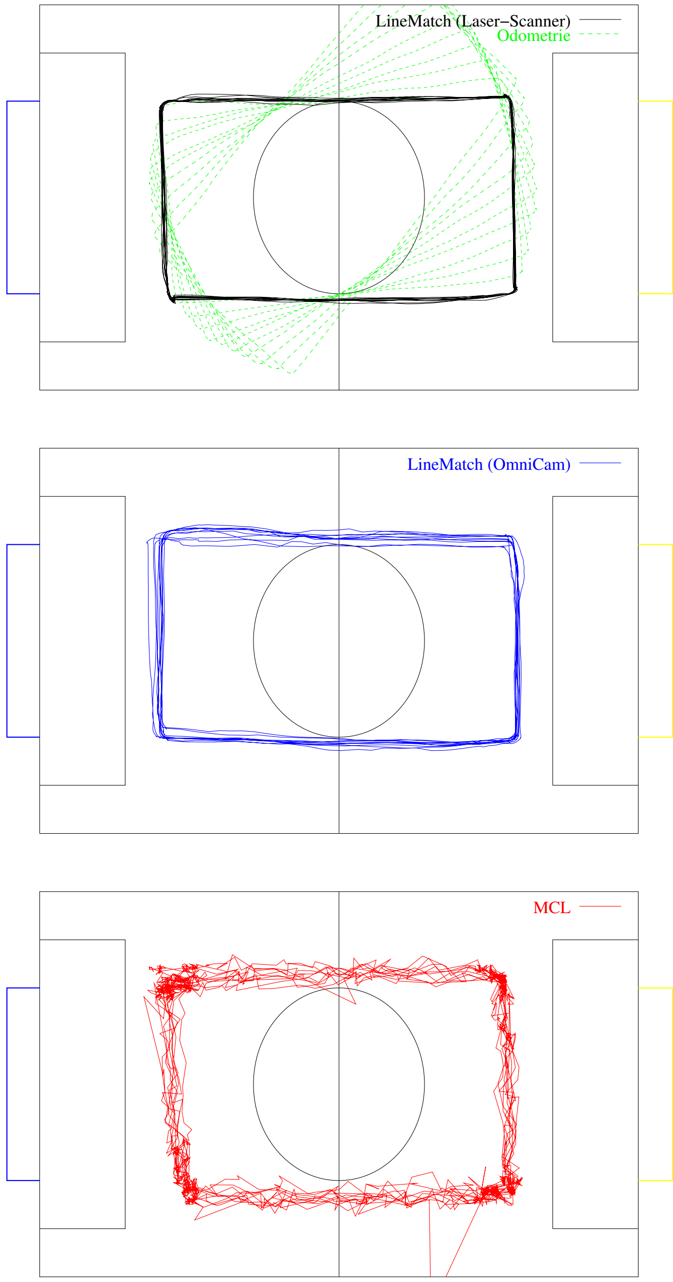


Abbildung 5.3: *Geschätzte Trajektorien des Roboters bei verschiedenen Lokalisierungsverfahren. Oben: „Tatsächliche" Position durch LineMatch-Lokalisierung mit dem LaserScanner bestimmt (die gestrichelte Linie ist die von der Odometrie gemessene Strecke). Mitte: LineMatch-Lokalisierung mit der OmniCam. Unten: Monte-Carlo-Lokalisierung.*

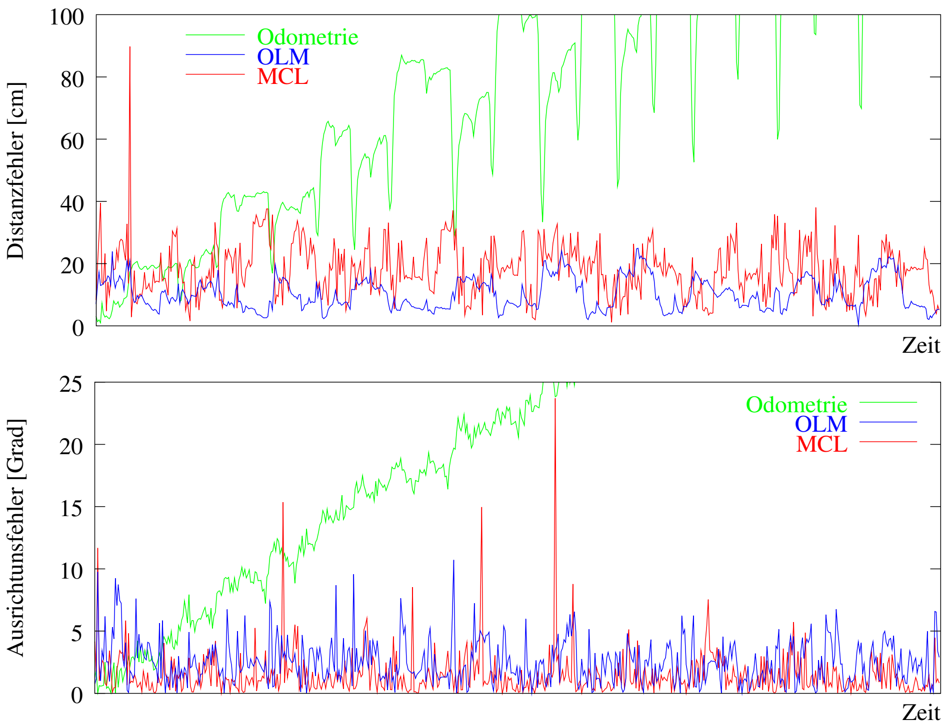


Abbildung 5.4: *Distanz- (oben) und Ausrichtungsfehler (unten) zwischen der Referenzposition und der geschätzten Roboterposition bei den verschiedenen Lokalisierungsverfahren. Der Roboter fuhr dabei zehn Rechtecke ab.*

sammengefasst. Außerdem sind die Mittelwerte und Varianzen in Abbildung 5.5 graphisch dargestellt.

Die Tore und Eckpfosten liefern offenbar keine ausreichenden Informationen über die (x,y)-Position des Roboters. Bei den Eckpfosten kommen außerdem große Fehler in der Richtungsschätzung hinzu. Möglicherweise werden zu viele falsche Eckpfosten erkannt, oder die Genauigkeit der Messungen ist deutlich schlechter als geschätzt.
Beim Tracking anhand von zwei Feldlinien (in diesem Fall die beiden längsten erkannten Linien), verlor der Roboter vorübergehend seine tatsächliche Position und schätzte sie um knapp 4 Meter auf dem Feld versetzt, wie an dem maximalen Fehler von 388 cm zu erkennen ist. Dies kann bei Verwendung von nur zwei Linien leicht vorkommen, da dadurch im Zustandraum mehrere gleich bewertete Positionen entstehen (siehe Abbildung 4.8(c)). In unserem Versuch korrigierte der Roboter seine Schätzung nach wenigen Sekunden. Daher ist der mittlere Distanzfehler zwar gering, aber die Varianz sehr hoch. Die Roboterausrichtung wurde bei diesem Test recht gut bestimmt.
Bei Verwendung aller Linien gelang das Tracking der Position sehr gut. Sowohl der mittlere Distanz- als auch der Ausrichtungsfehler sind mit 7,5 cm bzw. 1,2°

Merkmale		μ	σ	min	max
Tore	Δd [cm]	26,50	7,36	3,21	56,67
	$\Delta\theta$ [°]	1,66	1,54	0,00	18,54
Eckpfosten	Δd [cm]	24,23	17,21	0,13	122,18
	$\Delta\theta$ [°]	3,66	3,23	0,00	31,58
2 Linien	Δd [cm]	10,93	27,01	0,18	388,56
	$\Delta\theta$ [°]	1,39	1,14	0,00	8,89
alle Linien	Δd [cm]	7,47	4,15	0,22	47,51
	$\Delta\theta$ [°]	1,19	1,04	0,00	12,84
Tore, Eckpfosten	Δd [cm]	17,17	8,26	0,20	91,06
und 2 Linien	$\Delta\theta$ [°]	1,65	1,65	0,00	16,11
Tore, Eckpfosten	Δd [cm]	19,59	9,07	0,05	102,44
und alle Linien	$\Delta\theta$ [°]	1,76	2,54	0,00	44,41

Tabelle 5.2: *Genauigkeit der MCL bei Verwendung verschiedener Merkmale: Die Tabelle zeigt den mittleren Fehler μ in der Ebene (Δd) und in der Roboterausrichtung ($\Delta\theta$), zusammen mit der jeweiligen Standardabweichung σ und dem minimalen bzw. maximalen Fehler.*

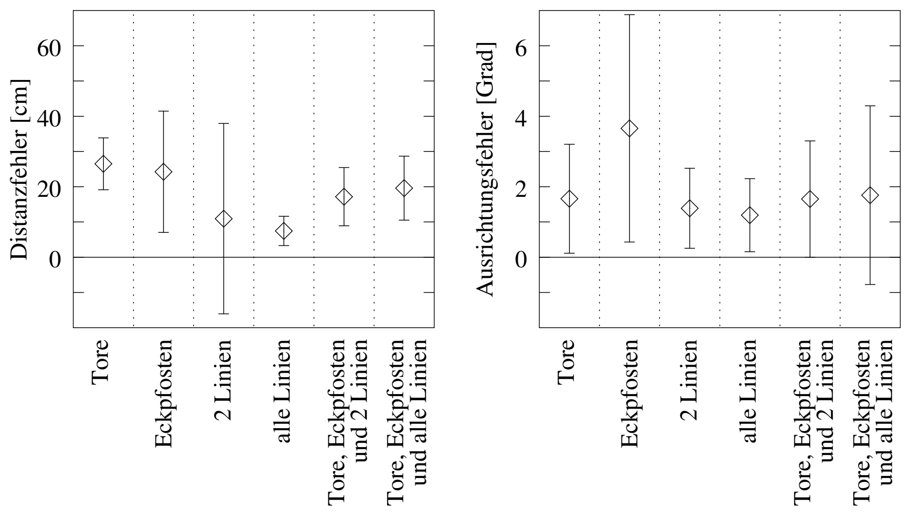


Abbildung 5.5: *Mittlere Abweichung von der Referenzposition bei Monte-Carlo-Lokalisierung mit verschiedenen Merkmalen. Der Fehlerbalken stellt die Standardabweichung dar.*

gering.
Die Kombination der Merkmale führte bei uns zu einer Verschlechterung der Positionsschätzungen. Die Fehler, die durch die Eckpfosten entstehen, wirken sich offenbar sehr stark aus. Dies deutet darauf hin, dass die Kombination der einzelnen Merkmale in der derzeitigen Form nicht zufriedenstellend gelöst ist.

Bei all diesen Versuchen sind einzelne Ausreißer zu verzeichnen, die sich durch hohe Werte in der max-Spalte niederschlagen. Eine mögliche Erklärung hierfür ist die Folgende: Damit eine Position mit der Referenzposition verglichen werden kann, müssen beide zum gleichen Zeitpunkt vorliegen. Daher werden beide Schätzungen anhand der Geschwindigkeiten des Roboters auf den selben Zeitpunkt projiziert. Sind die Zeitpunkte der Schätzungen sehr unterschiedlich, wie es auf Grund der langen Berechnungszeiten bei der MCL der Fall sein kann, entstehen dadurch Fehler. Diese sind vermutlich bei Geschwindigkeitsänderungen besonders groß, wodurch die Ausreißer verursacht werden.

Festzuhalten bleibt, dass die Feldlinien eine wichtige Rolle für die Genauigkeit der Positionsschätzungen spielen. Allerdings ist die Verwendung von nur zwei Linien möglicherweise nicht robust genug.

5.3.2 LineMatch

Der Test zur lokalen Lokalisierung mit dem LineMatch-Verfahren hat zu den Ergebnissen in Tabelle 5.3 geführt. Darunter ist zum Vergleich das *beste* Ergebnis der Monte-Carlo-Lokalisierung aufgeführt.

Leeres Feld		μ	σ	min	max
LineMatch	Δd [cm]	10,31	4,11	1,63	24,42
	$\Delta\theta$ [°]	2,22	1,48	0,01	10,39
MCL: alle Linien	Δd [cm]	7,47	4,15	0,22	47,51
	$\Delta\theta$ [°]	1,19	1,04	0,00	12,84

Tabelle 5.3: *Genauigkeit des LineMatch-Verfahrens bei der lokalen Lokalisierung. Zum Vergleich sind die entsprechenden Werte der Monte-Carlo-Lokalisierung angegeben.*

Die Werte des LineMatch-Verfahrens sind leicht schlechter, als die der Monte-Carlo-Lokalisierung. Besonders auffällig ist der Unterschied in der Ausrichtung θ. Offenbar gleicht das Vorgehen der MCL die Fehler in den einzelnen Linien besser aus, als das LineMatch-Verfahren.
Erwähnenswert ist noch, dass die Schwankungen der Fehler beim LineMatch-Verfahren weniger abrupt sind. Dies kann man an den maximalen Abweichungen in der letzten Spalte der Tabelle erkennen, oder deutlicher in den Abbildungen 5.3 und 5.4.

5.3.3 Statische Objekte

In einem weiteren Versuch sollte getestet werden, wie die Verfahren mit Messungenauigkeiten umgehen können. Dazu haben wir mehrere Holzkisten der Größe

50cm×70cm auf dem Feld verteilt, wie in Abbildung 5.6 dargestellt. Diese Objekte sollen die Messungen stören, indem sie Tore, Eckpfosten und Linien ganz oder teilweise verdecken, wie es während eines RoboCup-Spiels durch andere Roboter vorkommt. Der Roboter fuhr die gleiche Bahn, wie bei den vorherigen Versuchen.

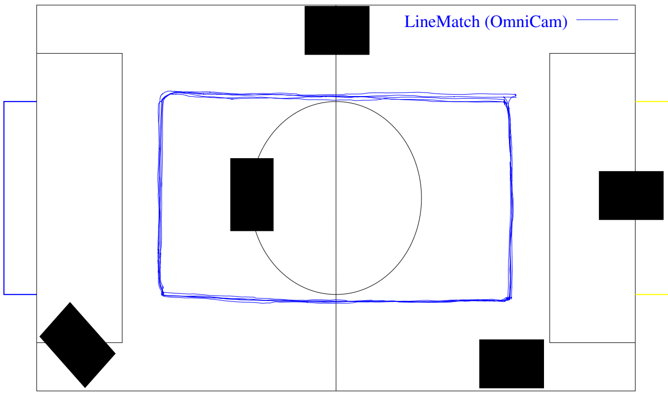


Abbildung 5.6: *Mit Kisten bestücktes Feld: durch sie werden unterschiedliche Merkmale der Umgebung verdeckt.*

mit Objekten		μ	σ	min	max
LineMatch	Δd [cm]	10,10	4,15	0,86	26,46
	$\Delta\theta$ [°]	2,23	1,54	0,00	13,27
MCL: alle Linien	Δd [cm]	8,02	5,21	0,22	72,07
	$\Delta\theta$ [°]	1,49	1,42	0,00	14,31
MCL: Tore, Eckpfosten und 2 Linien	Δd [cm]	12,74	12,30	0,16	80,55
	$\Delta\theta$ [°]	1,64	1,63	0,00	25,23

Tabelle 5.4: *Vergleich der Genauigkeit von OLM und MCL auf einem Feld mit Objekten*

Die Ergebnisse in Tabelle 5.4 zeigen keine signifikanten Unterschiede zur Lokalisierung ohne Hindernisse. Möglicherweise reicht die Anzahl der Objekte nicht aus, um genug Verdeckungen zu erzeugen.

5.3.4 Härtetest

In einem abschließenden Versuch wurde das Verhalten der Lokalisierung unter extremen Bedingungen untersucht. Durch starkes Beschleunigen, Abbremsen und schnelles Drehen kommt es zu großen Odometriefehlern. Außerdem gerät der Roboter – und damit die Kamera – ins Schwanken, wodurch grobe Messfehler bei der Merkmalsextraktion auftreten. Der Roboter wurde von Hand bei teilweise maximalen Translations- und Rotationsgeschwindigkeiten über das Feld gesteuert, auf dem sich auch die Holzkisten befanden. Die Trajektorien, die dabei vom Laser-LineMatch- bzw. Omnicam-LineMatch-Verfahren gemessen wurden, sind in Abbildung 5.7 dargestellt. Tabelle 5.5 zeigt die mittleren Abweichungen zwischen diesen beiden Positionsschätzungen.

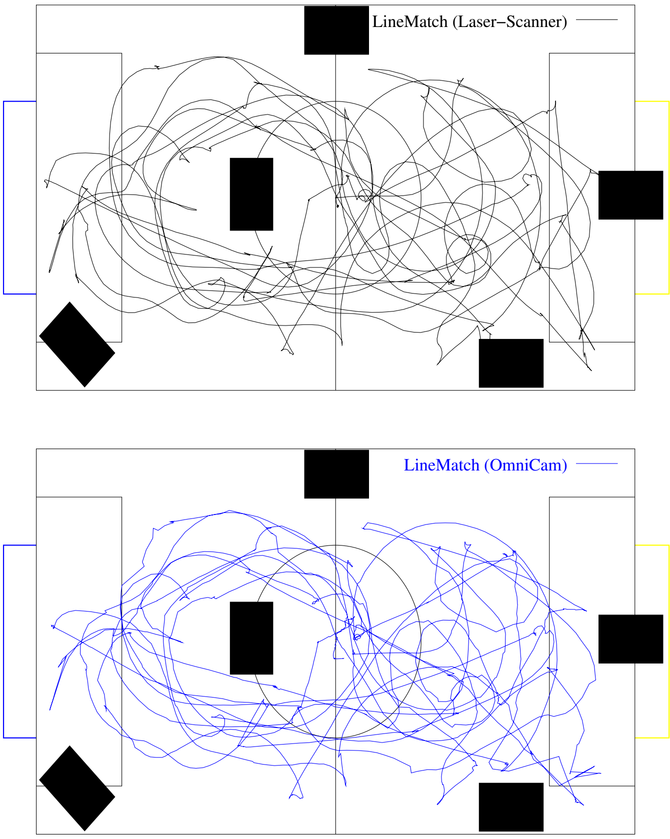


Abbildung 5.7: *Trajektorie des Roboters beim abschließenden Härtetest. Oben: Referenzpositionen, die mit der Laser-LineMatch-Lokalisierung bestimmt wurden. Unten: Positionsschätzungen des OLM-Verfahrens.*

Härtetest		μ	σ	min	max
LineMatch	Δd [cm]	15,85	11,34	0,23	108,19
	$\Delta\theta$ [°]	4,76	10,33	0,00	89,36

Tabelle 5.5: *Ergebnis des Härtetests mit dem LineMatch-Verfahren*

Dazu muss erwähnt werden, dass die Positionsschätzung des OLM-Verfahrens einige Sekunden um 90° gedreht war. Dies geschah in der Spielfeldecke unten rechts in Abbildung 5.7, als der Roboter eine Drehung ausführte, ohne dabei die Tore sehen zu können. Dies darf eigentlich nicht passieren. Das Problem lässt sich wahr-

scheinlich durch eine andere Auswahl der plausibelsten Positionshypothese lösen. Möglicherweise muss die Ausrichtung der Positionen beim Mahalanobis-Abstand (Formel 4.12) anders gewichtet werden. Als der Roboter in unserem Versuch wieder aus der Ecke herausfuhr, korrigierte er sein Position schnell.
Ein ähnlicher Test mit der Monte-Carlo-Lokalisierung scheiterte. Der Roboter schätzte seine Position oft um bis zu 180° gedreht oder um mehrere Meter versetzt und konnte nur bei langsamer Fahrt eine Korrektur vornehmen. Vermutlich liegt dies an dem verwendeten Bewegungsmodell, bei dem die Translations- und Rotationsmessungen der Odometrie nacheinander und unabhängig betrachtet werden [Fischer 2002]. Unsere langen Berechnungszeiten führen zu großen Zeitintervallen zwischen den einzelnen Prädiktionsschritten. In Verbindung mit hohen Geschwindigkeiten kann dies zu einer sehr schlechten Vorhersage der Samples führen.

5.3.5 Diskussion

Mit beiden in dieser Arbeit beschriebenen Verfahren ist eine lokale Lokaliserung möglich. Das LineMatch-Verfahren erlaubt durch den Kalman-Filter eine schnelle Integration neuer Messungen und ist damit auch zur Lokalisierung unter realen Bedingungen in der Lage. Die Monte-Carlo-Lokalisierung liefert unter optimalen Bedingungen zwar leicht bessere Ergebnisse, aber für den Einsatz beim RoboCup muss sie noch deutlich optimiert werden.
Auch wenn aus unseren Tests keine absoluten Genauigkeiten hervorgehen, so wird doch deutlich, dass die Genauigkeit beider Lokalisierungen geringer ist als beim Laser-LineMatch-Verfahren. Dies war aber auch zu erwarten. Unsere Ergebnisse in Tabelle 5.3 sind allerdings überraschend gut, vergleicht man sie mit denen der MCL mit dem Laserscanner. Für diese wurde bei 1000 Samples auf einem leeren Feld ein mittlerer Fehler von 16,62 cm bzw. 11,38° ermittelt [Fischer 2002]. Zur Bestimmung dieser Werte wurde allerdings auch eine andere Methode benutzt, bei der die Abweichungen von Hand gemessen wurden. Wahrscheinlich sind diese Ergebnisse daher nicht direkt vergleichbar.

Gewisse Fehler in der Positionsschätzung lassen sich bei Verwendung unserer Kamera nicht vermeiden. Durch Ungenauigkeiten im Aufbau entstehen systematische Fehler. Da sich Kamera und Spiegel nicht exakt zueinander ausrichten lassen, ist die Kameratransformation nicht wirklich eine Isometrie. Dieses Problem wird durch Unebenheiten des Bodens verstärkt. Gerät der Roboter auf schiefem Boden in Schräglage, ändert sich die Verzerrung des Bildes leicht. Wird ein Objekt nur um ein Pixel verschoben dargestellt, ändert sich die Distanzschätzung schon um über 5 cm. Das Gleiche geschieht, wenn der Roboter bei Beschleunigung ins Schwanken gerät. Diese Fehler lassen sich in der Praxis nicht vermeiden, beeinflussen aber die Genauigkeit der Positionsschätzung.
Das OLM-Verfahren bietet diesbezüglich auf Grund seiner einfachen Struktur wenig Verbesserungsmöglichkeiten. Bei der MCL lässt sich durch Einstellen der Parameter viel erreichen.

Kapitel 6

Zusammenfassung und Ausblick

Mit dieser Arbeit wurde eine omnidirektionale Kamera zur Selbstlokalisierung in das RoboCup-System des CS Freiburg integriert. Wir haben dabei den Prozess vom Aufbau der Kamera mit dem Entwurf eines Spiegels über die Extraktion der Spielfeldlinien, Eckpfosten und Tore bis zur Selbstlokalisierung beschrieben. Im letzten Kapitel haben wir die beiden vorgestellten Lokalisierungsverfahren auf ihre Tauglichkeit hin untersucht.

Die LineMatch-Lokalisierung ist in ihrer derzeitigen Form bereits zur Lokalisierung bei einem RoboCup-Spiel einsetzbar. Der einzige Ansatzpunkt zur Verbesserung ist im Moment die Auswahl der plausibelsten Hypothese. Bei der Verwendung des Mahalanobis-Abstandes scheinen die Winkel problematisch zu sein. Wie robust das Verfahren auf einem größeren Feld und unter anderen Lichtverhältnissen funktioniert, ist noch genauer zu untersuchen. Außerdem muss sich erst noch herausstellen, inwieweit das Gesamtsystem des CS Freiburg unter der erhöhten Unsicherheit leidet. Dieses System ist auf die sehr exakten Positionsschätzungen des Laser-LineMatch-Verfahrens abgestimmt.

Die Monte-Carlo-Lokalisierung ist in ihrer Entwicklung noch nicht abgeschlossen. Ansätze zur Verbesserung bieten die Geschwindigkeit des Korrekturschrittes und die Vorhersage der Samples. Das verwendete Bewegungsmodell [Fischer 2002] ist angesichts der großen Zeitintervalle zwischen den Prädiktionen zu ungenau. Möglicherweise müssen die Odometriemessungen in kürzeren Zeitintervallen verarbeitet werden.
Ein weiterer Punkt, der überdacht werden sollte, sind die Sensormodelle. Zu ihrer Realisierung haben wir ein sehr einfaches Modell benutzt. Eine ausgefeiltere Methode kann in Zukunft zu besseren Ergebnissen führen. Allerdings sind die Modelle auch in ihrer derzeitigen Form noch nicht optimiert, die Genauigkeiten der verschiedenen Beobachtungen wurden nur grob geschätzt.
Zur Vereinfachung haben wir die Spielfeldlinien bisher als Geraden angenommen, die durch zwei Parameter definiert sind. In Zukunft ist es sicher besser, wenn man Liniensegmente betrachtet. Zur Beschreibung eines Liniensegments benötigt man

vier Parameter, die bei der Gewichtung der Samples in geeigneter Weise berücksichtigt werden müssen.
Ein interessanter Aspekt ist, den Nutzen unterschiedlicher Beobachtungen in die Lokalisierung mit einfließen zu lassen, wie es in Abschnitt 4.2.3 angedeutet wurde. Bei der globalen Lokalisierung spielen die Tore eine entscheidende Rolle, zur lokalen Lokalisierung sind die Spielfeldlinien sehr wichtig. Durch die Beschränkung auf Merkmale, die in der jeweiligen Situation hilfreich sind, kann die MCL beschleunigt werden.
Sehr hilfreich bei der Entwicklung der Monte-Carlo-Lokalisierung war der RoboCup-Simulator *SimServer*, der am Lehrstuhl für Künstliche Intelligenz von Alexander Kleiner entwickelt wird [Kleiner 2002]. Bei diesem Projekt werden das Spielfeld, die Roboter-Hardware und die Sensordaten durch ein Programm simuliert. Über eine TCP/IP-Verbindung kann die Client-Software eines Roboters mit diesem Simulationsserver verbunden werden, ohne dass man einen echten Roboter benötigt. Parallel zu unserer Arbeit haben wir für den SimServer ein Plugin programmiert, das die Beobachtungen der omnidirektionalen Kamera simuliert. Anhand dieser Daten war auf dem Simulator eine erfolgreiche Monte-Carlo-Lokalisierung möglich. Das System kann bei der Weiterentwicklung der MCL sehr nützlich sein.

Der große Vorteil der Monte-Carlo-Lokalisierung gegenüber dem LineMatch-Verfahren ist die Erweiterbarkeit. Durch die allgemeine Struktur ist es einfach, weitere Beobachtungen in die Gewichtung der Samples einfließen zu lassen. Diese können sogar von anderen Sensoren stammen. Man muss dazu „nur” die erwarteten Messungen und ein geeignetes Sensormodell angeben. Eine Möglichkeit ist zum Beispiel, die Entfernung der Tore mit dem Laser-Scanner zu bestimmen. Eine solche Messung wäre sehr exakt und könnte die Positionsschätzung deutlich verbessern. Eine wichtige Rolle kann in Zukunft auch der Mittelkreis des Feldes spielen. Uns ist leider keine effiziente Erkennung dieses eindeutigen Merkmals möglich gewesen. Gelingt dies jedoch (z. B. durch Hough-Transformation), ist es leicht, die MCL dementsprechend zu erweitern.

Ein Nachteil der Monte-Carlo-Lokalisierung ist, dass wir derzeit keine Abhängigkeiten zwischen den Beobachtungen eines Typs betrachten. Besonders die Feldlinien sind aber stark korreliert. Eine Idee ist deshalb, die beiden Lokalisierungsverfahren zu kombinieren: der LineMatch-Algorithmus liefert Positionshypothesen, die zur Gewichtung der Samples bei der MCL benutzt werden. Dadurch wird die Korrelation der Linien berücksichtigt, ohne dass der Nachteil des Kalman-Filters gegeben ist, nur eine Roboterposition repräsentieren zu können.

Zuletzt noch eine Überlegung, die unsere omnidirektionale Kamera betrifft: Der isometrische Spiegel liefert im Idealfall ein unverzerrtes Abbild der Spielfeldlinien. Unser Verfahren zur Linienextraktion kann aber auch auf verzerrte Kamerabilder, wie sie zum Beispiel bei Verwendung eines hyperbolischen Spiegels entstehen, erweitert werden. Dazu müssen die Transitionen vor der Linienextraktion entzerrt werden. Allerdings liegen die Linienbeobachtungen dann nicht mehr mit einem konstanten Fehler vor, wie in Abschnitt 2.3.1 beschrieben. Besonders das LineMatch-Verfahren geht aber von dieser Voraussetzung aus, da alle Linien unabhängig von ihrer Entfernung gleich behandelt werden. Beim Einsatz eines nicht-isometrischen Spiegels müssen die Linien unter Umständen nicht nur in Abhängigkeit ihrer Länge gewichtet werden, wie in Abschnitt 4.1.3 geschehen, sondern auch abhängig von ihrer Entfernung zum Roboter. Mit diesen Erweiterungen ist das LineMatch-Verfahren allgemein genug, um mit anderen omnidirektionalen Kameras eingesetzt zu werden.

Anhang A

Softwarearchitektur

In diesem Kapitel wollen wir einen kurzen Überblick des Softwaresystems geben, das im Rahmen dieser Diplomarbeit entworfen wurde. Zur Implementierung wurde die Programmiersprache C++ verwendet. Ausgehend von dem bestehenden Bildverarbeitungsmodul des CS Freiburg wurde ein eigenständiges Modul namens *Omnivision* entwickelt.

A.1 Omnivision

Das Klassendiagramm in Abbildung A.1 zeigt den Aufbau des Omnivision-Moduls. Als zentrales Element ist der *OmnivisionServer* zu betrachten. In ihm wird ein Thread gestartet, der den Ablauf der Mustererkennung steuert. Darüber hinaus stellt der *OmnivisionServer* das Interface zu allen wichtigen Funktionen der darunter liegenden Klassen dar.
Das *OmnivisionWindow* bietet eine graphische Oberfläche, die mit QT programmiert wurde. Mit ihrer Hilfe lassen sich die Farbklassen und Kameraparameter einstellen und die wichtigsten Funktionen des Moduls testen.
Die folgende Aufzählung fasst die Aufgaben der einzelnen Klassen zusammen:

- *Frame*: Datenstruktur eines Kamerabildes mit Größe und Zeitstempel. Außerdem stellt die Klasse Methoden zur Konvertierung in andere Farbdarstellungen zur Verfügung.

- *Capture*: Schnittstelle zum Treiber der Kamera. Liefert das rechteckige Bild, das die Kamera aufnimmt. Teile dieses Bildes sind überflüssig.

- *OmniCam*: Verwaltung des aktuellen Kamerabildes durch Lock- und Unlock-Operationen. Um die darauffolgenden Berechnungen zu beschleunigen, wird dabei der relevante quadratische Bereich in der Bildmitte ausgeschnitten und als Frame gespeichert. In dieser Form wird er zur Weiterverarbeitung zur

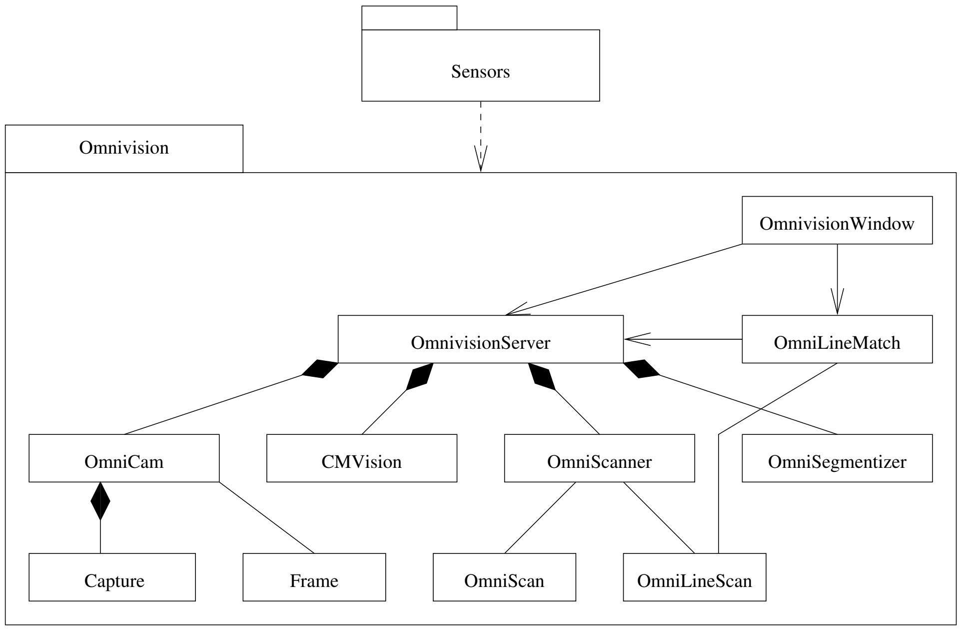


Abbildung A.1: *Klassendiagramm des Omnivision-Moduls*

Verfügung gestellt. Darüber hinaus speichert diese Klasse die Parameter, die die Kamera betreffen; dazu gehören u. a. die verschiedenen Spiegelbereiche.

- *CMVision*: Kernstück der CMVision-Bibliothek [Bruce 2000]. Führt die Farbklassifizierung durch und fasst Pixel gleicher Farbe zu Blobs zusammen.

- *OmniScan*: Datenstruktur zum Speichern von Transitionen. Implementiert die Datenfilter (siehe Seite 37) und andere nützliche Operationen.

- *OmniLineScan*: Datenstruktur für eine Menge von Linien.

- *OmniScanner*: Algorithmen zur Detektion der Spielfeldlinien in einem farbklassifizierten Frame.

- *OmniSegmentizer*: Methoden zum Erkennen der Eckpfosten und Tore in den Blobs des Kamerabildes.

- *OmnivisionServer*: Thread zur Steuerung der Merkmalsextraktion. Die aktuellen Beobachtungen der Kamera werden anderen Modulen zur Verfügung gestellt, zum Beispiel der Markov-Lokalisierung. Damit bildet der OmnivisionServer das Interface des Omnivision-Moduls.

- *OmniLineMatch*: Ausführung des LineMatch-Algorithmus mit den aktuellen Linien-Beobachtungen des OmnivisionServers. Liefert auf Anfrage eine Menge von Positionshypothesen oder die plausibelste Hypothese. Im Modul

Sensors wird mit diesen Informationen die LineMatch-Lokalisierung ausgeführt.

- *OmnivisionWindow*: Graphische Benutzerschnittstelle (GUI) zum Einstellen der Farbklassen und Kameraparameter. Außerdem dient diese Klasse zur Visualisierung und zum Test der Merkmalsextraktion.

A.2 Merkmalsextraktion

Wie bereits erwähnt, verbirgt sich im OmnivisionServer ein Thread, der gewährleistet, dass für andere Module stets die aktuellsten Beobachtungen zur Verfügung stehen. Dieser Thread ruft ständig die Methode *update()* des OmnivisionServers auf, wodurch der Vorgang der Merkmalsextraktion angestoßen wird. Das UML-Diagramm in Abbildung A.2 gibt einen Überblick über die Reihenfolge der Schritte, die daraufhin ausgeführt werden und die Objekte, die daran beteiligt sind.

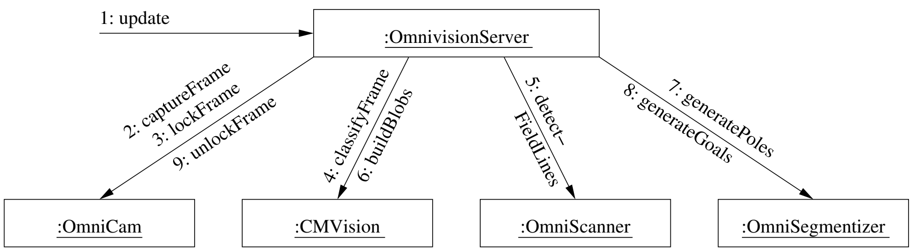


Abbildung A.2: *Kollaborationsdiagramm zur Merkmalsextraktion*

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Asada und Kitano 1999] ASADA, Minoru (Hrsg.) ; KITANO, Hiroaki (Hrsg.): *Lecture Notes in Computer Science*. Bd. 1604: *RoboCup-98: Robot Soccer World Cup II*. Springer, 1999

[Asimov 1952] ASIMOV, Isaac: *Ich, der Robot*. Heyne, 1952

[Birk et al. 2002] BIRK, Andreas (Hrsg.) ; CORADESCHI, Silvia (Hrsg.) ; TADOKORO, Satoshi (Hrsg.): *Lecture Notes in Computer Science*. Bd. 2377: *RoboCup 2001: Robot Soccer World Cup V*. Springer, 2002

[Bredenfeld et al. 2002] BREDENFELD, A. ; BECAMOVIC, V. ; CHRISTALLER, Th. ; GÜNTHER, H. ; INDIVERI, G. ; KOBIALKA, H.-U. ; PLÖGER, P.-G. ; SCHÖLL, P.: GMD-Robots. In: *Lecture Notes in Computer Science 2377*. 2002, S. 648–652

[Bronstein et al. 1999] BRONSTEIN, I. N. ; SEMENDJAJEW, K. A. ; MUSIOL, G. ; MÜHLIG, H.: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 1999

[Bruce 2000] BRUCE, James. *Realtime machine vision perception and prediction*. Carnegie Mellon University, Undergraduate Thesis. 2000

[Burkhard et al. 2002] BURKHARD, H.-D. ; DUHAUT, D. ; FUJITA, M. ; LIMA, P. ; MURPHY, R. ; ROJAS, R.: The Road to RoboCup 2050. In: *IEEE Robotics & Automation Magazine* (2002), S. 31–38

[Burkhardt 1999] BURKHARDT, Prof. Dr. H.: *Bildverarbeitungspraktikum I: Praktikumsanleitung*. Lehrstuhl für Mustererkennung und Bildverarbeitung, Albert-Ludwigs-Univerität Freiburg, 1999

[CS Freiburg 2001] CS FREIBURG. *The Robotic Soccer Team*. www.cs-freiburg.de. 2001

[Daniilidis 2002] DANIILIDIS, Kostas. *The Page of Omnidirectional Vision*. www.cis.upenn.edu/~kostas/omni.html. 2002

[Enderle et al. 2000] ENDERLE, S. ; RITTER, M. ; FOX, D. ; SABLATNÖG, S. ; KRAETZSCHMAR, G. ; PALM, G.: Soccer-Robot Localization Using Sporadic

Visual Features. In: *Intelligent Autonomous Systems 6 (IAS-6)*. Amsterdam, The Netherlands : IOS Press, 2000, S. 959–966

[Fischer 2002] FISCHER, Björn: *Robuste Positionsschätzung mittels Monte-Carlo-Lokalisierung in der RoboCup-Umgebung*, Albert-Ludwigs-Universität Freiburg, Diplomarbeit, 2002

[Fox 1998] FOX, Dieter: *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*, University of Bonn, Germany, Diss., 1998

[Fox et al. 1999] FOX, Dieter ; BURGARD, Wolfram ; THRUN, Sebastian: Markov Localization for Mobile Robots in Dynamic Environments. In: *Journal of Artificial Intelligence Research* 11 (1999)

[Gutmann 2000] GUTMANN, Jens-Steffen: *Robuste Navigation autonomer mobiler Systeme*, Albert-Ludwigs-Universität Freiburg, Diss., 2000

[Hicks und Bajcsy 1999] HICKS, R. A. ; BAJCSY, R.: Reflective Surfaces as Computational Sensors. In: *IEEE Workshop on Perception for Mobile Agents - Proc. of the 1999 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 99), Piscataway, NJ*, 1999

[Jähne 1997] JÄHNE, Bernd: *Digitale Bildverarbeitung*. Springer, 1997

[Kitano et al. 1995] KITANO, H. ; ASADA, M. ; KUNIYOSHI, Y. ; NODA, I. ; OSAWA, E.: RoboCup: The Robot World Cup Initiative. In: *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*, 1995

[Kitano 1998] KITANO, Hiroaki (Hrsg.): *Lecture Notes in Computer Science*. Bd. 1395: *RoboCup-97: Robot Soccer World Cup I*. Springer, 1998

[Kleiner 2002] KLEINER, Alexander. *SimServer: A universal Simulator for the RoboCup F-2000 (mid-sized) league*. sourceforge.net/projects/simsrv/. 2002

[Lima et al. 2001] LIMA, P. ; BONARINI, A. ; MACHADO, C. ; MARCHESE, F. ; RIBEIRO, F. ; SORRENTI, D.: Omni-directional catadioptric vision for soccer robots. In: *Robotics and Autonomous Systems* 36 (2001), Nr. 2–3, S. 87–102

[Marchese und Sorrenti 2001] MARCHESE, Fabio M. ; SORRENTI, Domenico G.: Omni-directional Vision with a Multi-part Mirror. In: *Lecture Notes in Computer Science 2019*. 2001, S. 179–188

[Marques und Lima 2001] MARQUES, Carlos F. ; LIMA, Pedro U.: A Localization Method for a Soccer Robot Using a Vision-Based Omni-Directional Sensor. In: *Lecture Notes in Computer Science 2019*. 2001, S. 96–107

[Maybeck 1990] MAYBECK, Peter S.: The Kalman Filter: An Introduction to Concepts. In: COX, Ingemar J. (Hrsg.) ; WILFONG, Gordon T. (Hrsg.): *Autonomous Robot Vehicles*. Springer, 1990, S. 194–204

[Ottmann und Widmayer 1993] OTTMANN, Thomas ; WIDMAYER, Peter: *Algorithmen und Datenstrukturen*. BI-Wissenschafts-Verlag, 1993

[Price et al. 1998] PRICE, Andrew ; JENNINGS, Andrew ; KNEEN, John: RoboCup 97: An Omnidirectional Perspective. In: *Lecture Notes in Computer Science 1395*. 1998, S. 320–332

[RoboCup 2002] ROBOCUP. *Rules and Regulations*. www.robocup.org. 2002

[Schaeffer 1997] SCHAEFFER, Jonathan: Kasparov versus Deep Blue: The Rematch. In: *ICCA Journal* 20 (1997), Nr. 2, S. 95–102

[Shannon 1950] SHANNON, Claude E.: Programming a Computer for Playing Chess. In: *The Philosophical Magazine* 41, 1 (1950), S. 256–275

[Stone et al. 2001] STONE, Peter (Hrsg.) ; BALCH, Tucker R. (Hrsg.) ; KRAETZSCHMAR, Gerhard K. (Hrsg.): *Lecture Notes in Computer Science*. Bd. 2019: *RoboCup 2000: Robot Soccer World Cup IV*. Springer, 2001

[Suzuki et al. 1998] SUZUKI, S. ; TAKAHASHI, Y. ; UCHIBE, E. ; NAKAMURA, M. ; MISHIMA, C. ; ISHIZUKA, H. ; KATO, T. ; ASADA, M.: Vision-Based Robot Learning Towards RoboCup: Osaka University »Trackies«. In: *Lecture Notes in Computer Science 1395*. 1998, S. 305–319

[Svoboda 1999] SVOBODA, Tomáš: *Central Panoramic Cameras: Design, Geometry, Egomotion*, Czech Technical University, Prague, Diss., 1999

[Veloso et al. 2000] VELOSO, Manuela M. (Hrsg.) ; PAGELLO, Enrico (Hrsg.) ; KITANO, Hiroaki (Hrsg.): *Lecture Notes in Computer Science*. Bd. 1856: *RoboCup-99: Robot Soccer World Cup III*. Springer, 2000

[Weigel et al. 2002] WEIGEL, T. ; GUTMANN, J.-S. ; DIETL, M. ; KLEINER, A. ; NEBEL, B.: CS-Freiburg: Coordinating Robots for Successful Soccer Playing. In: *IEEE Transactions on Robotics and Automation* 18 (5) (2002), S. 685–699

[Weigel 1998] WEIGEL, Thilo: *Roboterfußball: Selbstlokalisierung, Weltmodellierung, Pfadplanung und verhaltensbasierte Kontrolle*, Albert-Ludwigs-Universität Freiburg, Diplomarbeit, 1998

[Welch und Bishop 2002] WELCH, Greg ; BISHOP, Gary. *An Introduction to the Kalman Filter*. www.cs.unc.edu/~welch/kalman/kalmanIntro.html. 2002

Erklärung

Name: Erik Schulenburg　　　　　　　　　　　Matrikelnummer: 9333546

Hiermit erkläre ich, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Freiburg, den 13. Februar 2003