



Technische Universität Berlin



Mittelkreiserkennung im RoboCup auf Basis von Liniensegmenterkennung

Bachelorarbeit

am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der
Telekommunikation (AOT)

Prof. Dr.-Ing. habil. Sahin Albayrak

Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

vorgelegt von

Rico Jasper

Betreuer: Dipl-Inform. Martin Berger,
Dipl.-Ing. Thebin Lee

Rico Jasper
Matrikelnummer: 319396
Landsberger Allee 208
10367 Berlin

Erklärung der Urheberschaft

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Ort, Datum

Unterschrift

Zusammenfassung

Mit dem RoboCup-Projekt soll die Forschung in den Bereichen der künstlichen Intelligenz und der Robotik vorangetrieben werden. In alljährlich stattfindenden Weltmeisterschaften treten Teams aus Roboter im Fußballspiel gegeneinander an. Durch den Austausch der Erkenntnisse unter den teilnehmenden Entwickler möchte man bis zum Jahr 2050 erreichen, mit einem Roboterteam einen Sieg gegen menschliche Spieler zu erringen.

Eine große Herausforderung stellt dabei die Orientierung der Roboter auf dem Spielfeld dar. Diese müssen anhand von visuellen Informationen ihre Umgebung erfassen um sich zurechtzufinden. Das Spielfeld setzt sich aus mehreren Komponenten zusammen, die vom Roboter erkannt werden müssen.

In dieser Arbeit wird ein Verfahren zur Erkennung des Mittelkreis des Fußballfelds. Da der Mittelkreis in den meisten Fällen als Ellipse abgebildet wird, ist die Ellipsendetektion von zentraler Bedeutung, dem ein großer Umfang dieser Arbeit gewidmet ist. Genau wie Ecken, Linien und Flächen zählen Ellipsen zu den elementaren Merkmalen von komplexen Objekten.

Die Basis dieses Verfahrens ist eine Liniensegmenterkennung, die vom Line Segment Detector [1] durchgeführt wird. Dieser sucht in einem Bild nach geradlinigen Konturen. Somit wird von den Pixeldaten abstrahiert um eine effizientere Verarbeitung zu ermöglichen. Die so gewonnen Strecken dienen als Datengrundlage für die Extraktion der Position des Mittelkreises.

Der Algorithmus wurde in MATLAB implementiert und dessen Wirksamkeit anhand einer umfangreichen Bilddatenbank der HTWK Leibzig demonstriert und bewertet. Damit steht ein Prototyp zur Verfügung, welcher als Ausgangspunkt für praktische Umsetzungen in der RoboCup-Umgebung dienen kann.

Inhaltsverzeichnis

Erklärung der Urheberschaft	II
Zusammenfassung	III
Inhaltsverzeichnis	IV
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Struktur der Arbeit	3
2 Grundlagen und Stand der Forschung	4
2.1 RoboCup-Umgebung	4
2.2 Line Segment Detector	5
2.3 Projektion	7
2.4 Parameterschätzung von Kegelschnitten	8
2.5 Ellipsendetektoren	10
3 Problemanalyse	11
3.1 Abstraktion von Konturen	11
3.2 Liniensegmente des Mittelkreises	12
3.3 Projektion des Mittelkreises	13
3.4 Erzeugen einer Ellipse	14
3.5 Abweichung zwischen Strecke und Ellipse	14
3.6 Verifikation des Mittelkreises	15
3.7 Zusammenfassung	16
4 Lösungskonzept	17
4.1 Feldfarbenerkennung	17

4.2	Streckenselektion	18
4.2.1	Strecken filtern	18
4.2.2	Kettenbildung	19
4.2.3	Ketten verbinden	21
4.2.4	Ausreißer entfernen	22
4.2.5	Ketten erweitern	24
4.3	Ellipsenmodell	27
4.3.1	Modellberechnung	27
4.3.2	Modellvalidierung und Unterstützungsmenge	28
4.3.3	Entfernungsmaß	28
4.3.4	Orientierungskriterium	32
4.4	Verifikation	34
5	Umsetzung	38
5.1	Line Segment Detector	39
5.2	Streckenselektion	40
5.2.1	sieve_segs	40
5.2.2	detect_chains	40
5.2.3	merge_chains	40
5.2.4	improve_chains	41
5.2.5	extend_chains	41
5.3	Ellipsenmodell	41
5.3.1	fitellipse_segs	41
5.3.2	validate_model	42
5.3.3	find_consensus	42
5.3.4	select_consensus	43
5.3.5	line_rating	44
5.3.6	filter_angle	45
5.4	Verifikation	45
5.4.1	verify_models	45
6	Evaluierung	47
6.1	Berechnungszeiten	48
6.2	Erkennungsraten	49
6.3	Problemfälle	50
6.3.1	Fehlerhafte Feldfarbenerkennung	50
6.3.2	Schlechte Annäherung durch Parameterschätzung	50

6.3.3	Falsche Vereinigung von Ketten	52
6.3.4	Ausreißer nicht erkannt	52
6.3.5	Hinzufügen falscher Strecken zu einer Kette	53
6.3.6	Falsch-negative Verifikation	54
6.4	Erkannte Mittelkreise	55
7	Fazit und Ausblick	57
	Literaturverzeichnis	59
	Abbildungsverzeichnis	61
	Tabellenverzeichnis	63
	Abkürzungsverzeichnis	64

Kapitel 1

Einleitung

1.1 Motivation

Das RoboCup-Projekt hat sich zum Ziel gesetzt bis zur Hälfte des 21. Jahrhunderts ein Team aus selbstständigen Robotern gegen menschliche Spieler in einem Fußballspiel antreten zu lassen und zu gewinnen. Bis dorthin ist es noch ein weiter Weg. Der Zweck dieser Herausforderung soll es sein, die Forschung in Robotik und künstliche Intelligenz voranzutreiben. [2]

Zur Erreichung dieses hochgesteckten Ziels sind zahlreiche Hürden zu überwinden. Eine ist das Zurechtfinden der Roboter in ihrer Umgebung. Das erfordert sowohl motorische Fähigkeiten als auch Koordination.

Eine Grundvoraussetzung zum Agieren in Umgebungen wie beim RoboCup, ist die Fähigkeit des Roboters andere Objekte zu erkennen und darauf aufbauend diese und sich selbst zu lokalisieren. Erst dann ist es möglich einen Ball zu erfassen, diesen zu Mitspielern zu passen, Gegnern auszuweichen und gezielt Tore zu schießen.

Im RoboCup haben sich verschiedene Ligen herausgebildet, die ihren Fokus auf verschiedene Problemfelder konzentrieren. Dazu zählen verschiedene Größenklassen, in denen sich die Roboter physisch auf dem Feld gegenüberstehen, als auch völlig virtuelle Umgebungen. Daneben existieren ebenfalls Ligen, die sich nicht mit dem Fußballspiel befassen, wie RoboCup Rescue oder @Home.

In der Standard Plattform League des RoboCups sind nur Roboter eines bestimmten Typs, der Nao Roboter von Aldebaran Robotics, zugelassen, die nicht modifiziert werden dürfen. Dadurch werden gleiche Wettkampfvoraussetzungen für die Teilnehmer gewährleistet. Außerdem liegt so die Softwareentwicklung im Fokus [3]. Den Robotern steht nur ein begrenzter Umfang an Instrumenten zur Wahrnehmung ihrer Umgebung zur Verfügung. Dazu gehören Kameras, Sonar und Infrarot. Die Roboter dürfen

sich nicht durch externe Instrumente orientieren, sondern müssen autonom agieren [4]. Wie menschliche Fußballspieler müssen sie sich daher vor allem durch visuelle Informationen orientieren. [5] [6]

Zur eigenen Lokalisation (globale Lokalisation) sind die Spielfeldmarkierungen und Tore von zentraler Bedeutung. Der Roboter muss in der Lage sein, diese zuverlässig zu identifizieren und akkurate Messungen durchzuführen. Das Spielfeld setzt sich aus mehreren Komponenten zusammen und nicht alle können aufgrund von Verdeckungen oder der Perspektive zur selben Zeit im Blickfeld liegen. Daher muss der Roboter auch Einzelkomponenten detektieren können. In dieser Arbeit möchte ich mich der Mittelkreiserkennung widmen.

Bei vielen Problemen der Bildverarbeitung müssen elementare Merkmale wie Linien oder Ellipsen erfasst werden. Besonders zur Erkennung von komplexen Objekten kann die Zerlegung in Linien und Ellipsenbögen sehr hilfreich sein [7] [8]. Wie auch beim RoboCup-Spielfeld werden Kreise durch Projektion im Allgemeinen als Ellipsen abgebildet. Daher ist zur Mittelkreiserkennung die Detektion von Ellipsen essentiell.

Die Ellipsendetektion hat ein großes Anwendungspotential und findet in vielen Bereichen Gebrauch. Beispiele dafür sind die Erkennung von Verkehrszeichen [9], Pupillenverfolgung [10] oder das Detektieren von Zellkernen [11]. Es besteht somit für die Mittelkreiserkennung die Möglichkeit, dass auch andere Forschungsbereiche davon profitieren können.

1.2 Zielsetzung

Wenn man einen Algorithmus zur Lokalisation für den RoboCup entwickelt, so muss man auf die Gegebenheiten der RoboCup-Umgebung achten. Trotz der Vorgaben für das Spielfeld, können Beleuchtung und die äußere Umgebung variieren. Die Roboter sehen nie das gesamte Spielfeld. Es gibt auch kein Teil des Spielfeldes, welches immer im Blick ist, insbesondere der Mittelkreis. Mit- und Gegenspieler können Teile von Markierungen verdecken. Dennoch müssen die Resultate robust und akkurat sein.

Da die Roboter in Echtzeit agieren müssen, gibt es besondere Anforderungen an den Berechnungsaufwand. Ebenso sollte mit den Speicherverbrauch sparsam umgegangen werden. Ein Bild auf Pixelebene zu analysieren ist aufgrund des Datenumfangs meist eine kostspielige Angelegenheit. Es kann daher lohnenswert sein, die Abstraktionsstufe des Bildes anzuheben. Anstatt die Bildinformationen durch Bildpunkte auszudrücken, können Strecken verwendet werden. Diese zeigen Grenzen von zusammenhängenden Farbflächen auf. Natürlich fordert auch dieser Abstraktionsschritt seinen Tribut an Re-

chenleistung. Dennoch kann dies eine sinnvolle Investition sein, da auch die Erkennung von anderen Spielfeldelementen davon profitieren kann.

Um den Umfang der Arbeit aber angemessen zu halten, wird zunächst nur eine Referenzimplementierung anvisiert. Das bedeutet, dass die Implementierung nicht zwingend die Kriterien für Speicher oder Echtzeit erfüllen muss, sondern dient nur als Referenz bzw. Prototyp für Umsetzungen, welche später tatsächlich auf der Zielplattform laufen.

In einer Evaluierung soll der entwickelte Algorithmus anhand einer Bilddatenbank der HTWK Leibzig getestet werden. Diese beinhaltet Bilder aus der RoboCup-Umgebung, wie Teile des Spielfeldes aber auch Außenbereiche. An Kriterien wie Berechnungsduer und Erkennungsrate soll die Leistung des Verfahrens bewertet werden.

1.3 Struktur der Arbeit

Zu Beginn möchte ich in Kapitel 2 einige Grundlagen behandeln. Dazu gehört eine kurze Darstellung der RoboCup-Umgebung (2.1) und der Line Segment Detector 2.2 (LSD), dessen Daten die Grundlage für das hier vorgestellte Verfahren geben.

Anschließend fahre ich mit der Problemanalyse in Kapitel 3 fort. Dort werden die Problempunkte identifiziert, die es im Folgekapitel 4 zu lösen gilt. In diesem Kapitel werden die theoretischen Grundlagen erarbeitet, die zur Umsetzung erforderlich sind. Dabei werden die Fragen geklärt, wie man eine Auswahl unter den Strecken des LSDs treffen kann und wie sich daraus ein Modell für den Mittelkreis ermitteln lässt.

In Kapitel 5 handele ich die Details über die Umsetzung des Lösungskonzepts in MATLAB ab. Der Inhalt reicht von einem Überblick über das Zusammenspiel zentraler Funktionen bis zur Bestimmung von Schwellwerten.

Schlussendlich werden die erbrachten Ergebnisse in der Evaluierung (6) bewertet und im Abschlusskapitel 7 kritisch beurteilt.

Kapitel 2

Grundlagen und Stand der Forschung

2.1 RoboCup-Umgebung

In der Standard-Platform-League dient als Spielfeld ein 10.4 m mal 7.4 m großer Teppich (Stand 2013 [5], siehe Abbildung 2.1). Dessen Farbe ist ein ausreichend helles Grün, welches ansonsten jedoch nicht weiter spezifiziert ist. Die Torpfosten und die Torlatte sind gelbe Zylinder mit 10 cm Durchmesser. Das Tornetz hat eine Höhe von 80 cm und kann weiß, grau oder schwarz sein. Die Spielfeldlinien haben eine weiße Farbe und markieren unter anderem den Spielbereich sowie den Strafraum und trennen die beiden Feldhälften. Der Mittelkreis ist der einzige Kreisbogen auf dem Feld.

Der Aufbau des Spielfelds kann sich jährlich ändern. Zum Beispiel hatte es im Jahr 2012 noch eine Größe von 7.4 m mal 5.4 m [12]. Dies dient der schrittweisen Annäherung zum realen Fußballspiel.

Die Beleuchtung kann sich je nach Austragungsort unterscheiden. Als künstliche Lichtquellen sind jedoch lediglich Deckenlampen erlaubt. Das heißt, Standscheinwerfer sind beispielsweise als Lichtquelle ausgeschlossen. Ansonsten sind die Lichtverhältnisse, wie die Leuchtstärke, nicht genauer bestimmt. Die Roboter müssen also mit variablen Bedingungen zureckkommen können. Zudem können Schatten von Personen oder Robotern störend wirken.

Einzelne Spielfelder können dicht beieinander liegen, auch ohne Sichtschutz. Die Roboter müssen deshalb in der Lage sein, ihr eigenes Feld von fremden unterscheiden zu können. Sollten die Felder jedoch dichter als drei Meter nebeneinander aufgebaut sein, so können dennoch Wände platziert werden. Die Beschaffenheit des Außenbereiches ist vom Regelwerk nicht weiter definiert. In Turnieren befinden sich hier meist die Zuschauer sowie Werbeflächen. Dieser Bereich sollte möglichst von der visuellen Verarbeitung ausgeblendet werden.

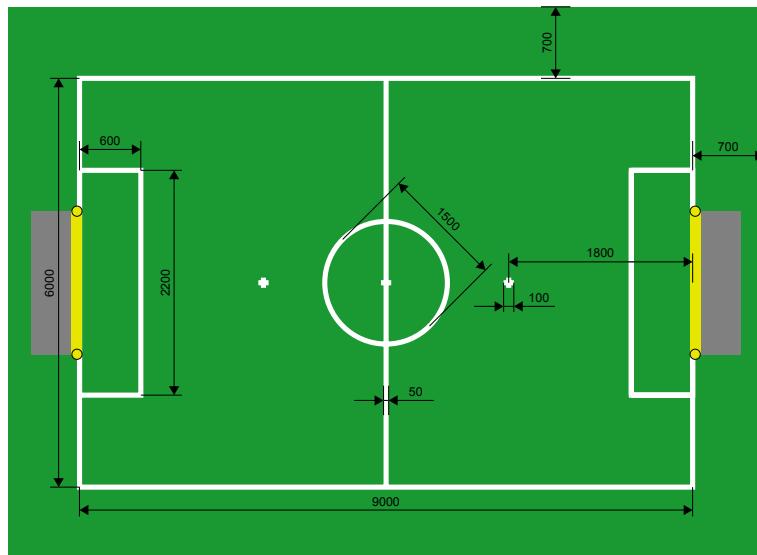


Abbildung 2.1: Spielfeld der Standard-Plattform-League (Maße in mm, Quelle: [5])

Während eines Spiels spielen zwei Teams mit jeweils bis zu fünf Nao-Robotern gegeneinander. Daher können Spielfeldelemente, wie Markierungen, Tore oder Ball häufig verdeckt werden.

2.2 Line Segment Detector

Der im Rahmen dieser Arbeit entwickelte Algorithmus zur Mittelkreiserkennung basiert auf der Auswertung von Liniensegmenten. Dabei handelt es sich um orientierte Strecken, mit einem Anfangs- und Endpunkt, welche die Konturen innerhalb eines Bildes aufzeigen. Erzeugt werden diese vom Line Segment Detector, vorgestellt in [1]. Dieser sucht in einem Graustufenbild auf Pixelebene nach Übergängen, die die Kontur einer Fläche gleicher Helligkeit ausmachen (siehe Abbildung 2.2).

Ein Übergang kann durch die Gradienten zwischen benachbarten Pixeln beschrieben werden. Der Winkel des Gradienten gibt dabei die Richtung an, in welcher der Übergang erfolgt und der Betrag die Stärke (siehe Abbildung 2.3). Die Berechnung der Gradienten zwischen allen Bildpunkten ergibt ein Feld aus Vektoren. Betrachtet man einen Punkt auf einer Kontur, so besitzen die Gradienten dieser Kontur in der näheren Umgebung auch ähnliche Beträge und Richtungen. Dies macht sich der LSD zu Nutze. Er sucht nach zusammenhängenden Gebieten, die ähnliche Gradientenwinkel besitzen. Dieser Vorgang nennt sich Region Growing. Dies kann man sich auch als Einfärben vorstellen, bei denen alle Gebiete unterschiedliche Farben erhalten (siehe Abbildung 2.4).

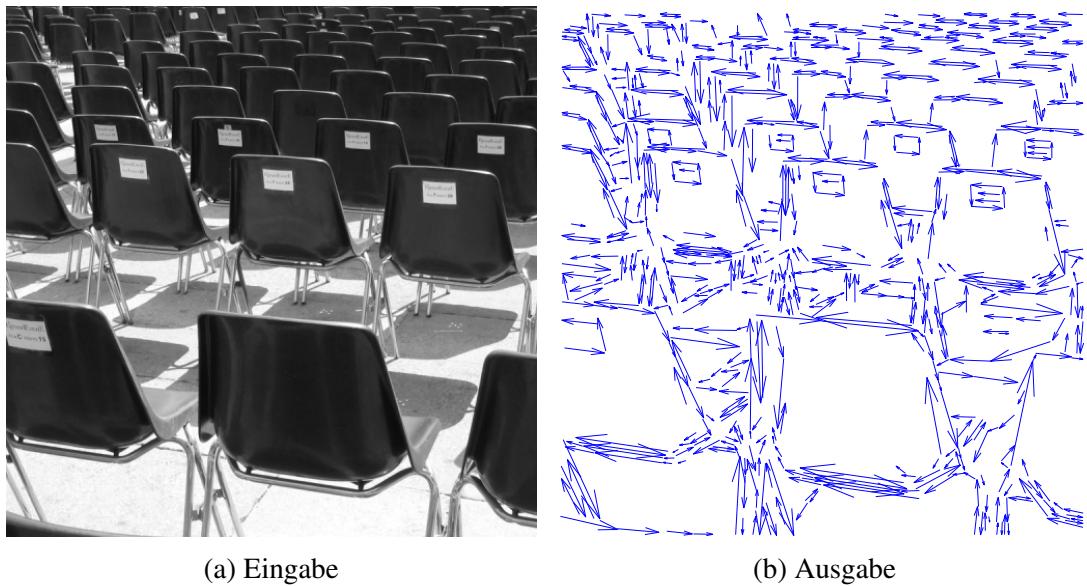


Abbildung 2.2: Resultat des LSDs (Quelle: [8])

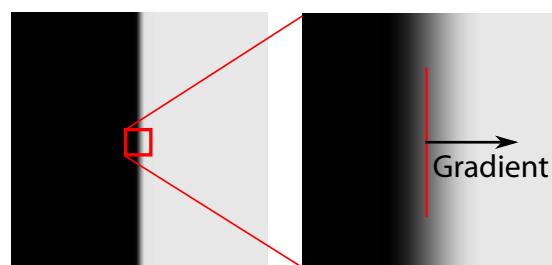


Abbildung 2.3: Gradient einer Kontur (Quelle: [8])

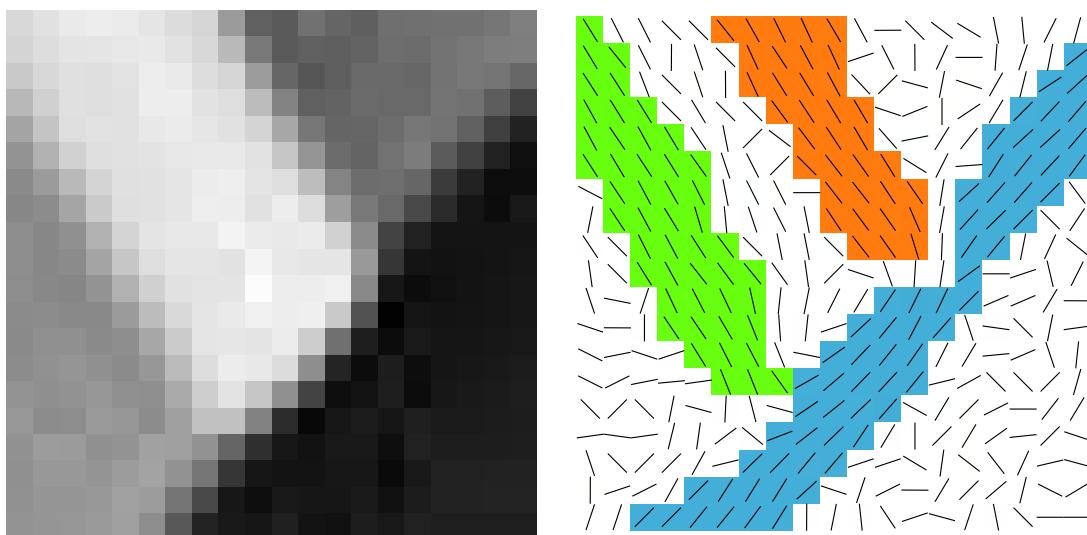


Abbildung 2.4: Region Growing (Quelle: [8])

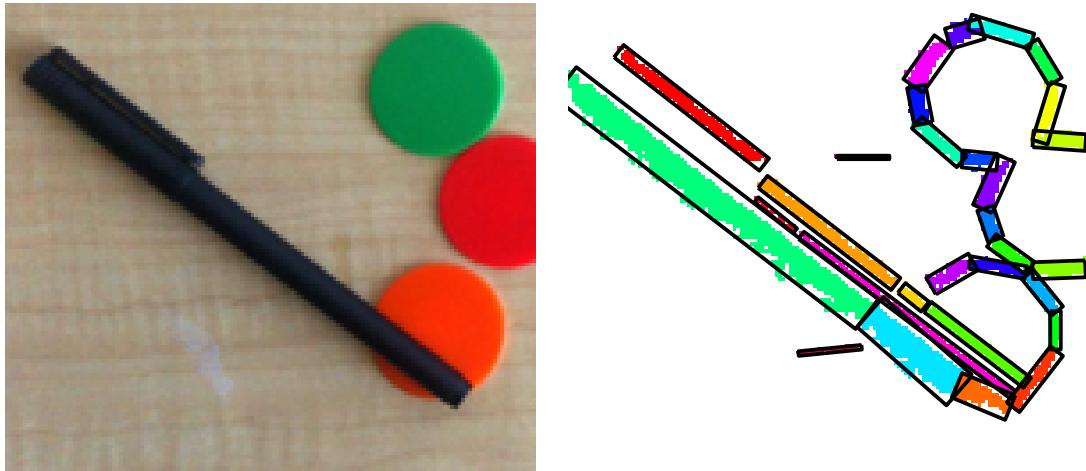


Abbildung 2.5: Rechtecke der eingefärbten Regionen (Quelle: MATLAB-Beispielbild *coloredChips.png*)

Jedes eingefärbte Gebiet wird anschließend von einem möglichst kleinen Rechteck eingegrenzt (siehe Abbildung 2.5). Die Rechtecke werden noch weiter verfeinert, so dass sie die Konturen möglichst genau annähern. Die Strecken, die sich durch die Mitte der Rechtecke entlang ihrer Orientierung ziehen, sind das Ergebnis der Liniensegmenterkennung. Darum besitzen die Strecken auch eine Dicke, sodass die Beschreibung als Strecke oder als Rechteck äquivalent sind. Start- und Endpunkt werden durch die Richtung des Gradienten definiert. Betrachtet man eine Strecke von ihrem Anfang zum Ende, dann befindet sich auf ihrer rechten Seite die dunkle Fläche und auf der linken die helle.

2.3 Projektion

Da die Roboter der Standard Plattform League autonom agieren müssen und nicht auf externe Sensoren zurückgreifen können, müssen sie ihre Umgebung selbst wahrnehmen und verarbeiten. Ihre Eindrücke sind deshalb nur lokal und daher abhängig von ihrer Position und Pose. Damit unterscheidet sich diese Liga von der Small Size League, in der das Spielfeld von einer oder mehreren externen Kameras aufgezeichnet und die Verarbeitung von einem zentralen Rechner vorgenommen wird.

Durch die Verwendung von Kameras wird die dreidimensionale Umgebung nur in zwei Dimensionen erfasst. Die Reduzierung von drei auf zwei Dimensionen kann mathematisch durch eine 3D-Projektion beschrieben werden. Dies ist eine Zuordnung von 3D- in 2D-Koordinaten. Solch eine Transformation lässt sich durch die Angabe eines Brennpunkts und einer Projektionsebene innerhalb des dreidimensionalen Raums dar-

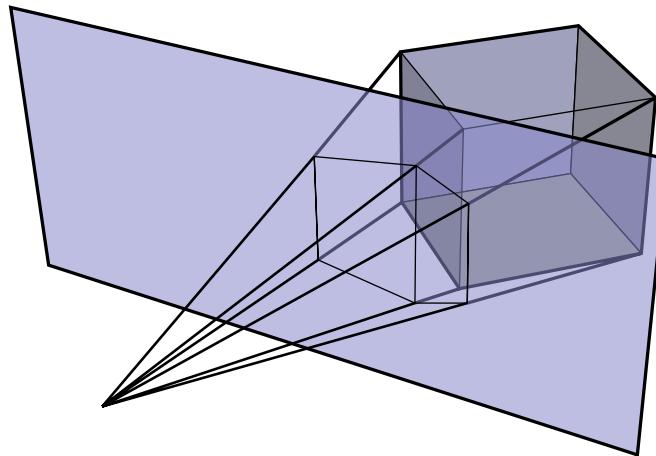


Abbildung 2.6: Projektion eines Würfels (Quelle: Wikipedia¹)

stellen (siehe Abbildung 2.6). Die Projektionsebene repräsentiert dabei das zweidimensionale Koordinatensystem. Verbindet man einen 3D-Punkt mit dem Brennpunkt zu einer Geraden, so ist der Schnittpunkt dieser Geraden mit der Projektionsebene der dazugehörige 2D-Punkt.

2.4 Parameterschätzung von Kegelschnitten

Die folgenden Betrachtungen sind der Arbeit von Gander, Golub und Strebel aus [13] entnommen. Die Ellipse gehört wie die Parabel und Hyperbel zu den Kegelschnitten. Dies sind Schnittpunkte einer Ebene mit einem senkrechten Doppelkegel (siehe Abbildung 2.7). Alle Schnittmengen können durch die Kegelschnittgleichung

$$F(\vec{x}) = \vec{x}^T A \vec{x} + \vec{b}^T \vec{x} + c = 0 \quad (2.1)$$

mit $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$, $\vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ und $\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ beschrieben werden.

Mit der Methode der kleinsten Fehlerquadrate kann man für eine Messpunktreihe $\vec{P}_i = (x_{i1}, x_{i2})^T$ mit $i \in \{1, \dots, m\}$ die Parameter A , \vec{b} und c schätzen. Dabei werden die

¹<http://en.wikipedia.org/wiki/File:Perspectiva-2.svg>

²http://en.wikipedia.org/wiki/File:Conic_sections_with_plane.svg

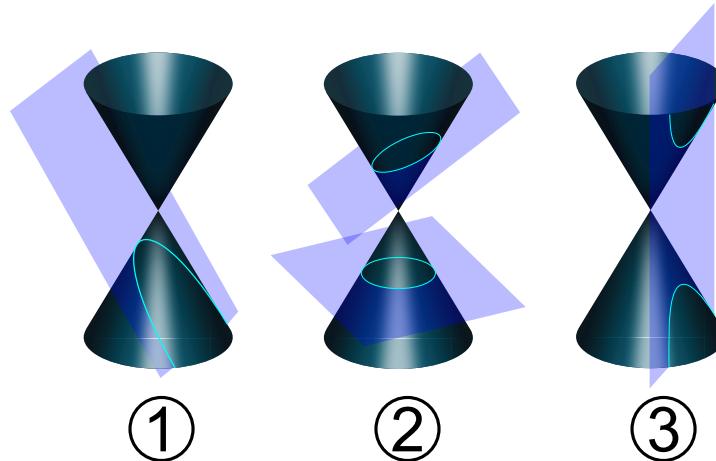


Abbildung 2.7: Kegelschnitte: 1. Parabel, 2. Ellipse und Kreis, 3. Hyperbel (Quelle: Wikipedia²)

Messpunkte in eine Matrix B und die Parameter in einen Vektor \vec{u} geschrieben:

$$B = \begin{pmatrix} x_{11}^2 + x_{12}^2 & x_{11} & x_{12} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1}^2 + x_{m2}^2 & x_{m1} & x_{m2} & 1 \end{pmatrix}$$

$$\vec{u} = (a_{11} \quad 2a_{12} \quad a_{22} \quad b_1 \quad b_2 \quad c)^T$$

Mit $B\vec{u} = \vec{0}$ erhalten wir ein lineares Gleichungssystem, welches äquivalent zum folgenden ist:

$$\begin{aligned} F(\vec{P}_1) &= 0 \\ &\vdots \\ F(\vec{P}_m) &= 0 \end{aligned}$$

Um die triviale Lösung auszuschließen, stellen wir eine Nebenbedingung (bspw. $\|\vec{u}\| = 1$) auf. Im Allgemeinen hat das System für $m > 5$ keine Lösung, es sei denn alle Punkte befinden sich zufällig exakt auf dem Kegel und der schneidenden Ebene. Für den Normalfall möchten wir das überbestimmte System $B\vec{u} = \vec{r}$ für den Parametervektor \vec{u} lösen, welcher den Wert $\|\vec{r}\|$ minimiert.

Ein Nachteil dieser Methode ist, dass im geometrischen Sinne nicht so recht klar ist, was minimiert wird. Deshalb gibt es auch Ansätze, bei denen versucht wird, den

geometrischen Fehler zu minimieren. Im Falle eines Kreises wäre dies die Abweichung vom Radius. Dies führt jedoch zu nicht-linearen Gleichungssystemen.

Zudem muss beachtet werden, dass mit dieser Methode nach einem im oberen Sinne optimalen Kegelschnitt gesucht wird. Das Resultat ist deshalb keineswegs immer eine Ellipse, sondern könnte zum Beispiel eine Hyperbel sein.

2.5 Ellipsendetektoren

Die Erkennung von Ellipsen gehört zu den entscheidenden Problemen der Bildverarbeitung. Sie treten in digitalen Bildern häufig auf und stellen wichtige Schlüsselmerkmale dar. Ihre Bedeutung führte zur Entwicklung vieler Algorithmen, die sich grob in drei Kategorien einordnen lassen [14]:

1. Algebraische und geometrische Ausgleichsrechnungen nach der Methode der kleinsten Fehlerquadrate.
2. Clustering- oder Voting-Methoden basierend auf der Hough-Transformation [15].
3. Statistische oder heuristische, kombinierte und sonstige Techniken.

Methoden, die auf einer Parameterschätzung beruhen, werden häufig verwendet, da sie meist einfach (nicht-iterativ) zu berechnen sind. Darum sind sie besonders für Echtzeitanwendungen geeignet. Allerdings können diese Verfahren empfindlich auf Ausreißer reagieren, produzieren in manchen Fällen unerwünschte Formen (z.B. Hyperbeln) und führen bei Verdeckungen nur zu durchschnittlichen Ergebnissen.

Die zweite Gruppe basiert auf der Hough-Transformation. Dabei wird zunächst ein Konturbild aus dem Eingabebild erzeugt. Dieses wird dann in ein Parameterraum transformiert, indem alle Konturpunkte für die Parameter jener Ellipsen stimmen, auf denen sie liegen könnten. Da Ellipsen fünf Parameter besitzen, führt dies zu einem fünfdimensionalen Raum, wodurch das Stimmen und Auszählen viel Zeit in Anspruch nehmen kann. Aufgrund der Robustheit gegenüber Verdeckung wird dieser Ansatz dennoch häufig genutzt. Zur Verringerung des Berechnungsaufwands wird die Methode modifiziert, um die Anzahl der Dimensionen zu reduzieren.

Einer dieser Modifikationen stammt von Xie und Ji [16], die den Parameterraum auf eine Dimension reduziert haben. Für jedes Paar von Konturpixeln nehmen sie dabei an, dass es sich um die zwei Scheitelpunkte der Hauptachse handelt, wodurch sich bereits vier der fünf Parameter bestimmen lassen. Allein die Länge der Nebenachse muss nun noch durch Abstimmen ermittelt werden. Dies funktioniert jedoch nur, wenn die Scheitelpunkte auch sichtbar sind.

Kapitel 3

Problemanalyse

3.1 Abstraktion von Konturen

Betrachtet man ein Bild aus der Perspektive des Roboters, so tragen vor allem die Konturen der sichtbaren Objekte wertvolle Informationen. Eine Kontur ist eine Grenzlinie zwischen zwei Farbflächen. Diese begrenzen das Spielfeld und die Tore, zeigen die Position der anderen Roboter und dem Ball sowie die Verläufe der Linien des Feldes. Der grüne Teppich inklusive der weißen Markierungen enthalten sehr klare und deutliche Konturen. Allerdings enthalten andere Bereiche viele Konturen, die uninteressant sind, wie den Außenbereich oder das Tornetz.

Die Extraktion von Konturen bzw. Kanten ist aufgrund ihres Informationsgehalts von hoher Bedeutung. Sie ermöglichen die Abstraktion und somit die effiziente Weiterverarbeitung des visuellen Inputs. Die Repräsentation von Konturen kann unterschiedlich sein. Eine einfache Möglichkeit ist die Angabe von Pixeln, die einer Kante angehören. Das Resultat einer Extraktion ist wie beim Canny Edge Detector ein binäres Bild (siehe Abbildung 3.1b), welches die Konturen des Originalbildes kennzeichnet. Man kann mit der Abstraktion jedoch auch noch einen Schritt weiter gehen, indem man Abschnitte von Konturen zusammenfasst und in einer höheren Form repräsentiert. Gerade Kanten können zum Beispiel durch Linien dargestellt werden und Rundungen durch Ellipsen (siehe Abbildung 3.1c).

Allerdings stellt sich hier die Frage, inwieweit eine Kontur überhaupt durch Liniensegmente angenähert werden kann. Im Falle von runden Konturen ist solch eine Annäherung auch immer fehlerbehaftet. Die Ränder des Mittelkreises können deswegen nicht exakt durch Linien beschrieben werden. Ein anderes Problem ist, dass sich Konturen unterschiedlich deutlich abzeichnen. Durch Bewegungs- oder Tiefunschärfe können die Grenzen von Objekten verwischen.

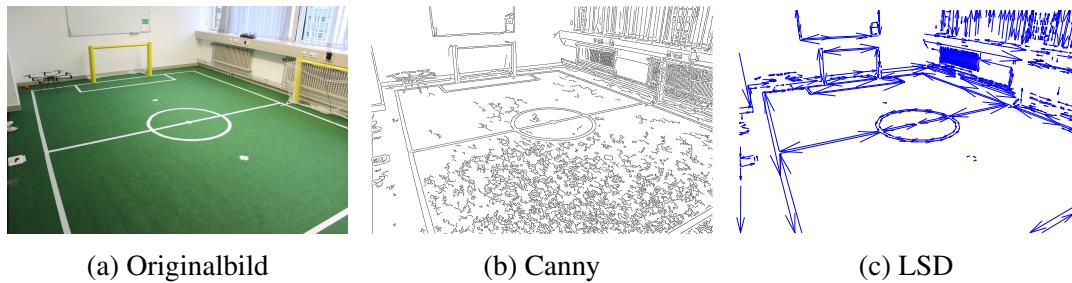


Abbildung 3.1: Konturextraktion



Abbildung 3.2: Grüntöne des Spielfeldes

3.2 Liniensegmente des Mittelkreises

Extrahiert man aus dem Bild des Spielfeldes Konturen in Form von Liniensegmenten, so steht man vor der Aufgabe, die relevanten Segmente zu finden. Für die Mittelkreiserkennung heißt das, die Strecken zu suchen, die zum Mittelkreis gehören. Auf dem Spielfeld generieren neben dem Mittelkreis die übrigen Markierungen, Tore, Roboter und der Ball ebenso Linien. Außerhalb des Feldes existieren zudem meist eine Vielzahl nicht definierter Objekte, die Konturen produzieren. Es gilt also bereits frühzeitig zu filtern, sodass möglichst viele Segmente vor einer weiteren Verarbeitung eliminiert werden.

Ein mögliches Kriterium sind die Farben, die von der Kontur getrennt werden. Relevant für den Mittelkreis sind nur Konturen, welche weiße und grüne Flächen voneinander trennen. Verwendet man den LSD zur Erkennung der Liniensegmente, so steht man allerdings vor dem Problem, dass dieser nur mit Graustufen arbeitet. Außerdem ist das Grün des Spielfeld nicht genau definiert. In Abbildung 3.2 ist zu sehen, welche Grüntöne in Turnieren verwendet werden können. Aufgrund von Reflexion, Bewegungsunschärfe und Kalibrierungsfehler können zudem die weißen Markierungen einen grünen Stich erhalten (siehe Abbildung 3.3).

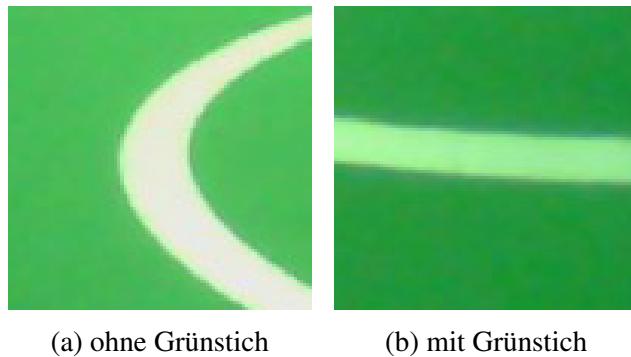


Abbildung 3.3: Linienfarbe

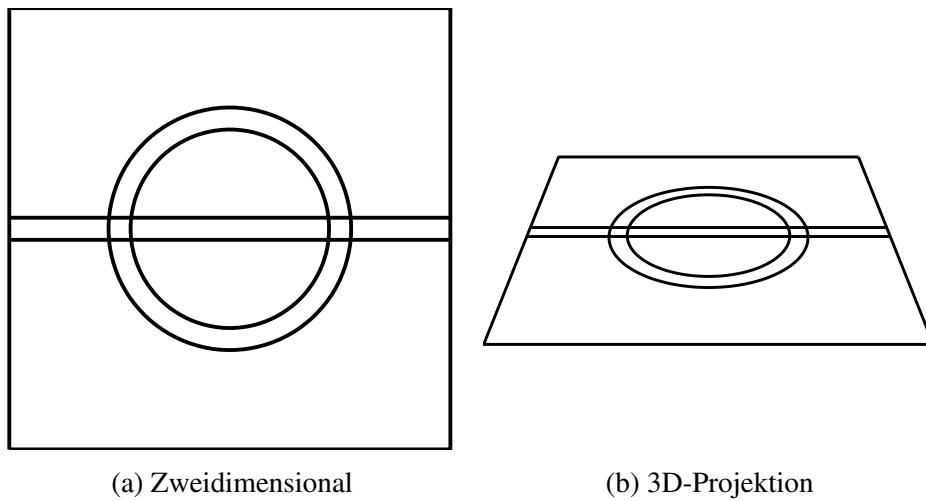


Abbildung 3.4: Projektion des Mittelkreises

3.3 Projektion des Mittelkreises

Durch 3D-Projektion erscheint der Mittelkreis meist als Ellipse (siehe Abbildung 3.4). Die Lage des Kreises im Raum wird in ein zweidimensionales Bild transformiert. Die resultierende Abbildung lässt sich auch als eine Schnittmenge einer Ebene und eines schiefen Kreiskegels auffassen. Die Grundfläche des Kreiskegels ist dabei der Mittelkreis und die Spitze der Brennpunkt der Kamera. Unter Vernachlässigung der Linsenoptik entspricht der CCD-Chip der schneidenden Ebene.

Ähnlich wie bei senkrechten Kegeln, entspringt dem Schnitt abhängig von der Lage der Ebene zum Kegeln nicht immer eine Ellipse. Für den senkrechten Kegel kann man drei Fälle unterscheiden, die den Pendants des schiefen Kegels ähneln. Verschiebt man die Projektionsebene so, dass sie den Brennpunkt enthält ohne die Orientierung zu verändern, dann folgt:

1. Die verschobene Ebene schneidet die Grundfläche nicht.

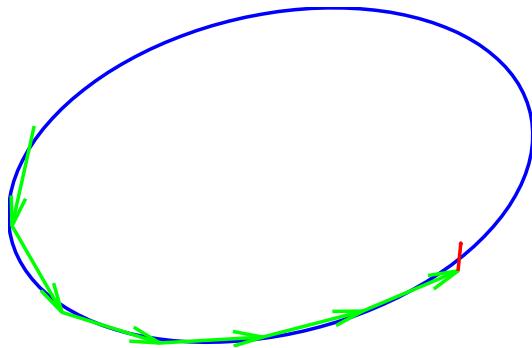


Abbildung 3.5: Die Orientierung kurzer Strecken (rot) wird kaum berücksichtigt.

2. Die verschobene Ebene tangiert die Grundfläche in einem Punkt.
3. Die verschobene Ebene schneidet die Grundfläche in zwei Punkten.

Im ersten Fall resultiert eine Ellipse, im zweiten eine Parabel und im letzten eine Hyperbel.

3.4 Erzeugen einer Ellipse

Die Aufgabe der Mittelkreiserkennung ist, den Mittelkreis zu lokalisieren. Das Resultat des Verfahrens ist eine Ellipse, die die Lage des Mittelkreises im Bild angibt. Da der Algorithmus allerdings mit geraden Liniensegmenten der Konturen arbeitet, müssen diese in eine Ellipse umgerechnet werden.

Wie man aus einer Reihe von Messpunkten die Parameter einer passenden Ellipse berechnen kann, wurde bereits in 2.4 beschrieben. Diese Methode kann auch für Strecken verwendet werden, indem man ihnen einige Punkte entnimmt. Allerdings wird so die Orientierung der Strecke von der Parameterschätzung nicht berücksichtigt. Dies kann besonders bei kurzen Strecken dazu führen, dass diese dann einen falschen Winkel zur berechneten Ellipse besitzen (siehe Abbildung 3.5).

3.5 Abweichung zwischen Strecke und Ellipse

Um die Parameter einer Ellipse anhand einer Menge von Punkten zu ermitteln, versucht man den Fehler der Punkte zur Ellipse zu minimieren. Die Definition dieses Fehlers kann unterschiedlich sein. Beispiele sind der algebraische oder der geometrische Fehler. Je geringer der Fehler eines Punktes ausfällt, desto eher passt er zur Ellipse. Er ist also ein Maß dafür, wie sehr der Punkt das Modell der Ellipse unterstützt. Ein solches Fehlermaß gilt es auch für Strecken zu definieren.

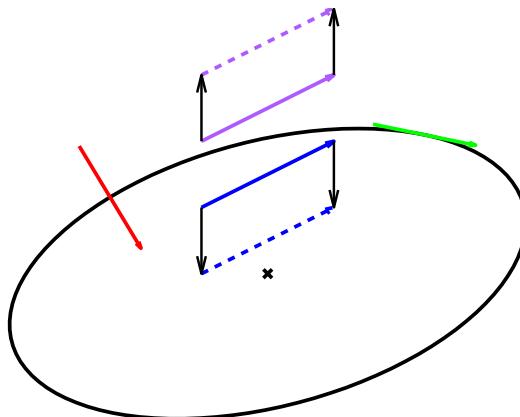


Abbildung 3.6: Entfernung und Orientierung einer Linie zur Ellipse (rot: Normale, grün: Tangente)

Anhand dieses Maßes können beispielsweise Ausreißer oder Unterstützer des Modells ermittelt werden. So, wie es auch bei Punkten möglich ist, könnte das Maß eine Art Entfernung der Strecke zur Ellipse angeben. Intuitiv sollte eine genaue Definition den folgenden Bedingungen genügen:

- Befindet sich die Linie außerhalb der Ellipse, so sollte die Entfernung steigen, wenn man sie in eine Richtung weg vom Zentrum bewegt. Violett in Abbildung 3.6.
- Befindet sie sich innerhalb der Ellipse, so sollte die Entfernung steigen, wenn man sie zum Zentrum bewegt. Blau in Abbildung 3.6.

Ein weiteres Kriterium ist die Orientierung der Strecken. Eine Strecke, die sich am Rand der Ellipse befindet, sollte beispielsweise nicht orthogonal zur Tangente ausgerichtet sein.

3.6 Verifikation des Mittelkreises

Der Innen- und Außenrand des Mittelkreises bilden nach der Projektion zwei Ellipsen. Nachdem diese zwei Ellipsen erkannt worden sind, sollte überprüft werden, ob sie zum Mittelkreis gehören bzw. ob sie ihn korrekt nachbilden (siehe Abbildung 3.7). Dies zu gewährleisten ist die Funktion der Verifikation.

Ein einfaches Kriterium zum Auffinden von Paaren ist, dass eine Ellipse die andere umschließen muss. Zwischen den Ellipsen sollte die weiße Linie des Mittelkreises zu sehen sein, während außerhalb das grüne Spielfeld liegt. Jedoch könnten Teile des Krei-

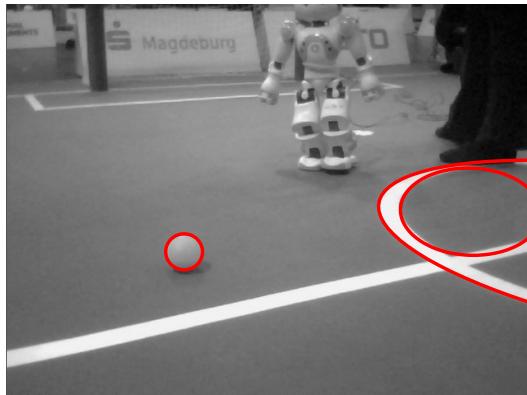


Abbildung 3.7: Irrelevante und verfälschte Ellipsen

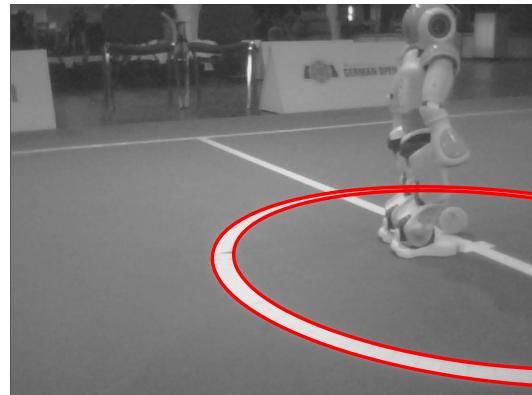


Abbildung 3.8: Verifizierung eines unvollständigen Mittelkreises

ses verdeckt sein. Trotzdem sollte die Verifikation in der Lage sein, jene Mittelkreise zu akzeptieren (siehe Abbildung 3.8).

3.7 Zusammenfassung

Aus den identifizierten Problempunkten lassen sich nun die Lösungsschritte ableiten. Vom RGB-Bild der Roboterkamera muss zunächst geeignet abstrahiert werden. Das Ergebnis sind Liniensegmente (Strecken). Aufgrund der Perspektive erscheint der Mittelkreis im Regelfall als Ellipse. Also müssen die korrekten Strecken ausgewählt werden, um daraus eine Ellipse zu berechnen. Weitere Strecken müssen zu dieser Ellipse in Beziehung gesetzt werden können, um eine Erweiterung der Auswahl zu ermöglichen. Am Ende steht die Verifikation, die gewährleisten soll, dass es sich bei einem Ellipsenpaar tatsächlich um den Mittelkreis handelt.

Kapitel 4

Lösungskonzept

Wie bereits erwähnt ist der Ausgangspunkt der Mittelkreiserkennung das Ergebnis des Line Segment Detectors. Im Großteil der folgenden Schritten wird mit Strecken gearbeitet.

Bei der Entwicklung des Lösungskonzepts stehen zwei Fragen im Vordergrund: Welche Strecken gehören zum Mittelkreis und wie lassen sie sich mit Ellipsen in Verbindung bringen? Die erste Frage soll im Abschnitt Streckenselektion geklärt werden. Der Abschnitt über das Ellipsenmodell beschäftigt sich mit der Beziehung zwischen Strecken und Ellipsen. Ein weiterer Punkt ist die Verifikation, in der sich die Frage stellt, wie man Modelle des Mittelkreis überprüfen kann. Abseits davon wird auch kurz ein Ansatz für die Farberkennung des Spielfelds skizziert, welche jedoch nicht im Fokus steht und lediglich der Vollständigkeit halber angesprochen wird.

4.1 Feldfarbenerkennung

Eine robuste und zuverlässige automatische Farberkennung zu entwickeln, welche in der Lage ist die zentralen Farben des Spielfeldes zu bestimmen ist keineswegs eine leichte Aufgabe. Dennoch ist sie für ein initiales „Aussieben“ der Strecken (siehe 4.2.1) sowie der Verifizierung (4.4) erforderlich. Um also den Umfang der Arbeit nicht zu sprengen, möchte ich hier nur einen einfachen Lösungsansatz skizzieren.

Bei diesem Ansatz wird das Bild vom RGB-Farbraum in den HSV-Raum transformiert. Dieser schlüsselt die Farben nach Farbwert (Hue), Sättigung (Saturation) und Hellwert (Value) auf. Es genügt bereits Grenzwerte für diese Kanäle festzulegen, um Entscheidungen darüber zu treffen, welche Pixel des Bildes zum Grün des Spielfeldes gehören.

Üblicherweise nimmt das Spielfeld einen großen Teil des Bildes ein. Mithilfe von Histogrammen lassen sich die Grenzwerte bestimmen, indem man die Werte mit dem höchsten Auftreten sucht und einen Bereich darum absteckt.

Ergebnis dieser Erkennung ist ein Binärbild, welches die Positionen für „grüne“ Pixel angibt. Es ist somit also eine Maske des Spielfeldes.

Da dieser Ansatz sehr einfach ist, genügt er nicht den Anforderungen der Robustheit und Zuverlässigkeit. Darum sollte dieser für eine ernsthafte Anwendung ersetzt werden. Zum Zweck dieser Arbeit ist er allerdings ausreichend, sorgt aber in manchen Fällen für Probleme.

4.2 Streckenselektion

In diesem Abschnitt geht es darum aus der Menge von Strecken, welche der LSD hervorgebracht hat, jene auszuwählen, aus denen sich Ellipsen bilden lassen. Diese Ellipsen werden später darauf geprüft, ob sie zu einem Mittelkreis gehören.

4.2.1 Strecken filtern

Da dem LSD nur Bilder in Graustufen übergeben werden, haben Farben keinen Einfluss auf die Erkennung von Konturen¹. Von Interesse sind allerdings nur die Konturen der weißen Spielfeldlinien auf grünen Grund. Alle übrigen Strecken können ignoriert werden.

Eine Strecke besitzt eine Orientierung, sodass sich auf ihrer rechten Seite stets die dunklere Fläche als links befindet. Demnach sollten nur Strecken zugelassen werden, auf deren rechten Seite sich die grüne Feldfarbe wiederfinden lässt und auf der linken die weiße Linienmarkierung. Durch Abtasten einiger Pixel kann man dies überprüfen.

Bei der Abtastung muss bedacht werden, dass die Strecken des LSD auch eine Dicke besitzen. Damit wird ein $l \times w$ großes Rechteck aufgespannt, welches einen orthogonalen Verlauf der angrenzenden Farben enthält. Es sollten die Farben der Pixel in einem Bereich außerhalb des Rechtecks überprüft werden.

Abbildung 4.1 zeigt das verwendete Muster der Lösung. Links und rechts vom aufgespannten Rechteck der Strecke $\overline{P_1 P_2}$ sind zwei weitere Rechtecke zu sehen, die in einem regelmäßigen Raster abgetastet werden. Die Länge dieser Rechtecke ist gleich

¹Tatsächlich werden Konturen zwischen manchen Farben nicht erkannt, da sie auf dieselben Graustufe abbilden.

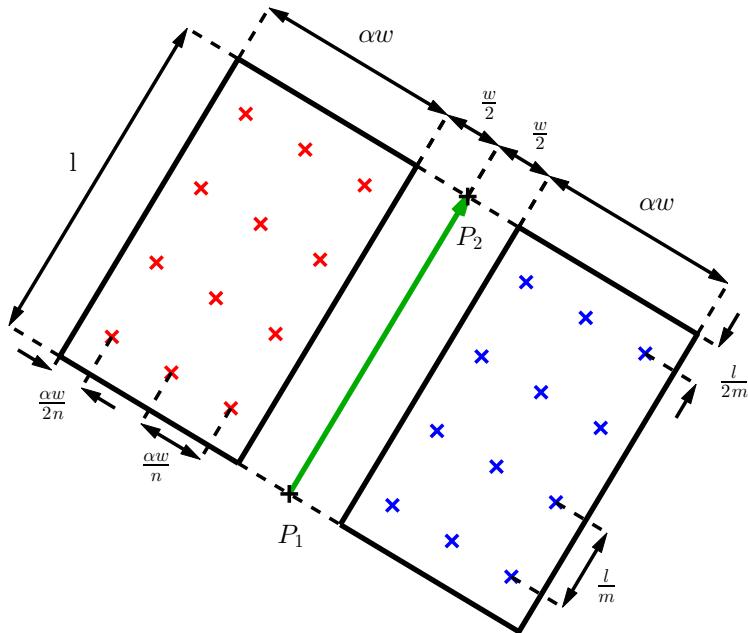


Abbildung 4.1: Abtastmuster zur Farbprüfung einer Kontur

die der Strecke. Die Breite ist um den Faktor α vergrößert. Das Raster enthält m orthogonale und n parallele Pixelreihen und ist in den äußereren Rechtecken zentriert.

Nun kann pixelweise die Prüfung auf die Farbe erfolgen. Zur Grünerkennung werden die Daten aus dem Schritt 4.1 verwendet. Da keine Weißenkennung implementiert wurde, beschränkt sich die Prüfung auf die rechte Seite. Um möglichst falsch-negative Tests zu vermeiden, muss lediglich ein Pixel grün sein.

4.2.2 Kettenbildung

Die zum Mittelkreis gehörigen Strecken treten häufig als Ketten auf (siehe Abbildung 4.2). Das bedeutet, ein Streckenende zeigt auf den Beginn der nächsten Strecke des Kreises. Zudem laufen sie stets nur in eine Richtung. Im Falle des äußeren Randes nach links, beim inneren nach rechts. Solche Ketten bieten einen guten Ausgangspunkt für eine weitere Verarbeitung, da der Mittelkreis in den meisten Fällen solche Ketten aufweist und die Menge der zu betrachtenden Strecken einschränkt.

Ketten werden in zwei Schritten gebildet. Im ersten wird ein gerichteter Graph aus den vorhandenen Strecken gebildet. Jede Strecke ist dabei ein Knoten. Eine Kante zeigt auf den Nachfolger einer Strecke. Eine Strecke gilt dann als Nachfolger, wenn ihr Startpunkt eine ausreichend geringe Distanz zum Endpunkt des Vorgängers besitzt. Außerdem darf der Winkel zwischen beiden Strecken nur in einem festgelegten Bereich liegen.

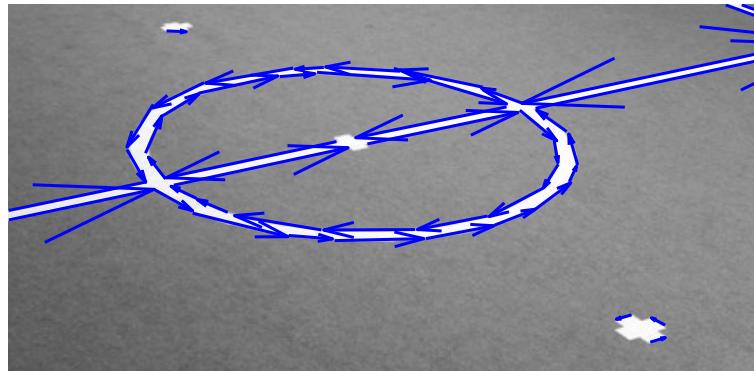


Abbildung 4.2: Ketten aus Strecken

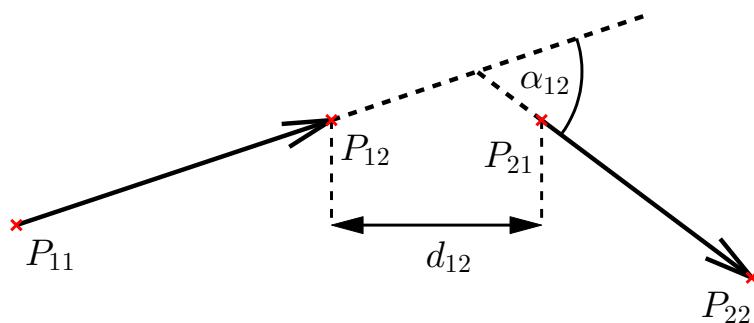


Abbildung 4.3: Abstand und Winkel zweier Strecken

Damit wird erreicht, dass benachbarte Strecken nicht zu weit entfernt liegen und dass im nächsten Schritt nur nach links bzw. rechts laufende Ketten erkannt werden.

Die Distanz d_{12} und der Winkel α_{12} von der Strecke $S_1 := \overline{P_{11}P_{12}}$ zu $S_2 := \overline{P_{21}P_{22}}$ sind dabei definiert als (siehe Abbildung 4.3):

$$d_{12} := \|P_{21} - P_{12}\|$$

$$\alpha_{12} := \alpha_1 - \alpha_2 + \begin{cases} +2\pi, & \alpha_1 - \alpha_2 \leq -\pi \\ -2\pi, & \alpha_1 - \alpha_2 > +\pi \\ 0, & \text{sonst} \end{cases}$$

Wobei $\alpha_i := \text{atan2}(y_{i2} - y_{i1}, x_{i2} - x_{i1})$ mit $P_{ij} := (x_{ij}, y_{ij})$.

Liegen zwei Strecken näher beieinander, als die kürzeste Strecke lang ist, so gelten diese als benachbart. Damit soll erreicht werden, dass eine Strecke nicht ihr eigener Nachbar sein kann. Auf der Suche nach linksläufigen Ketten sind Winkel aus dem Bereich $[-90^\circ, +2^\circ]$ und für rechtsläufige aus $[-2^\circ, +90^\circ]$ zugelassen. In Abbildung 4.4 sind oben sechs Strecken zu sehen zu denen der untere Graph konstruiert wird.

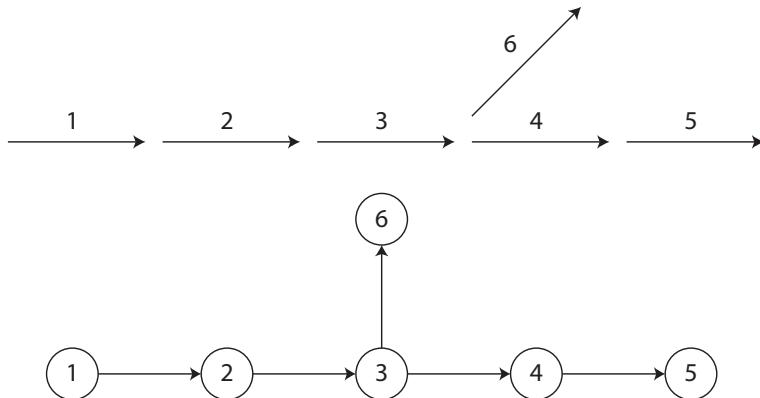


Abbildung 4.4: Graphkonstruktion

Der Algorithmus des zweiten Schrittes arbeitet auf dem zuvor erstellten Graphen. Dieser sucht nach Teilgraphen, der keine Abzweigungen enthält. Das heißt, außer dem ersten Knoten dürfen alle nur einen Vorgänger besitzen und außer dem letzten nur einen Nachfolger. Zudem muss ein Teilgraph mindestens drei Knoten enthalten. Für das Beispiel aus Abbildung 4.4 bedeutet das, dass 1-2-3 eine Kette bilden. Knoten 3 ist das Ende dieser Kette, da er zwei Nachfolger besitzt.

4.2.3 Ketten verbinden

Der Mittelkreis kann sich aus mehreren Ketten zusammensetzen. Durch Verdeckung, Mittellinie oder Bildrand kann die Linie des Kreises mehrfach unterbrochen werden (siehe Abbildung 4.5). Eine einzelne Kette ist allerdings meist nicht ausreichend um daraus eine Ellipse zu erzeugen, die den Mittelkreis wiedergibt. Aus diesem Grunde sollten Ketten zusammengeführt werden. Es gilt zu beachten, dass nach links laufende Ketten niemals mit nach rechts laufenden verbunden werden.

Zwei Ketten können dann verbunden werden, wenn sie gemeinsam eine gute Ellipse erzeugen. Das heißt, die Ellipse wird von beiden Ketten bzw. deren Strecken unterstützt. Die genauen Details darüber, in welchem Fall Strecken eine Ellipse unterstützen, werden im Abschnitt 4.3 geklärt. In manchen Fällen können Ketten, die zum Mittelkreis gehören, Ausreißer enthalten. Häufig betrifft dies Teile der Mittellinie (siehe Abbildung 4.6). Im folgenden Abschnitt 4.2.4 wird erläutert, wie die Ausreißer durch ein RANSAC-Verfahren ausgeschlossen werden.

In Algorithmus 4.1 werden jene Ketten zusammengefügt, die eine gemeinsame Ellipse erzeugen. Eine Kette kann dabei allerdings nicht Teil von mehreren zusammengefügten Ketten werden, sondern immer nur einer. Die Reihenfolge in welcher die Ketten in den Schleifen in Zeile 4 und 9 betrachtet werden, ist entscheidend für das Ergebnis.

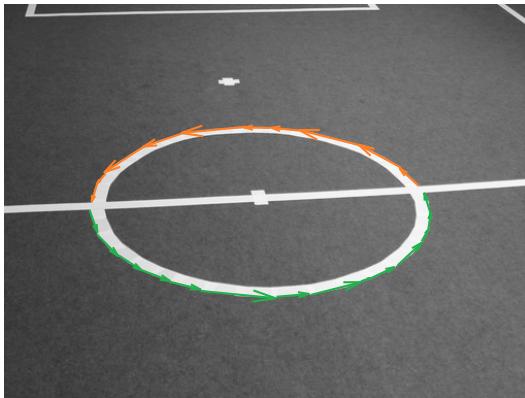


Abbildung 4.5: Kettenunterbrechungen

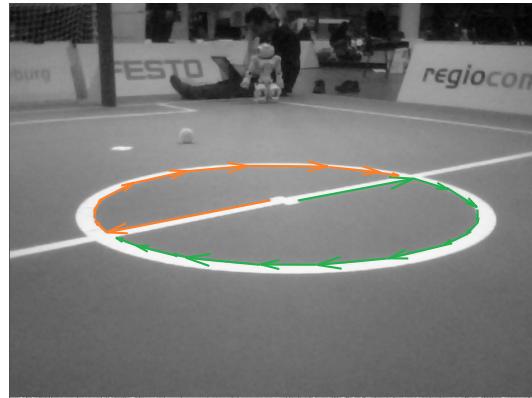


Abbildung 4.6: Ketten, die eine Strecke der Mittellinie enthält

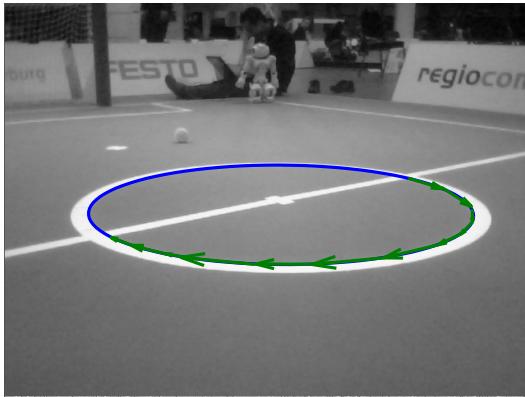


Abbildung 4.7: Verfälschung durch Ausreißer

Wenn zwei Ketten fusionieren, kann dies eine dritte Kette als Kandidaten ausschließen. Man könnte hier den Algorithmus auch so gestalten, dass eine Kette mit mehreren anderen fusioniert, würde allerdings auch die Anzahl der zu betrachtenden Möglichkeiten stark erhöhen.

4.2.4 Ausreißer entfernen

Die Methode eine Folge von Strecken als eine Kette zusammenzufassen, wie unter Kettenbildung (4.2.2) beschrieben, ist eine einfache Möglichkeit, nach Ellipsen zu suchen. Bildet eine Streckenfolge den Rand des Mittelkreises nach, so können nah gelegene fremde Strecken allerdings ebenfalls in einer gemeinsamen Kette aufgenommen werden. Diese Ausreißer verfälschen die Parameterschätzung der Ellipse, wie in Abbildung 4.7 zu sehen.

Algorithmus 4.1 Fügt Ketten zusammen, die eine gemeinsame Ellipse unterstützen

input: eine Menge von Ketten**output:** die zusammengefügten Ketten

```

1: procedure MERGE_CHAINS(chains)
2:   used(chains)  $\leftarrow$  false  $\triangleright$  ein Hash, welcher speichert, welche Ketten bereits
   zusammengefügt wurden
3:   merged  $\leftarrow$   $\emptyset$ 
4:   for c  $\in$  chains do
5:     if used(c) then  $\triangleright$  Ketten, die bereits Teil einer anderen Kette sind, werden
       übersprungen
6:       continue
7:     end if
8:     used(c)  $\leftarrow$  true
9:     for o  $\in \{c \in \text{chains} \mid \neg \text{used}(c)\}$  do  $\triangleright$  für alle bisher nicht verwendeten
       Ketten
10:    union  $\leftarrow c \cup o$ 
11:    model  $\leftarrow \text{fit}(\text{union})$ 
12:    if union supports model then
13:      used(o)  $\leftarrow$  true
14:      c  $\leftarrow$  union
15:    end if
16:  end for
17:  merged  $\leftarrow$  merged  $\cup \{c\}$ 
18: end for
19: return merged
20: end procedure

```

Dies führt nicht selten dazu, dass nur noch wenige Strecken der Kette der erzeugten Ellipse folgen und diese somit nicht mehr unterstützt wird. Daher ist es wichtig Ausreißer zuverlässig zu erkennen und zu entfernen.

Die Bereinigung von Messdaten mit Ausreißern gelingt mit dem RANSAC-Algorithmus [17]. Dieser zeichnet sich durch seine Robustheit aus und unterstützt somit das Ausgleichsverfahren, welches die Werte der Ellipse bestimmt.

Die Idee des RANSAC-Verfahrens ist, nur für einen Teil der ursprünglichen Messdaten die Modellparameter zu bestimmen. Für das resultierende Modell wird die Unterstützermenge bestimmt. Dass sind jene Werte der ursprünglichen Menge, welche einen gewissen Fehler zum ermittelten Modell nicht überschreiten. Man nennt diese Menge auch *Consensus Set*. Dies wiederholt man für viele verschiedene Teilmengen, sodass mehrere Modelle mit ihren jeweiligen Unterstützungsmengen gebildet werden. Zuletzt wählt man die beste bzw. größte Unterstützungsmenge, die gefunden wurde. Dies geschieht unter der Annahme, dass ein ausreißerfreies Modell eine große Menge an Unterstützern findet.

Im vorliegenden Fall werden drei Strecken einer Kette ausgewählt, zu denen eine Ellipse bestimmt wird. Die Anzahl der gewählten Strecken sollte so gering wie möglich sein, um die Wahrscheinlichkeit zu verringern, dass diese Ausreißer enthalten. Um eine Ellipse² anhand mehrerer Punkte zu bestimmen, sind mindestens fünf Punkte erforderlich, die auf mindestens drei unterschiedlichen Geraden liegen. Da die aus Strecken gewonnenen Punkte allerdings stets auf Geraden liegen, sind also mindestens drei Strecken notwendig.

Danach wird bestimmt, welche Strecken der Kette auch zur berechneten Ellipse passen. Dies geschieht nach zwei Kriterien: die Entfernung und die Orientierung der Strecken zur Ellipse. Das genaue Vorgehen wird im Abschnitt Modellvalidierung und Unterstützungsmenge (4.3.2) behandelt.

Wie für RANSAC üblich, wird dies mehrere Male wiederholt. Am Ende wird die beste Unterstützungsmenge ausgewählt. Dies ist die, welche die meisten Strecken enthält. Sollte es mehrere geben, die dieselbe Anzahl an Unterstützern vorzuweisen haben, so wird diejenige mit der geringsten Abweichung von der Ellipse gewählt.

4.2.5 Ketten erweitern

Verschiedene Ketten zu verbinden ist eine Form Ketten zu verlängern um damit die Genauigkeit des Models zu erhöhen. Strecken, die zwar zum Mittelkreis gehören, aber

²Das gilt auch für die anderen Kegelschnitte, die Hyperbel und Parabel.

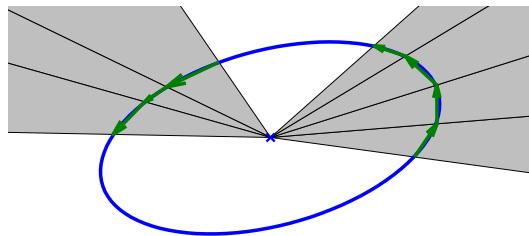


Abbildung 4.8: *Schatten*, der durch den Mittelpunkt der Ellipse und den Kettengliedern definiert wird

nicht Teil einer Kette geworden sind, bleiben vom Schritt Ketten verbinden (4.2.3) allerdings ausgeschlossen. Solche kettenlosen Strecken können in diesem Sinne als *Außenseiter* bezeichnet werden. Sie werden erst in einem weiteren Schritt, dem Erweitern von Ketten betrachtet.

Beim Erweitern werden *Kandidaten* (d.h. Strecken) gesucht, welche eine vorhandene Kette sinnvoll ergänzen. Dazu muss ein Kandidat zusammen mit der Kette ein Ellipsenmodell erzeugen können, welches beide unterstützen. Bei den Kandidaten handelt es sich um eine Teilmenge der Außenseiter. Ein Außenseiter gilt dann als Kandidat, wenn er zur bisherigen Ellipse eine sinnvolle Orientierung besitzt (siehe Abschnitt 4.3.4 Orientierungskriterium) und sich nicht im *Schatten* eines Kettengliedes befindet. Mit Schatten ist hier die Region gemeint, die sich zwischen dem Mittelpunkt der Ellipse bis zu einer Strecke und darüber hinaus erstreckt, wie in Abbildung 4.8 dargestellt.

Für das neue Modell kann es wiederum neue Kandidaten geben, da sich die Form der neuen Ellipse von der alten stark unterscheiden kann (siehe Abbildung 4.9). Darum kann man an dieser Stelle bereits in Rekursion gehen.

Abhängig vom betrachteten Kandidaten, können sich unterschiedliche Ellipsen ergeben und somit unterschiedliche Varianten der erweiterten Kette. Aus dieser so entstandenen Menge von Ketten soll die vielversprechendste ausgewählt werden. Dies erfolgt auf dieselbe Weise, wie bereits beim RANSAC-Verfahren aus 4.2.4. Es wird also die Kette mit den meisten Gliedern ausgewählt.

Dieses Vorgehen ist in Algorithmus 4.2 mit einer kleinen Modifikation dargestellt. Da sich der Verlauf der Ellipse bereits durch eine kleine Änderung stark verändern kann, könnten zugleich auf einen Schlag mehrere neue Unterstützer unter den Außenseitern auftreten. Diese werden in Zeile 11 der Kette hinzugefügt noch bevor die Prozedur in Rekursion geht.

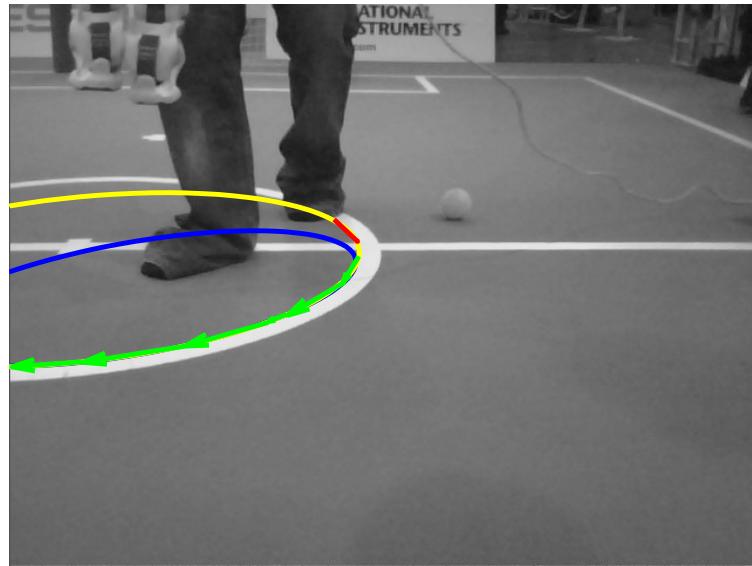


Abbildung 4.9: Veränderung der Ellipse durch Hinzufügen einer neuen Strecke

Algorithmus 4.2 Erweitert eine Kette um Außenseiter

input: eine um Außenseiter zu erweiternde Kette

output: die erweiterte Kette

```

1: procedure EXTEND_CHAIN(chain, outsiders)
2:   model  $\leftarrow$  fit(chain)
3:   candidates  $\leftarrow$  find_candidates(model, chain, outsiders)
4:   consensi  $\leftarrow \{ \text{chain} \}$   $\triangleright$  Eine Menge von Ketten, die zu Beginn die Kette selbst
   enthält
5:   for cand  $\in$  candidates do
6:     chain'  $\leftarrow$  chain  $\cup$  {cand}
7:     outsiders'  $\leftarrow$  outsiders  $\setminus$  {cand'}
8:     model'  $\leftarrow$  fit(chain')
9:     if valid_model(model') then
10:       consensus'  $\leftarrow$  find_consensus(model', outsiders')
11:       chain''  $\leftarrow$  chain'  $\cup$  consensus'
12:       outsiders''  $\leftarrow$  outsiders'  $\setminus$  consensus'
13:       consensus''  $\leftarrow$  extend_chain(chain'', outsiders'')
14:       consensi  $\leftarrow$  consensi  $\cup$  {consensus''}
15:     end if
16:   end for
17:   return select_best_consensus(consensi)
18: end procedure

```

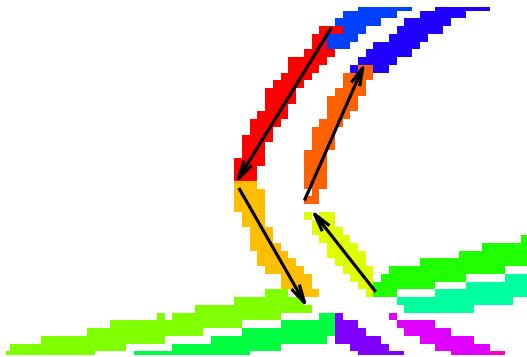


Abbildung 4.10: Krümmung einer Pixelregion und die dazu platzierte Strecke

4.3 Ellipsenmodell

Für den Prozess der Streckenauswahl ist es nötig, die Strecken mit dem Modell einer Ellipse in Verbindung zu bringen. Strecken einer Kette sollten beispielsweise eine Ellipse nachbilden. Dabei kommen die Fragen auf, wie man aus Strecken eine Ellipse bilden kann oder wie gut eine Strecke zu einer Ellipse passt. Diese werden hier in den folgenden Abschnitten beantwortet.

4.3.1 Modellberechnung

Zur Berechnung einer Ellipse aus den Daten von Liniensegmenten kann eine Parameterschätzung nach der Methode der kleinsten Fehlerquadrate verwendet werden. Dazu entnimmt man den Strecken eine Reihe von Abtastpunkten, die als Datengrundlage für die Parameterschätzung von Kegelschnitten (2.4) dienen.

Mindestens zwei Punkte sollten je einer Strecke entnommen werden, sodass Ausmaß und Orientierung in einem gewissen Sinne Rechnung getragen wird. An dieser Stelle muss im Auge behalten werden, dass die Strecken Krümmungen der Ellipse annähern. Das bedeutet, dass einige Punkte näher an der Ellipse liegen als andere. Eine geeignete Wahl der Abtastpunkte sollte hier systematische Fehler verringern.

Die vom LSD ermittelten Strecken entspringen aus sogenannten Line-Support-Regionen. Das sind jene Pixel der Kontur, die die Strecke anzunähern versucht. Die Strecke wird dabei so platziert, dass sich ihr Zentrum im Massepunkt der Pixelregion befindet. Deshalb befinden sich die äußersten Punkte der Strecke näher an der Außenseite der Krümmung und das Zentrum näher an der Innenseite (siehe Abbildung 4.10). Aus diesem Grund werden genau diese Punkte entnommen und bei der Parameterschätzung verwendet, sodass sich deren Abweichungen kompensieren.

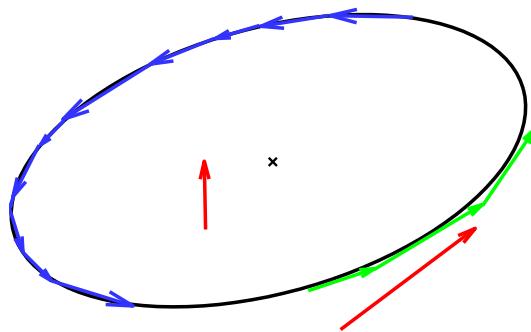


Abbildung 4.11: Unterstützungsmenge (grün) eines Modells

Eine Alternative dazu ist die Koordinaten der Pixel aus den Line-Support-Regionen zu entnehmen. Diese Methode liefert wesentlich mehr Messpunkte. Dies dürfte zu genaueren Ergebnissen führen, erfordert jedoch auch einen erhöhten Rechenaufwand.

4.3.2 Modellvalidierung und Unterstützungsmenge

Die Aufgabe der Modellvalidierung ist es zu überprüfen, ob ein Modell von einer Menge von Strecken unterstützt wird. Das heißt, die Strecken nähern den Verlauf der Ellipse ausreichend an. Anders ausgedrückt: Die Menge der Strecken, aus denen das Modell berechnet wurde, ist eine Teilmenge der Unterstützungsmenge. Ein Beispiel für eine Unterstützungsmenge eines Modells ist in Abbildung 4.11 gegeben. Die blauen Strecken erzeugen die Ellipse. Unterstützer sind grün und sonstige Strecken rot gefärbt.

Eine Strecke gilt dann als Element der Unterstützungsmenge, wenn sie eine Entfernung im Sinne des im Abschnitts 4.3.3 definierten Entfernungsmaßes nicht übersteigt und das Orientierungskriterium (4.3.4) erfüllt.

4.3.3 Entfernungsmaß

Bevor wir eine Definition für die Entfernung zwischen Strecke und Ellipse aufstellen, sehen wir uns zunächst den Fall für einen Punkt an. Die Entfernung zwischen zwei Punkten $P := (p_1, \dots, p_n)$ und $Q := (q_1, \dots, q_n)$ ist gegeben durch den euklidischen Abstand:

$$d(P, Q) := \|P - Q\| = \sqrt{\sum_i^n (p_i - q_i)^2} \quad (4.1)$$

Darauf aufbauend lässt sich der Abstand zwischen einem Punkt P und einer Ellipse E als der geringste Abstand zwischen einem Punkt P_E auf E und P definieren:

$$d(P, E) := \min_{P_E \in E} \{ \|P - P_E\| \} \quad (4.2)$$

Eine Idee wäre nun im Fall einer Strecke $S = \overline{P_1 P_2}$ in einer ähnlichen Weise den geringsten Abstand zwischen einem Punkt auf der Strecke und der Ellipse zu definieren. Jedoch würde dadurch beispielsweise die Länge der Strecke nicht berücksichtigt. Außerdem hätte dann jede Strecke, die die Ellipse schneidet, den Abstand null.

Eine andere Möglichkeit ist, die Entfernung zur Ellipse über alle Punkte der Strecke zu integrieren:

$$d(S, E) := \int_S d(P, E) \, dP \quad (4.3)$$

Allerdings ist die Berechnung von (4.2) nicht trivial, sondern erfolgt in einem iterativen Verfahren. Demnach müsste die Lösung von (4.3) ebenfalls numerisch erfolgen.

Ein anderer Ansatz geht von der Definition einer Ellipse aus: Alle Punkte auf der Ellipse haben eine konstante Entfernung zu deren Brennpunkten F_1 und F_2 , die der Länge der Hauptachse entspricht.

$$\|F_1 - P_E\| + \|F_2 - P_E\| = 2a \quad (4.4)$$

Für alle anderen Punkte, die nicht auf der Ellipse liegen, weicht der Wert ab. Ausgehend von (4.4) lässt sich dazu ein Fehler definieren, den ich Punkt-Brennpunkt-Fehler (PFE) nennen möchte:

$$\text{PFE}(P, E) := \left| \|F_1 - P\| + \|F_2 - P\| - 2a \right| \quad (4.5)$$

Wobei $E = \{P_E \mid \|F_1 - P_E\| + \|F_2 - P_E\| = 2a\}$. In Abbildung 4.12 sind die Konturen von unterschiedlichen PFE-Werten einer Ellipse (gestrichelt) gegeben. Man erkennt, dass es sich hierbei um Ellipsen handelt.

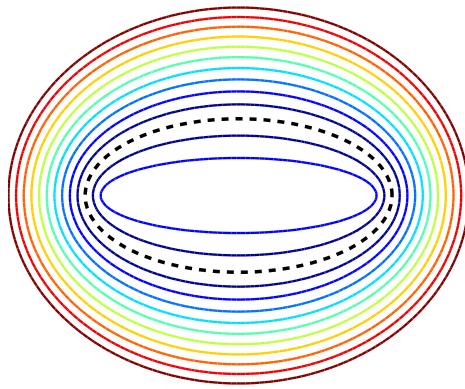


Abbildung 4.12: PFE-Feld

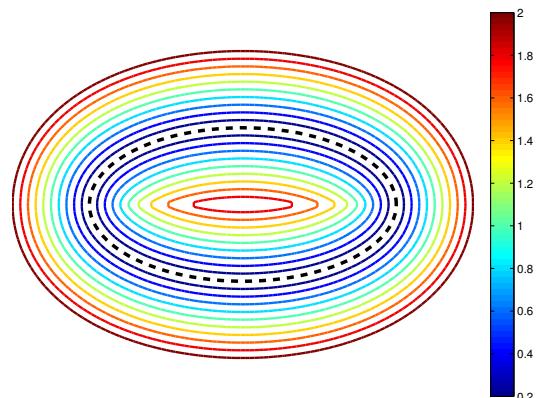
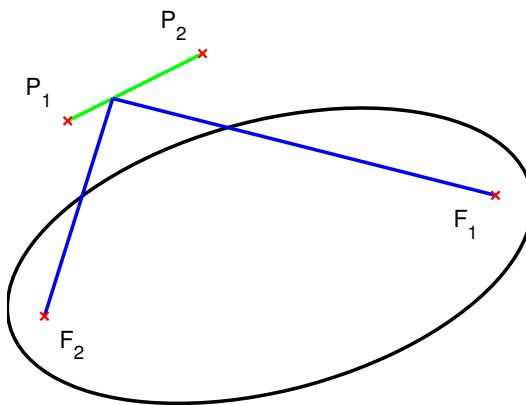


Abbildung 4.13: Geometrische Distanz

Abbildung 4.14: Integration des PFE-Feldes entlang der Strecke $\overline{P_1P_2}$

Anhand dieses Ansatzes lässt sich nun wie schon bei (4.3) ein Integral ableiten, sodass man eine Definition für Strecken erhält (siehe auch Abbildung 4.14):

$$\text{aLFE}(S, E) := \int_S \text{PFE}(P, E) dP \quad (4.6)$$

$$\text{rLFE}(S, E) := \int_S \frac{\text{PFE}(P, E)}{2a} dP = \frac{\text{aLFE}(S, E)}{2a} \quad (4.7)$$

Im Gegensatz zu (4.3) lassen sich diese Integrale als Elementarfunktionen darstellen. aLFE ist der absolute und rLFE der relative Strecken-Brennpunkt-Fehler, welcher gegenüber Skalierung invariant ist.

Zum Vergleich von Strecken unterschiedlicher Länge, sind diese beiden Maße allerdings ungeeignet, da lange Strecken gegenüber kurzen „bestraft“ werden. Eine sehr kurze Strecke kann beispielsweise einen wesentlich höheren geometrischen Abstand zur Ellipse besitzen, als eine nah liegende lange Strecke. Darum existieren die

folgenden normierte Varianten von (4.6) und (4.7):

$$\text{naLFE}(S, E) := \frac{\text{aLFE}(S, E)}{\|S\|} \quad (4.8)$$

$$\text{nrLFE}(S, E) := \frac{\text{rLFE}(S, E)}{\|S\|} \quad (4.9)$$

Der naLFE-Wert gibt dabei den durchschnittlichen PFE-Wert aller Punkte auf der Strecke an. Analog entspricht der nrLFE-Wert dem durchschnittlichen $\frac{\text{PFE}}{2a}$ -Wert.

Eine Schwäche der Brennpunkt-Fehler ist, dass sie in Richtung der Nebenachse eine höhere Toleranz aufweisen. Das heißt, dass der Fehler entlang der Hauptachse gegenüber der kleinen schneller zu- bzw. abnimmt. Dies wird in Abbildung 4.12 an den Konturen deutlich, die nach außen hin immer kreisförmiger werden. Der Fehler ist also winkelabhängig.

Um diesen Effekt zu kompensieren, kann man das Koordinatensystem vor der Berechnung transformieren. Würde man beispielsweise das System entlang der Nebenachse soweit dehnen, dass die Ellipse zu einem Kreis wird, so kehrt sich der Effekt ins Gegenteil. Die Toleranz entlang der Hauptachse ist hoch und entlang der Nebenachse gering.

Im Falle einer Ellipse, deren Hauptachse sich auf der x-Achse und deren Nebenachse sich auf der y-Achse befindet, kann man das System wie folgt transformieren:

$$x' = x$$

$$y' = \alpha y$$

Die Transformation hat auf die Strecke S und der Ellipse E folgenden Einfluss:

$$P'_1 = (x_1, \alpha y_1)$$

$$P'_2 = (x_2, \alpha y_2)$$

$$a' = a$$

$$b' = \alpha b$$

$$F'_1 = (-\sqrt{a'^2 - b'^2}, 0) = (-\sqrt{a^2 - (\alpha b)^2}, 0)$$

$$F'_2 = (+\sqrt{a'^2 - b'^2}, 0) = (+\sqrt{a^2 - (\alpha b)^2}, 0)$$

Wobei $P_i = (x_i, y_i)$.

Es gilt nun α so zu wählen, dass der Fehler möglichst winkelunabhängig wird. Für zwei Punkte $P = (a + \delta, 0)$ und $Q = (0, b + \delta)$ (siehe Abbildung 4.15) soll der Fehler

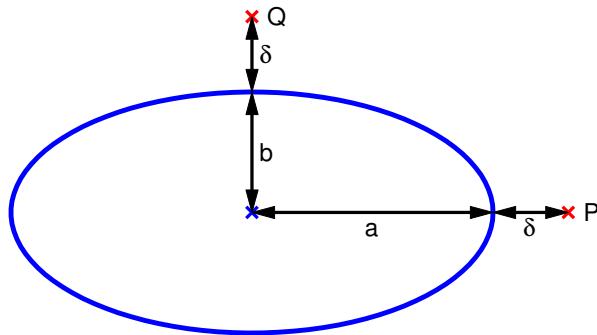


Abbildung 4.15: Fixierung des Fehlers an den Punkten P und Q

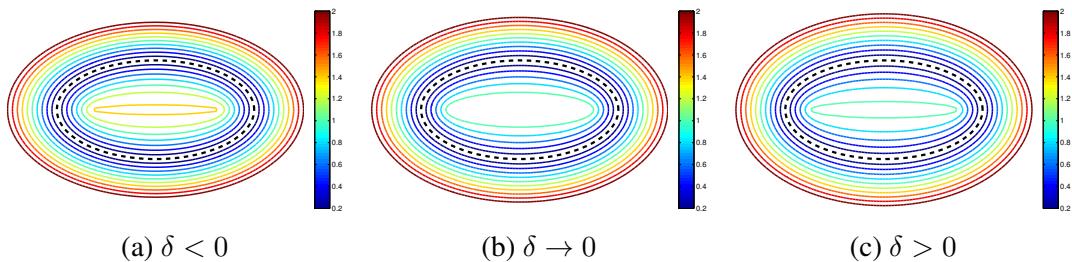


Abbildung 4.16: PFE-Felder für unterschiedliche δ -Werte

im transformierten System gleich sein:

$$\text{PFE}(P', E') \stackrel{!}{=} \text{PFE}(Q', E') \quad (4.10)$$

$$= 2d = 2\sqrt{(\alpha b + \alpha\delta)^2 + a^2 - (\alpha b)^2} - 2a \quad (4.11)$$

$$\Leftrightarrow \alpha = \sqrt{\frac{(a+\delta)^2 - a^2}{(b+\delta)^2 - b^2}} \underset{\delta \neq 0}{=} \sqrt{\frac{2a-\delta}{2b-\delta}} \quad (4.12)$$

Nach dieser Methode ergibt sich im PFE-Feld eine Kontur für den Wert $|\delta|$, welche eine Ellipse E_δ mit den Parametern $a_\delta = a + \delta$ und $b_\delta = b + \delta$ ergibt. Somit kann man eine Kontur für einen beliebigen δ -Wert fixieren. Je nachdem, wie man nun δ wählt, ergeben sich unterschiedliche PFE-Felder (siehe Abbildung 4.16). Einen Wert für δ sollte man dort ansetzen, wo die Unterscheidung der PFE-Werte kritisch ist. Das betrifft besonders Schwellwerte. Möchte man beispielsweise Strecken verwerfen, die außerhalb von E_δ liegen, so betrifft dies jene dessen naLFE-Wert δ übersteigen.

4.3.4 Orientierungskriterium

Das Orientierungskriterium soll angeben, ob eine Strecke zu einer Ellipse geeignet orientiert ist. Dabei gilt zu berücksichtigen, dass die Strecken des LSDs zwischen Start-

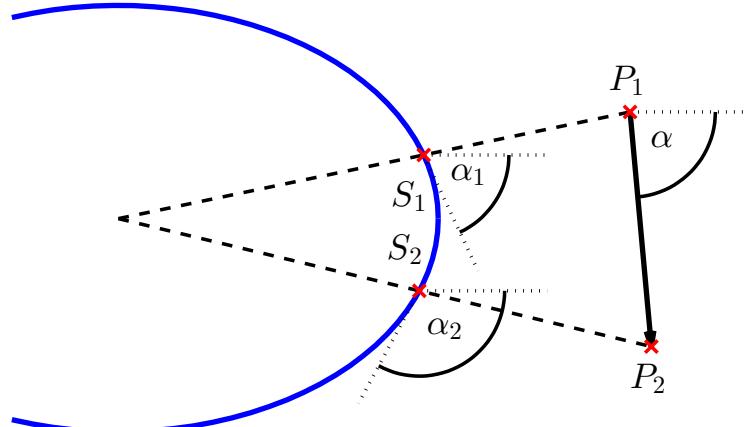


Abbildung 4.17: Schnittpunkte der Geraden mit der Ellipse

und Endpunkt unterscheiden. Außerdem werden Ellipsen stets nur von links- oder rechtsläufigen Ketten gebildet, sodass man einer Ellipse eine Rotation (im und gegen Uhrzeigersinn) zuordnen kann.

Wir betrachten eine Ellipse E und eine Strecke $S = \overline{P_1 P_2}$. Die Hauptachse befindet sich auf der x-Achse und die Nebenachse auf der y-Achse. zieht man zwei Gerade durch das Zentrum und die beiden Punkte P_1 und P_2 , so entstehen zwei Schnittpunkte S_1 und S_2 auf der Ellipse (siehe Abbildung 4.17). An diesen Punkten kann man zwei Tangenten der Ellipse anlegen. Diese haben dann die folgenden Winkel:

$$\alpha_1 = \text{atan}2(b^2 x_1, -a^2 y_1) \quad (4.13)$$

$$\alpha_2 = \text{atan}2(b^2 x_2, -a^2 y_2) \quad (4.14)$$

Wobei $P_i = (x_i, y_i)$. a und b geben die Längen der beiden Achsen der Ellipse an.

Die Orientierung der Strecke S , gegeben durch

$$\alpha = \text{atan}2(y_2 - y_1, x_2 - x_1), \quad (4.15)$$

muss sich nun zwischen den beiden Winkeln α_1 und α_2 befinden. Wenn die Ellipse gegen den Uhrzeigersinn „rotiert“, so muss folgende Bedingung erfüllt sein:

$$(\alpha - \alpha_1) \bmod 2\pi \leq (\alpha_2 - \alpha_1) \bmod 2\pi \quad (4.16)$$

Rotiert sie im Uhrzeigersinn, so vertauscht α_1 und α_2 miteinander und dreht α um 180° :

$$(\alpha + \pi - \alpha_2) \bmod 2\pi \leq (\alpha_1 - \alpha_2) \bmod 2\pi \quad (4.17)$$

4.4 Verifikation

Bis zu dieser Stelle wurde beschrieben, wie relevante Strecken ausgemacht und daraus Ellipsen erzeugt werden. Insofern die Ränder des Mittelkreises erkannt worden sind, müssen diese nun einander zugeordnet werden. Der innere Rand liegt dabei als rechtsläufige Ellipse E_R vor und der äußere als linksläufige Ellipse E_L . Der Mittelkreis wird also als Ellipsenpaar repräsentiert.

Algorithmus 4.3 sucht nach möglichen Ellipsenpaaren, die zum Mittelkreis gehören. Dabei gibt es drei Kriterien:

1. Die Ellipse E_R befindet sich innerhalb von E_L .
2. Die Formen beider Ellipsen sind ähnlich zueinander.
3. Beide Ellipsen besitzen eine ähnliche Größe. Das heißt, E_R sollte gegenüber E_L nicht zu klein geraten.

Die Form der Ellipse, lässt sich durch das Verhältnis ihrer beiden Achsen beschreiben:

$$r = \frac{a}{b} \quad (4.18)$$

Für die beiden Ellipsen bedeutet dies, dass sich wiederum ihre Werte r_L und r_R nicht allzu sehr unterscheiden dürfen:

$$r_{min} \leq \frac{r_L}{r_R} = \frac{a_L/b_L}{a_R/b_R} \leq r_{max} \quad (4.19)$$

r_{min} und r_{max} geben die Grenzen bzw. den Toleranzbereich an, in welchen sich die Formen beider Ellipsen unterscheiden dürfen.

Die Größe der Ellipsen wird ebenfalls durch ihre Achsen bestimmt. Beide Achsen der größeren Ellipse sollten gegenüber der kleineren Ellipse nicht zu groß geraten:

$$\max \left\{ \frac{a_L}{a_R}, \frac{b_L}{b_R} \right\} \leq size_{max} \quad (4.20)$$

Nach Anwendung des Algorithmus 4.3 müssen die gefundenen Paare noch verifiziert werden. Es soll geprüft werden, ob der Mittelkreis so verläuft, wie es von dem Paar beschrieben wird. Zwischen den beiden Ellipsen E_L und E_R sollte sich das Weiß der Mittelkreislinie befinden und außerhalb, das Grün des Spielfeldes. Unterbrechungen des Mittelkreises durch Hindernisse oder der Mittellinie sollten dennoch toleriert werden.

Zur Überprüfung muss das Eingabebild zunächst entlang des mutmaßlichen Mittelkreises abgetastet werden. Dies erfolgt in mehreren Schritten (siehe auch Abbildung 4.18):

Algorithmus 4.3 Sucht nach Ellipsenpaare

input: Die links- und rechtsläufigen Ellipsen
output: Eine Menge von Ellipsenpaaren

```

1: procedure FIND_PAIRS( $models_l, models_r$ )
2:    $pairs \leftarrow \emptyset$ 
3:   for  $model_l \in models_l$  do
4:     for  $model_r \in models_r$  do
5:        $(a_l, b_l) \leftarrow \text{get\_axes}(model_l)$ 
6:        $(a_r, b_r) \leftarrow \text{get\_axes}(model_r)$ 
7:        $\text{similar\_shape} \leftarrow r_{min} \leq \frac{a_l/b_l}{a_r/b_r} \leq r_{max}$ 
8:        $\text{similar\_size} \leftarrow \max(a_l/a_r, b_l/b_r) \leq size_{max}$ 
9:       if  $model_r$  is within  $model_l \wedge \text{similar\_shape} \wedge \text{similar\_size}$  then
10:         $pairs \leftarrow pairs \cup \{(model_l, model_r)\}$ 
11:      end if
12:    end for
13:  end for
14:  return  $pairs$ 
15: end procedure
```

1. Abtasten der inneren Ellipse E_R in regelmäßigen Abständen.
2. Anlegen eines Strahls an einem Abtastpunkt P . Dieser zeigt nach außen und befindet sich in einem Winkel senkrecht zur Ellipse E_R .
3. Bestimmung des Schnittpunkts S des Strahls mit der äußeren Ellipse E_L . Da sich E_R vollständig innerhalb von E_L befindet, existiert genau ein Schnittpunkt.
4. Bestimmung von zwei Punkten Q_1 und Q_2 , von denen sich einer innerhalb von E_R und der andere außerhalb von E_L befindet. Dabei gilt:

$$Q_1 := P - (S - P) \quad (4.21)$$

$$Q_2 := S + (S - P) \quad (4.22)$$

5. Abtasten der Strecke $\overline{Q_1 Q_2}$ und Entnahme der Farbinformationen aus dem Bild an den Abtastpunkten.

Durch Befolgung dieser Schritte entnimmt man spaltenweise Abtastpunkte, sodass ein gerade gezogenes Abbild der Ellipse wie in Abbildung 4.19 entsteht. Dieses Abbild kann man in drei gleich hohe horizontale Bereiche einteilen. In der Mitte befindet sich die Region, in der die weiße Linie des Mittelkreises zu sehen sein sollte. In den äußeren Bereichen sollte sich das grüne Spielfeld befinden.

Aufgrund der Schwierigkeit Weiß zu erkennen, wird nach dem Algorithmus 4.4 vorgegangen. Dabei wird geprüft, ob sich in der in der Mitte des Abtastbildes ein heller

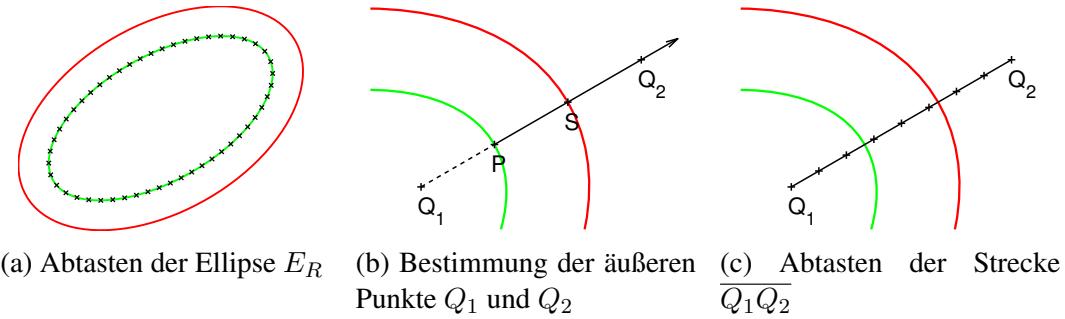


Abbildung 4.18: Schritte der Abtastung

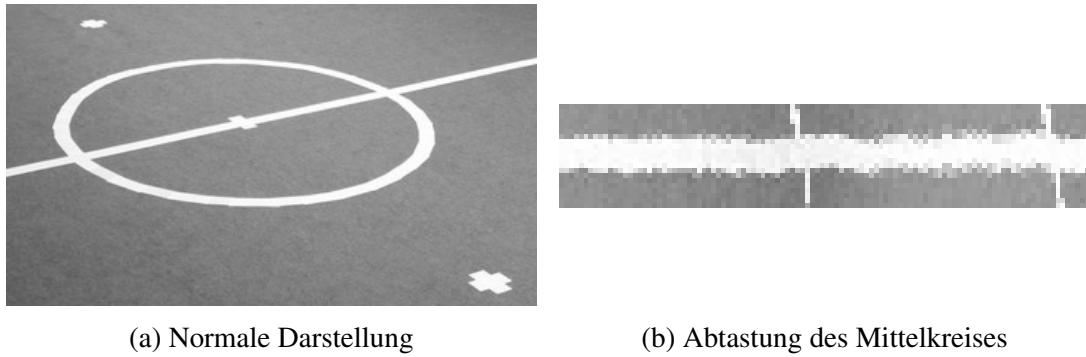


Abbildung 4.19: Abgetasteten Mittelkreise

Streifen befindet. Da Bereiche, welche Verdeckungen oder die Mittellinie enthalten, störend wirken, werden diese ausgeblendet. Dazu werden nur jene Spalten betrachtet, welche in der Ober- und Unterhälfte grüne Pixel enthalten.

Um den Einfluss von Rauschen zu reduzieren, wird das Abbild geglättet, welches zuvor in Graustufen umgewandelt wurde. Anschließend wird spaltenweise der höchste Helligkeitswert festgestellt. Dieser sollte sich möglichst in der Mitte befinden. Je geringer die Abweichungen, desto geringer der Fehler und umso höher die Güte des Mittelkreises. Für eine bessere Justierung von Parametern wird das Koordinatensystem des Abbilds normiert. Die Höhe aller drei Bereiche wird dabei auf 1 festgelegt und die x-Achse in die Mitte versetzt. Die summierten Fehlerquadrate dürfen ein bestimmtes Maximum nicht überschreiten, ansonsten wird das Paar nicht als Mittelkreis akzeptiert.

Algorithmus 4.4 Verifiziert ein Ellipsenpaar

input: Das Eingabebild $image$ und das Ellipsenpaar $(model_l, model_r)$

output: `true`, wenn das Paar verifiziert wurde, ansonsten `false`.

```
1: procedure VERIFY_PAIR( $image, model_l, model_r$ )
2:    $s \leftarrow \text{sample}(image, model_l, model_r)$ 
3:    $s' \leftarrow \text{columns from } s \text{ which contain green pixels on both horizontal halves}$ 
4:    $(w, h) \leftarrow \text{size}(s')$                                  $\triangleright$  Höhe  $h$  und Breite  $w$  von  $s'$ 
5:    $s'' \leftarrow \text{smooth(rgb2gray}(s'))$ 
6:    $I \leftarrow \text{row indices of } s''$  maximum column values
7:    $error \leftarrow \frac{1}{w} \sum_{i \in I} \left( \frac{3}{h-1} i - \frac{3}{2} \right)^2$ 
8:   return  $error \leq error_{max}$ 
9: end procedure
```

Kapitel 5

Umsetzung

Die in Kapitel 4 entwickelte Lösung wurde in MATLAB R2013a implementiert. Davor ausgenommen ist der LSD, welcher in Version 1.6 als C-Code vorliegt und für die Ausführung unter Windows kompiliert wurde.

Der MATLAB-Code umfasst mehrere Funktionen, welche das beschriebene Vorgehen in den Abschnitten des Lösungskonzepts umsetzen. Eine Zuordnung ist in Tabelle 5.1 aufgelistet.

Die Verarbeitungsschritte erfolgen in einer Art „Fließbandverfahren“, sodass die Ausgabe eines Schritts die Eingabe des nächsten ist. Diese Folge ist in Abbildung 5.1 dargestellt. Dort ist auch eine Zuordnung der Funktionen in Streckenselektion und Ellipsenmodell zu sehen, welche parallel zueinander stehen.

Die Hauptfunktion, welche für ein Bild die Mittelkreiserkennung ausführt, heißt `center_circle_detection`. Sie nimmt ein RGB-Bild entgegen und gibt nach der Verarbeitung die Modelle von erkannten Mittelkreisen zurück.

Lösungsabschnitt	MATLAB-Funktion
Strecken filtern	<code>sieve_segs</code>
Kettenbildung	<code>detect_chains</code>
Ketten verbinden	<code>merge_chains</code>
Ausreißer entfernen	<code>improve_chains</code>
Ketten erweitern	<code>extend_chains</code>
Modellberechnung	<code>fitellipse_segs</code>
Modellvalidierung und Unterstützungsmenge	<code>validate_model</code>
Entfernungsmaß	<code>line_rating</code>
Orientierungskriterium	<code>filter_angle</code>
Verifikation	<code>verify_models</code>

Tabelle 5.1: MATLAB-Funktionen

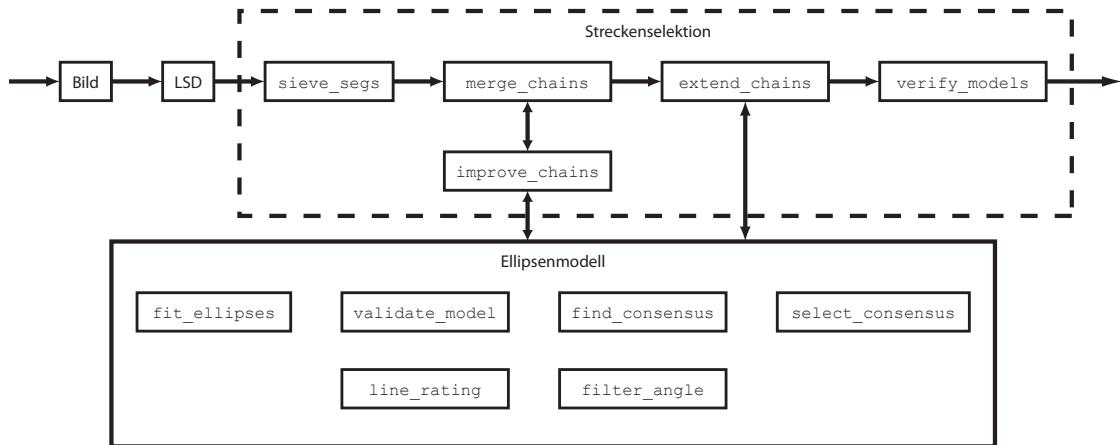


Abbildung 5.1: Beziehung der MATLAB-Funktionen

5.1 Line Segment Detector

Zur Erkennung von Liniensegmenten wird der Line Segment Detector (Version 1.6) verwendet. Dieser lag als C-Quellcode vor und wurde geringfügig angepasst, dessen Verhalten jedoch nicht verändert. Das Programm wurde um die Funktionalität ergänzt, weitere Bildformate laden zu können, wie JPG oder PNG. Dies zog allerdings eine Abhängigkeit zur OpenCV-Bibliothek in Version 2.4.3 nach sich. Prinzipiell lässt sich der Quellcode sich auch für andere Plattformen wie Linux übersetzen. Dazu muss jedoch die Abhängigkeit zu OpenCV erfüllt werden.

Das kompilierte Programm ist in Windows über die Kommandozeile bedienbar. Das folgende Beispiel zeigt eine einfache Anwendung, in der die Strecken eines Bildes in eine Textdatei ausgegeben werden:

```
lsd my_picture.jpg line_segments.txt
```

Es ist möglich weitere Optionen anzugeben, mit denen man gewisse Parameter des Algorithmus festlegen kann. Zudem kann man auch SVG- und EPS-Dateien erzeugen, welche die Lage der Strecken anzeigen. Interessant ist auch die Option, die Line-Support-Regionen der Strecken in ein Bild ausgeben zu lassen.

Von MATLAB aus lassen sich die benötigten LSD-Dateien automatisch erzeugen, indem es das Kommandozeilenprogramm aufruft.

5.2 Streckenselektion

5.2.1 sieve_segs

Die initiale Filterung der Strecken basierend auf der Farbe ihrer Umgebung wird von der `sieve_segs`-Funktion übernommen. Zur Bestimmung der Farbe wird der Funktion eine „Grünkarte“ übergeben. Das ist ein zweidimensionales Binärfeld mit derselben Größe des Eingabebildes. Für jedes Segment wird die Grünkarte wie im Abschnitt 4.2.1 erläutert abgetastet. Dazu wurden die Werte $\alpha = 2$, $m = 5$ und $n = 5$ gewählt, sodass insgesamt pro Segment zehn Pixel abgetastet werden müssen.¹

5.2.2 detect_chains

Die Funktion `detect_chains` implementiert die Kettenbildung. Dies erfolgt in zwei Schritten, die von den Funktionen `build_graph` und `build_chains` durchgeführt werden. Diese entsprechen dem Vorgehen, wie es schon im Lösungskapitel beschrieben wurde.

Entscheidend für die Kettenbildung sind die Schwellwerte d_{min} , α_{min} und α_{max} . Diese legen die maximale Entfernung zwischen Nachbarn und den zulässigen Winkelbereich fest. Vor allem durch den Winkelbereich wird definiert, welche Ketten als links- oder rechtsläufig aufgefasst werden. Aufgrund dieser Unterscheidung wird `detect_chains` zwei mal aufgerufen. Dies ist ein wiederkehrendes Muster, da die beiden Kettenarten meist getrennt betrachtet werden müssen.

5.2.3 merge_chains

`merge_chains` realisiert den Algorithmus 4.1 unter Verwendung von `improve_chains`, welche das Entfernen von Ausreißern (4.2.4) implementiert.

Zeile 12 des Algorithmus „*union supports model*“ ist noch relativ ungenau definiert. Das genaue Vorgehen von `merge_chains` beinhaltet den Aufruf von `improve_chains`, dessen Rückgabe geprüft wird. Bei der Rückgabe handelt es sich um die vom RANSAC-Verfahren ermittelte Unterstützungsmenge von der Vereinigung der beiden Ketten c und o . Diese wird aber nur dann akzeptiert, wenn sie auch Glieder beider Ketten enthalten. Außerdem muss die zusammengefügte Kette, welche der Unterstützungsgröße entspricht, mehr Glieder enthalten, als die einzelnen Ketten für

¹Tatsächlich werden sogar zwanzig abgetastet, da immer die linke als auch die rechte Seite des Segments betrachtet werden. Genutzt wird jedoch nur die linke.

sich alleine. Konnte eine Kette nicht mit einer anderen verbunden werden, so werden zumindest ihre Ausreißer mithilfe von `improve_chains` entfernt.

5.2.4 `improve_chains`

Die Aufgabe von `improve_chains` ist es, Ketten von Ausreißern zu bereinigen. Dazu wird auf das RANSAC-Verfahren zurückgegriffen. Zunächst wird jedoch überprüft, ob die Kette überhaupt Ausreißer enthält, indem sie einer Validierung (siehe `validate_model`) unterzogen wird.

Erst wenn die Validierung fehlschlägt, wird der RANSAC-Algorithmus, implementiert durch `ransac_chain` ausgeführt. Die MATLAB-Implementierung hält sich weitestgehend an das Lösungskonzept. Es werden zwanzig Iterationen durchgeführt. Zur Ermittlung der Unterstützungsmengen wird die Funktion `find_consensus` und für die anschließende Auswahl `select_consensus` verwendet.

5.2.5 `extend_chains`

Neben `merge_chains` ist auch `extend_chains` eine Funktion, welche Ketten um weitere Elemente anreichert, wie schon unter 4.2.5 beschrieben. Dieser Schritt erweitert eine Kette um sogenannte *Außenseiter* und folgt nach dem Verbinden von Ketten.

Der Algorithmus 4.2 wurde um einen Aspekt erweitert, sodass unnötige Rekursionsschritte vermieden werden. Bei der Erweiterung von Ketten müssen viele Alternativen betrachtet werden, wenn es darum geht, Kandidaten der Kette hinzuzufügen. Dabei kann es vorkommen, dass mehrere Wege zum selben Ziel führen. Das heißt, durch das Hinzufügen eines Kandidaten zu der Kette kann eine neue Kette entstehen, die in einem anderen Rekursionszweig bereits betrachtet worden ist. Darum wird jedem Zweig eine Menge aller bisher betrachteten Ketten mitgegeben um eine mehrfache Betrachtung auszuschließen.

5.3 Ellipsenmodell

5.3.1 `fitellipse_segs`

Die Parameterschätzung von Ellipsen auf Basis von Strecken wird von der `fitellipse_segs`-Funktion durchgeführt. Dazu werden die Strecken an drei Punkten (Anfang, Mitte und Ende) abgetastet und an die Funktion `fitellipse` übergeben.

Der Quell-Code der eigentlichen Ausgleichsrechnung stammt von Richard Brown² und bietet mehrere Modi zur Berechnung des Ellipsenmodells an.

Neben der linearen Lösung zur Minimierung des algebraischen Fehlers, beherrscht der Algorithmus von Brown auch nicht-lineare Lösungen, welche den geometrischen Fehler iterativ minimieren. Dazu kann man aus zwei Nebenbedingungen zur Lösung des Gleichungssystems wählen:

$$\lambda_1^2 + \lambda_2^2 = 1 \quad (5.1)$$

$$\lambda_1 + \lambda_2 = 1 \quad (5.2)$$

Bei der ersten Gleichung (5.1) handelt es sich um die Bookstein-Bedingung und bei der zweiten (5.2) um die Spur-Bedingung. λ_1 und λ_2 sind die Eigenwerte der Matrix A aus Gleichung (2.1).

Zur Bestimmung der Parameter wird die lineare Lösungsmethode unter Verwendung der Spur-Bedingung verwendet. Abschließend wird sichergestellt, dass a stets die Hauptachsen ist und sich die Orientierung β der Ellipse stets im Bereich $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ befindet. Außerdem werden Ellipsen verworfen, deren Nebenachse kürzer als 10 Pixel ist.

5.3.2 validate_model

Mit `validate_model` lässt sich überprüfen, ob ein Ellipsenmodell zu einer gegebenen Menge von Strecken passt. Diese Validierung wird durch Bestimmung der Unterstützungsmaße realisiert. Ein Modell wird erst dann als gültig anerkannt, wenn alle gegebenen Strecken auch zur Unterstützungsmaße gehören. Dazu wird die Funktion `find_consensus` verwendet.

5.3.3 find_consensus

`find_consensus` bestimmt die Unterstützungsmaße innerhalb einer gegebenen Menge von Strecken zu einem Ellipsenmodell. Eine Strecke gilt dann als Unterstützer, wenn sie das Orientierungskriterium erfüllt und sie eine Maximalentfernung zur Ellipse nicht überschreitet. Dazu werden die zwei Funktionen `filter_angle` und `line_rating` verwendet.

Die Kriterien, wann eine Strecke als Unterstützer gilt, sind von drei Parametern abhängig und der Angabe, ob es sich um einer links- oder rechtsdrehende Ellipse handelt.

²Verfügbar unter <http://www.mathworks.com/matlabcentral/fileexchange/15125-fitellipse-m>

Der erste Parameter err_{thr} gibt die obere Grenze für die Entfernung zwischen Ellipse und Strecke nach 4.3.3 an und hat den Wert 10. Um die Winkelabhängigkeit für flache Ellipsen zu kompensieren wird das Koordinatensystem, wie in Abschnitt 4.3.3 beschrieben, transformiert. Dazu wird $\delta = -err_{thr}$ gesetzt, sodass im sich PFE-Feld eine Kontur für den Wert 10 mit der Form einer Ellipse ergibt, dessen Achsen um jeweils 20 Pixel verkürzt wurden. Grund für die Fixierung der Kontur innerhalb der Ellipse ist, dass das Feld zum Zentrum hin weniger stark ansteigt, als es im Außenbereich nach außen der Fall ist. Eine Fixierung außerhalb der Ellipse würde diesen Effekt verstärken. Eine Alternative ist, ein Feld aus zwei PFE-Felder für den Außen- und Innenbereich zusammenzusetzen.

Die anderen beiden Parameter sind κ , welches eine Toleranz für die Winkelgrenzen α_1 und α_2 aus 4.3.4 einräumt, und τ , welches den Maximalabstand zwischen α_1 und α_2 angibt. Allerdings wird τ stets auf den Wert π gesetzt, sodass sich keine Restriktionen ergeben. Besonders bei flachen Ellipsen nahe der Scheitelpunkte führen selbst geringe Änderungen der Lage von Strecken zu großen Änderungen der Winkel α_1 und α_2 , weshalb κ den Wert $\frac{\pi}{8} = 22.25^\circ$ hat.

5.3.4 select_consensus

Beim RANSAC-Verfahren sowie beim Erweitern von Ketten werden verschiedene Unterstützungsmengen gegenübergestellt, um darüber zu entscheiden, welche dieser Mengen die beste darstellt. Die Auswahl wird von der Funktion `select_consensus` übernommen.

Ausschlaggebendes Kriterium für die Güte einer Unterstützungsmenge C ist die Anzahl ihrer Elemente. Im einfachen Fall wird lediglich die Menge mit den meisten Elementen ausgewählt. Haben jedoch mehrere dieselbe Anzahl von Elementen muss ein anderes Maß herangezogen werden. Dazu wird bestimmt, wie gut die Unterstützungsmengen mit dem Ellipsenmodell harmonieren, welches sie produzieren. Das Maß für die „Harmonie“ wird durch die Entfernung der Strecken zu dieser Ellipse definiert:

$$\text{naCFE}(C, E) = \frac{\sum_{S \in C} \text{aLFE}(S, E)}{\sum_{S \in C} \|S\|} \quad (5.3)$$

Je geringer naCFE ausfällt, desto höher die Harmonie. Unter den Unterstützungsmengen mit den meisten Elementen wird also diejenige mit geringsten naCFE-Wert ausgewählt.

5.3.5 line_rating

Die Berechnung der Entfernungsmaße aLFE, rLFE, naLFE und nrLFE wird von `line_rating` implementiert. Neben dem Ellipsenmodell und der Strecke muss außerdem ein Wert für δ angegeben werden, welches eine Kontur fixiert.

Zur Kompensation der Winkelabhängigkeit wird das Koordinatensystem zunächst transformiert. Dazu wird es so rotiert, dass die Ellipse entlang der x-Achse orientiert ist. Anschließend wird die y-Achse um den Faktor α gestreckt, der durch (4.12) bestimmt wurde.

Nun kann das Integral aus Gleichung (4.6) bestimmt werden. Die Lösung dieses Integrals wird auf ein einfacheres zurückgeführt, welches die Entfernung zwischen einem festen Punkt und einem auf der x-Achse integriert:

$$\text{LPE}(\hat{x}, P) = \int_0^{\hat{x}} \sqrt{(x_P - x)^2 + y_P^2} dx \quad (5.4)$$

$$= \begin{cases} \left| \frac{\hat{x}^2}{2} - \hat{x}x_P \right|, & y_P = 0 \wedge x_P \notin [0, \hat{x}] \\ \hat{x}^2 - \hat{x}x_P + x_P^2, & y_P = 0 \wedge x_P \in [0, \hat{x}] \\ \frac{1}{2} \left[(x - x_P)d - y_P^2 \ln(d + x_P - x) \right]_{x=0}^{x=\hat{x}}, & y_P \neq 0 \end{cases} \quad (5.5)$$

Wobei $d = \sqrt{(x_P - x)^2 + y_P^2}$ und $P = (x_P, y_P)$.

Dazu muss das System abermals transformiert werden, sodass der Startpunkt der Strecke ins Zentrum rückt und an der x-Achse ausgerichtet ist. Im nächsten Schritt müssen die Schnittpunkte der x-Achse mit der Ellipse bestimmt werden, um so das Integral aufzuspalten zu können, um den Betrag der PFE-Definition aus (4.5) zu eliminieren.

Jetzt kann man stückweise das Integral zwischen den Schnitt und Endpunkten der Strecke berechnen. Die Berechnung eines Intervalls $[x_1, x_2]$ sieht dabei folgendermaßen aus:

$$D_1 = \text{LPE}(x_2 - x_1, F_1 - (x_1, 0)) \quad (5.6)$$

$$D_2 = \text{LPE}(x_2 - x_1, F_2 - (x_1, 0)) \quad (5.7)$$

$$\text{aLFE} \Big|_{\text{Intervall}} = |D_1 + D_2 - 2a(x_2 - x_1)| \quad (5.8)$$

Schlussendlich wird der aLFE-Wert aus der Summe aller Teilintegrale gebildet. Aus diesem lassen sich wiederum der rLFE-, naLFE- und nrLFE-Wert bestimmen.

5.3.6 filter_angle

Das Orientierungskriterium wird von der Funktion `filter_angle` überprüft. Wie auch bei der `line_rating`-Funktion, werden Ellipse, Strecke und Rotationsrichtung angegeben. Zusätzlich existieren zwei weitere Parameter τ und κ , welche im Orientierungskriterium in der Form von Abschnitt 4.3.4 nicht auftraten. τ gibt den maximalen Abstand zwischen α_1 und α_2 an, die in (4.13) und (4.14) definiert wurden. κ gibt einen Toleranzwinkel an, der in den Ungleichungen (4.16) und (4.17) überschritten werden darf.

Vor der Prüfung der Kriterien werden, wird das Koordinatensystem so verschoben und rotiert, sodass sich die Ellipse wie schon im Lösungskapitel im Ursprung befindet und an der x-Achse orientiert ist. Um spätere Fallunterscheidungen zu vermeiden, werden die Endpunkte der Strecken vertauscht, falls es sich um eine im Uhrzeigersinn rotierende Ellipse handelt. Somit müssen nur die folgenden Bedingungen geprüft werden:

$$(\alpha_2 - \alpha_1) \bmod 2\pi \leq \tau \quad (5.9)$$

$$(\alpha - \alpha_1 + \kappa) \bmod 2\pi \leq (\alpha_2 - \alpha_1 + 2\kappa) \bmod 2\pi \quad (5.10)$$

5.4 Verifikation

5.4.1 verify_models

Der letzte Schritt der Mittelkreiserkennung, die Verifikation, erledigt die `verify_models`-Funktion. Aus einer Menge von links- und rechtsläufiger Ketten, überprüft sie alle möglichen Paare darauf, ob sie die Eigenschaften eines Mittelkreises erfüllen. Die Implementierung richtet sich dabei stark an die Algorithmen 4.3 und 4.4.

Als Schwellwerte für das Auffinden von Paaren, welche die drei geometrischen Kriterien der Ähnlichkeit und Lage aus 4.4 erfüllen, wurden die Werte $r_{min} = 0.9$, $r_{max} = 1.1$ und $size_{max} = 1.25$ gewählt.

Bei der darauffolgenden Farbprüfung der Paare, wurde für das Abtasten eine Spaltengröße von 21 gewählt. Diese Zahl sollte durch drei teilbar sein und ihr Quotient ungerade. Somit ist gewährleistet, dass für alle drei dieselbe Anzahl von Pixeln zugeschrieben wird und eine mittlere Pixelreihe existiert.

Spalten, die Pixel enthalten, die nicht mehr innerhalb des Bildes liegen, werden ignoriert. Zum Weichzeichnen wird ein Gaußfilter eingesetzt. Die dafür berechnete Gaußglocke ist eine 5×5 -Matrix mit der Standardabweichung $\sigma = 2.5$.

Als Schwellwert für der Summe der Fehlerquadrate wird ein Wert von 0.03 verwendet. Bei einem konstanten Fehler entspräche das beispielsweise einer Abweichung von etwa 1.15 Pixeln von der Mitte.

Kapitel 6

Evaluierung

Zum Zweck der Evaluierung liegt eine Bibliothek von Aufnahmen der HTWK Leibzig vor, welche verschiedene Situationen aus der Standard Platform League darstellen. Sie umfasst 600 Fotos, auf denen unterschiedliche Bereiche des Spielfeldes aus mehreren Winkeln aufgenommen wurden. Darunter befinden sich aber auch Aufnahmen, welche das Spielfeld kaum oder gar nicht zeigen, sondern die Umgebung außerhalb. Die Auflösung der Aufnahmen beträgt 640×480 Pixel.

Aufgrund der Vielfältigkeit liegt der Mittelkreis häufig nicht oder nur unvollständig im Sichtfeld. Zudem wird er auf einem Teil der Bilder durch Hindernisse stückweise verdeckt. Daher wurden die Aufnahmen zur Auswertung nach der Sichtbarkeit des Mittelkreises durch die folgenden Kriterien kategorisiert:

Im Blickfeld Der Mittelkreis befindet sich zu etwa mindestens drei Viertel im Bildausschnitt (unabhängig von Verdeckungen).

Am Rand Der Mittelkreis befindet sich am Bildrand und ist zumindest zu einem Viertel sichtbar (unabhängig von Verdeckungen).

Verdeckung Ein oder mehrere Hindernisse verdecken Teile des Mittelkreises.

Aus diesen Kriterien entspringen fünf Kategorien:

- Im Blickfeld ohne Verdeckung
- Am Rand ohne Verdeckung
- Im Blickfeld mit Verdeckung
- Am Rand mit Verdeckung
- Nicht sichtbar

Die Evaluierung konzentriert sich vor allem auf die Umsetzung der Streckenselektion und Verifikation. Eine Auswertung des LSDs wird nicht vorgenommen, da dieser für

Schritt	Durchschnitt	Median	Minimum	Maximum
field_color_detection	51 ms	52 ms	19 ms	96 ms
sieve_segs	52 ms	48 ms	3 ms	201 ms
detect_chains	4 ms	3 ms	1 ms	22 ms
merge_chains	83 ms	50 ms	1 ms	728 ms
extend_chains	511 ms	10 ms	1 ms	61732 ms
verify_models	9 ms	2 ms	1 ms	663 ms
Gesamt	707 ms	195 ms	45 ms	62527 ms

Tabelle 6.1: Berechnungszeiten

die Verwendung nicht modifiziert wurde und damit die Ergebnisse seiner Autoren [1] weiterhin gelten.

6.1 Berechnungszeiten

Das Verfahren zur Mittelkreiserkennung teilt sich in mehreren Schritten, zu denen die Berechnungszeiten bestimmt worden sind. Zudem wurde auch die Gesamtzeit aus der Summe der Einzelschritte ermittelt. Die Ergebnisse dieser Messung sind in Tabelle 6.1 dargestellt.

Aus den Ergebnissen lassen sich zunächst zwei Erkenntnisse gewinnen. Erstens ist die Berechnungszeit stark abhängig vom analysierten Bild und zweitens liegt der Großteil des Berechnungsaufwands im Schritt `extend_chains`.

Die geringen minimalen Laufzeiten der Schritte 3-6 lassen sich dadurch erklären, dass manche Bilder nur wenig Strecken oder Ketten enthalten. In zwei Fällen sind nach dem zweiten Schritt sogar gar keine Strecken übrig geblieben, weshalb die verbleibenden Schritte überhaupt nicht ausgeführt wurden¹.

Die Berechnungsdauer des Schritts `merge_chains` ist stark von der Kettenanzahl abhängig, da für jedes Paar das RANSAC-Verfahren angewendet wird. `extend_chains` ist dagegen auch von der Anzahl der Strecken abhängig, die nicht Teil anderer Ketten sind. Da für jede Kette nach Kandidaten gesucht wird, kann die Laufzeit stark zunehmen, wenn viele Kandidaten untersucht werden müssen. Das Maximum von 61.7 s stellt dabei einen Extremwert dar. Schon der zweitschlechteste Wert gibt ‚nur‘ noch eine Dauer von 15.3 s an.

In Abbildung 6.1 ist das Foto dargestellt, welches die lange Berechnungszeit von etwa einer Minute verursacht. Nach rechts laufende Ketten sind blau und nach links lau-

¹Da in diesen Fällen keine Messungen vorlagen, wurden diese für die Schritte 3-6 nicht betrachtet. Darum ergibt die Summe der Durchschnitte auch nicht 707 ms.

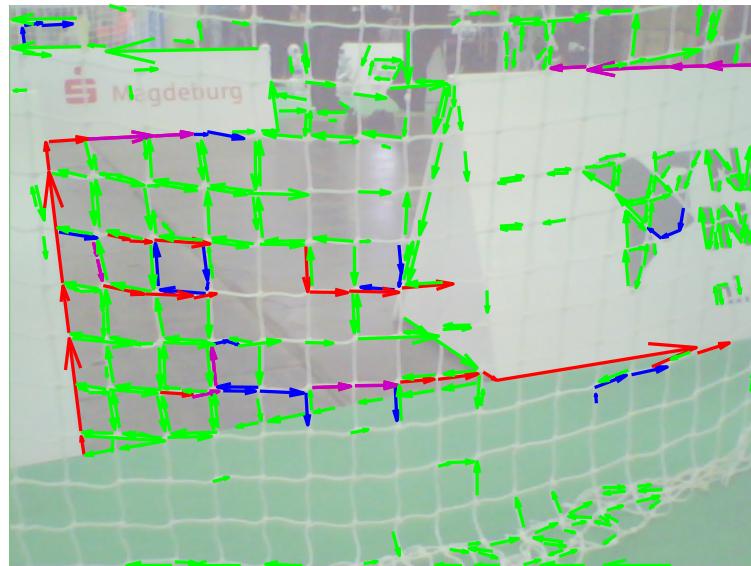


Abbildung 6.1: Verzögerung der Berechnung durch Tornetz

Kategorie	Anzahl	Erkannt	Anteil
Im Blickfeld ohne Verdeckung	48	40	83.3 %
Am Rand ohne Verdeckung	106	26	24.5 %
Im Blickfeld mit Verdeckung	10	3	30.0 %
Am Rand mit Verdeckung	32	8	25.0 %
Nicht sichtbar	404	0	–

Tabelle 6.2: Erkennungsraten

fende rot gefärbt. Strecken, die beiden Kettenarten angehören, sind violett, die übrigen grün. Das Problem liegt hier an der großen Anzahl von irrelevanten Strecken, die dennoch vom Algorithmus betrachtet werden. Zudem hätten viele Segmente schon beim initialen Filtern aussortiert werden müssen, die hier aber aufgrund einer fehlerhaften Farberkennung versagte.

6.2 Erkennungsraten

In Tabelle 6.2 sind die Erkennungsraten sortiert nach den oben eingeführten Kategorien angegeben. Abgesehen von der Kategorie „Nicht sichtbar“ ist der Mittelkreis in allen Kategorien zu sehen. Dabei trat kein Fall eines falsch erkannten Mittelkreises auf.

Die Ergebnisse der Einzelfälle lassen sich aufgrund des nicht-deterministischen RANSAC-Verfahren nur mit einer gewissen Wahrscheinlichkeit reproduzieren. Abhän-

gig davon, können manche Mittelkreise in unterschiedlichen Durchläufen erkannt werden oder unerkannt bleiben.

Erwartungsgemäß erzielt der Algorithmus das beste Ergebnis in der ersten Kategorie, wobei in 16.7 % der Fälle der Mittelkreis nicht erkannt werden konnte. Dabei handelt es sich um eine Schwäche des `merge_chains`-Schrittes, welches im nächsten Abschnitt (6.3) behandelt wird.

Ist der Mittelkreis zu weniger als drei Viertel sichtbar oder wird verdeckt, nehmen auch die Probleme mit der Erkennung weiter zu. So werden nur noch etwa ein Viertel bis ein Drittel aller Mittelkreise erkannt.

6.3 Problemfälle

Die große Anzahl an nicht erkannten Mittelkreisen lassen sich auf mehrere Probleme zurückführen. Diese liegen in den verschiedenen Phasen des Algorithmus. Einige davon möchte ich in den nächsten Abschnitten erläutern.

6.3.1 Fehlerhafte Feldfarbenerkennung

Die Erkennung der Feldfarbe ist für das initiale Filtern der Strecken und der abschließenden Verifikation von großer Bedeutung. Abbildung 6.2 zeigt zwei Fälle, in denen die Verwendete Lösung unzureichend für eine korrekte Erkennung ist. Links wurden zu viele Bereiche aufgrund eines allgemeinen Grünstichs als Feldfarbe identifiziert. Rechts dagegen führt eine Farbverschiebung im Zentrum des Bildes zu einem falschen Ergebnis, sodass das gesamte Feld nicht als grün erkannt wird.

6.3.2 Schlechte Annäherung durch Parameterschätzung

Grundlage für viele Berechnungen und Entscheidungen im Verlauf der Mittelkreiserkennung ist die Erzeugung eines Modells aus Streckendaten. Doch die dazu verwendete Methode ist manchmal zu ungenau. In Abbildung 6.3 sind zwei Beispiele zu sehen, bei denen die produzierte Ellipse zu schmal geraten ist und Abweichungen von den Strecken bzw. dem Mittelkreis zu sehen sind. Dies kann darauf zurückgeführt werden, dass den Strecken lediglich Punkte für die Ausgleichsrechnung entnommen werden. Daten über die Orientierung der Strecken werden dazu nicht herangezogen.



Abbildung 6.2: Feldfarbe nicht korrekt erkannt. Oben die Ausgangsbilder und unten die Bereiche der Feldfarbe in grün, andere Bereiche rot.

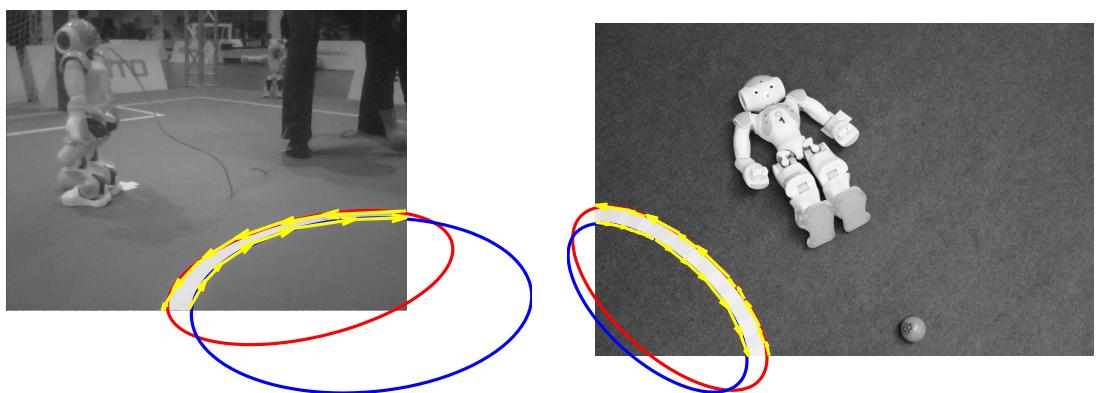


Abbildung 6.3: Schlechte Übereinstimmung zwischen Ellipsen und Strecken

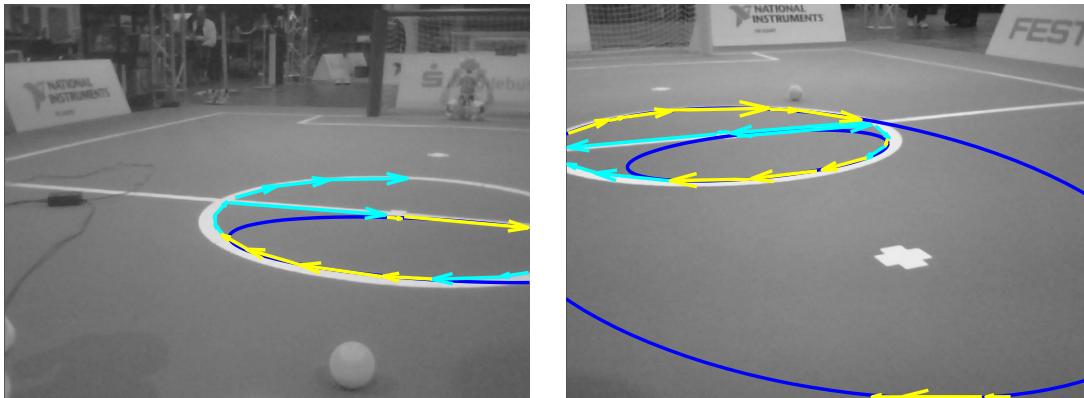


Abbildung 6.4: Falsche Zusammenführung von Ketten des Mittelkreises (gelb: Unterstützer, türkis: Keine Unterstützer bzw. Ausreißer)

6.3.3 Falsche Vereinigung von Ketten

In einigen Fällen werden Ketten des Mittelkreises mit den falschen Ketten zusammengeführt. Zwei Beispiele sind in Abbildung 6.4 zu sehen. In beiden Fällen wird der Innenrand des Kreises durch den Mittelkreis unterbrochen. Infolgedessen entstehen zwei getrennte Ketten, die zusammengeführt werden müssen.

Im linken Bild wird fälschlicherweise die untere Kette mit der Kette der Mittellinie vereinigt. Dabei entsteht eine blaue Ellipse deren Unterstützer in Gelb gekennzeichnet sind. Dies geschieht wegen des Umstands, dass die Reihenfolge, in der versucht wird Ketten zu verbinden, entscheidend ist. Die untere und mittlere Kette werden zusammengeführt noch bevor die obere Kette betrachtet wird. Die Reihenfolge wird durch die Anzahl der Kettenglieder bestimmt. Des Weiteren sorgt der RANSAC-Algorithmus dafür, dass vermeintlich Ausreißer entfernt werden. In diesem Fall führt dies jedoch zum Ausschluss von einigen korrekten Strecken.

Im rechten Bild gelingt es dem RANSAC-Algorithmus nicht drei Strecken aus der oberen und unteren Kette so zu selektieren, dass sich eine ausreichend große Unterstützermenge bildet. Darum wird die untere Kette mit keiner anderen Kette zusammengefügt. Stattdessen wird die obere Kette später mit einer Kette am unteren Bildrand vereinigt. Hier ist Nachbesserung am RANSAC-Schritt nötig.

6.3.4 Ausreißer nicht erkannt

In manchen Fällen werden Ausreißer nicht vom RANSAC-Verfahren erkannt. Im Beispiel des linken Bildes aus Abbildung 6.5 ist die Abweichung des Ausreißers (rot) zu gering, verhindert aber eine spätere Erweiterung der Kette. Im anderen Beispiel rechts

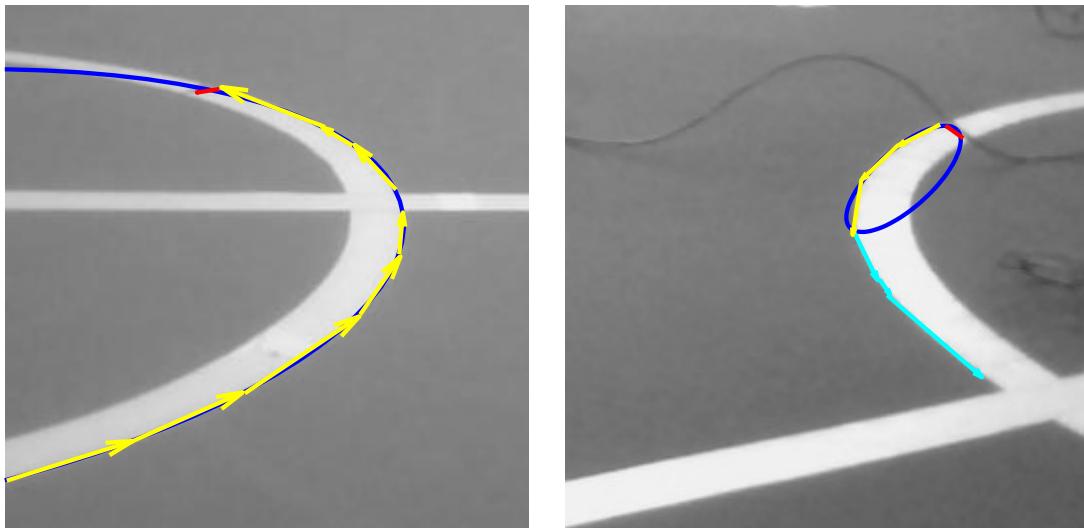


Abbildung 6.5: Ausreißer werden nicht erkannt (gelb: korrekt-positiv, türkis: falsch-negativ, rot: falsch-positiv)

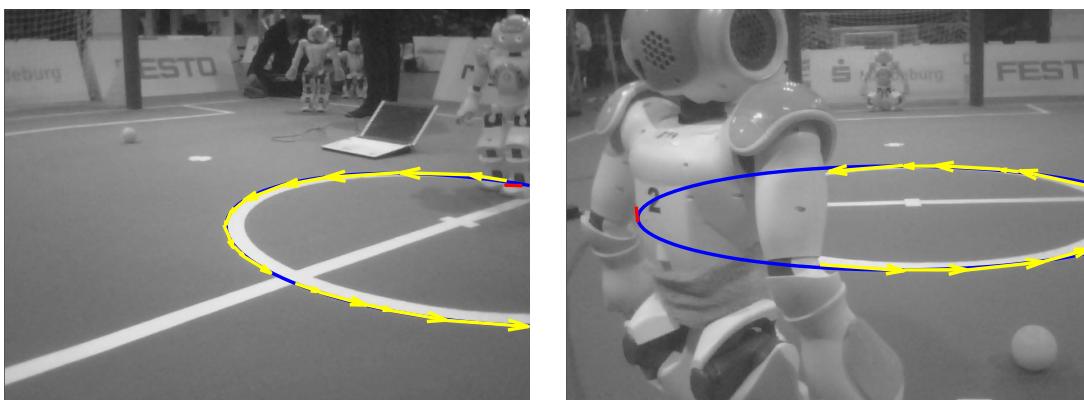


Abbildung 6.6: Ausreißer werden nicht erkannt (gelb: korrekt, rot: falsch)

werden sogar zwei korrekte Strecken zugunsten eines Ausreißers entfernt. Dieser Fall tritt jedoch nur mit einer bestimmten Wahrscheinlichkeit ein. In anderen Versuchen wird der Ausreißer durchaus entfernt.

6.3.5 Hinzufügen falscher Strecken zu einer Kette

Ähnlich wie in 6.3.4 können falsche Strecken (rot) auch einer Kette hinzugefügt werden, wenn sie zusammen mit den übrigen Strecken (gelb) eine gültige Ellipse hervorbringen. Dies führt in den Abbildungen aus 6.6 zu einer kleinen (links) und großen (rechts) Verfälschung der resultierenden Ellipse.

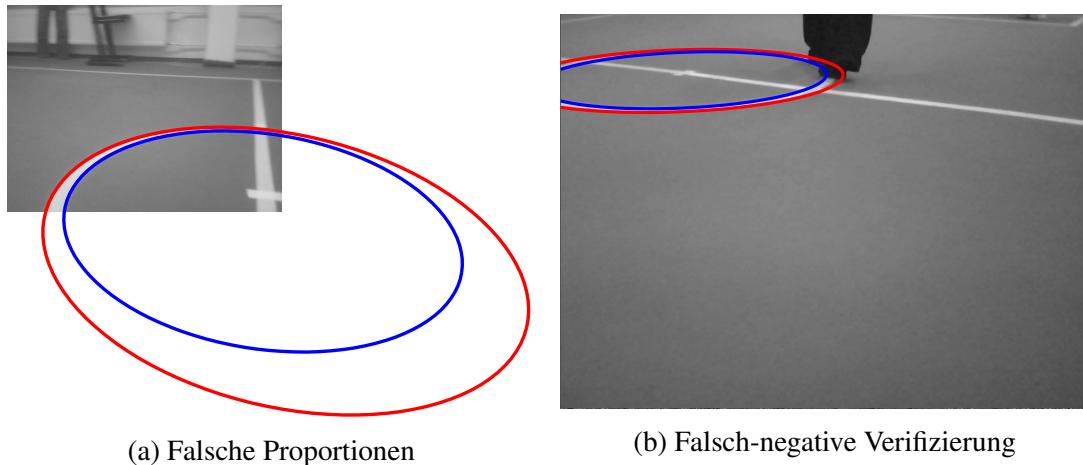


Abbildung 6.7: Verworfene Mittelkreise

6.3.6 Falsch-negative Verifikation

Im letzten Schritt, der Verifikation, wird entschieden, ob unter den gefundenen Ellipsen ein Paar existiert, welches zum Mittelkreis gehört. Während innerhalb der untersuchten Bibliothek keine falsch-positive Mittelkreise erkannt wurden, so kamen jedoch falsch-negative Fälle vor. Dies betrifft häufig jene Aufnahmen, welche nur einen kleinen Ausschnitt des Kreises darstellen wie im Beispiel 6.7 links. Dort ist die innere Ellipse gegenüber der größeren zu klein geraten.

Im zweiten Beispiel scheitert die Verifikation an der Verdeckung durch den Fuß. Bei der Überprüfung, ob sich im Bereich zwischen dem Ellipsenpaar die weiße Linie des Kreises befindet, führt der Fuß zu einer Störung. Die Phasen dieser Überprüfung sind in Abbildung 6.8 dargestellt. Im ersten Schritt wird die Umgebung des Ellipsenpaares abgetastet, sodass eine gerade gezogene Version des Kreises entsteht. Dabei wird anhand der Farbe bestimmt, welche Pixel zum grünen Feld gehören. Diese sind im zweiten Bild grün dargestellt, die anderen rot. Vor dem darauffolgenden Schritt werden alle Spalten verworfen die nicht in beiden Hälften mindestens einen grünen Pixel aufweisen. Diese sind im dritten Bild rot dargestellt. Daraus resultiert das vierte Bild, welches nun nicht mehr die verworfenen Spalten enthält. Das Problem hierbei ist, dass ein Teil des Fußes nicht entfernt wurde sondern links noch sichtbar ist. Deshalb entsteht eine große Abweichung der Position des Maximumwerts, woraufhin das Ellipsenpaar verworfen wird.

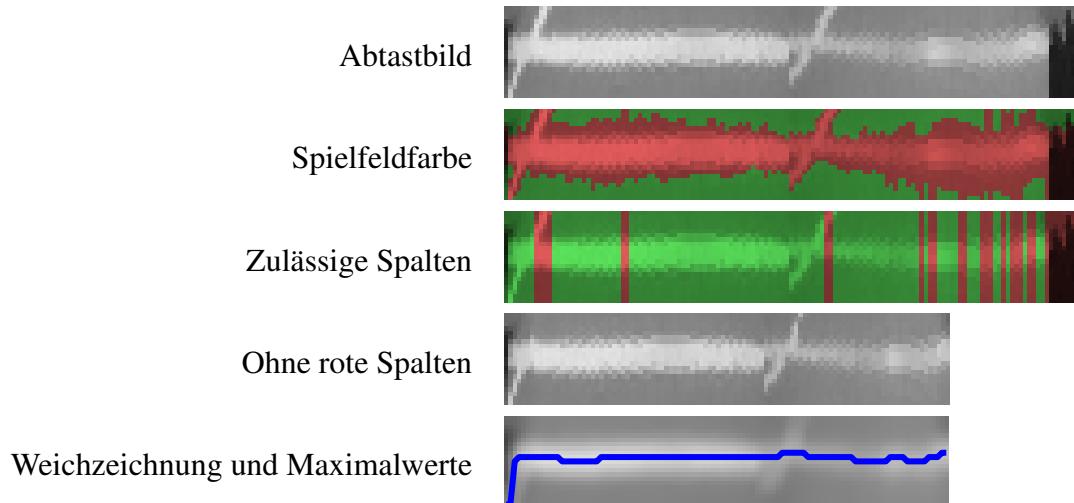


Abbildung 6.8: Überprüfung der weißen Linie des Mittelkreises

6.4 Erkannte Mittelkreise

Trotz der oben genannten Probleme werden viele Mittelkreise dennoch erkannt. Allerdings ist das Ergebnis der Ellipsenerkennung auch vom Zufall abhängig und kann daher von Versuch zu Versuch unterschiedlich ausgehen. Im Folgenden sind acht erfolgreiche Resultate der Mittelkreiserkennung aufgeführt.



Abbildung 6.9: Erkannte Mittelkreise

Kapitel 7

Fazit und Ausblick

Die Ergebnisse der Evaluierung führen zu dem Schluss, dass die hier entwickelte Mittelkreiserkennung noch einige Probleme aufweist, die es noch zu lösen gilt bevor es zu einem praktischen Einsatz im RoboCup kommt. Dennoch zeigen sie auch das Potential von Erkennungssystemen auf Basis von Liniensegmenten. Zum einen müssen die in Abschnitt 6.3 aufgezeigten Probleme beseitigt werden. Dies sollte in weiterführenden Arbeiten möglich sein, da zum Teil bereits Lösungswege existieren oder diese zumindest keine zu hohe Herausforderung darstellen sollten. Zum anderen konnte an einigen Beispielen gezeigt werden, dass selbst schwierige Fälle erkannt werden können.

Der hier entwickelte Ansatz, Strecken als Ausgangspunkt zur Detektion von Objekten zu nutzen, halte ich daher für vielversprechend. Sie bieten die Möglichkeit zur intuitiven Herangehensweise zur Lösung des Mittelkreisproblems, die ich bei der Entwicklung des Konzepts in Kapitel 4 genutzt habe.

Für einen praktischen Einsatz auf dem Feld ist die Implementierung in MATLAB vorerst nicht geeignet. Diese stellt zunächst nur einen Prototypen dar, auf dessen Grundlage weitere Umsetzungen realisiert werden können. Eine fortgeschrittene C++-Variante in Verbindung mit der OpenCV-Bibliothek ist beispielsweise denkbar.

Die Grundidee einer Liniensegmenterkennung zur Erkennung von Ellipsen zu verwenden ist nicht auf den Mittelkreis oder gar den RoboCup beschränkt. Mit wenigen Modifikationen ist sogar die hier vorgestellte Lösung dazu in der Lage Ellipsen auch außerhalb des RoboCups zu erkennen. Zwei Beispiele dafür sind in Abbildung 7.1 dargestellt. Zudem lassen sich auf dieser Basis auch Detektoren anderer Formen (z.B. Viercke) entwickeln.

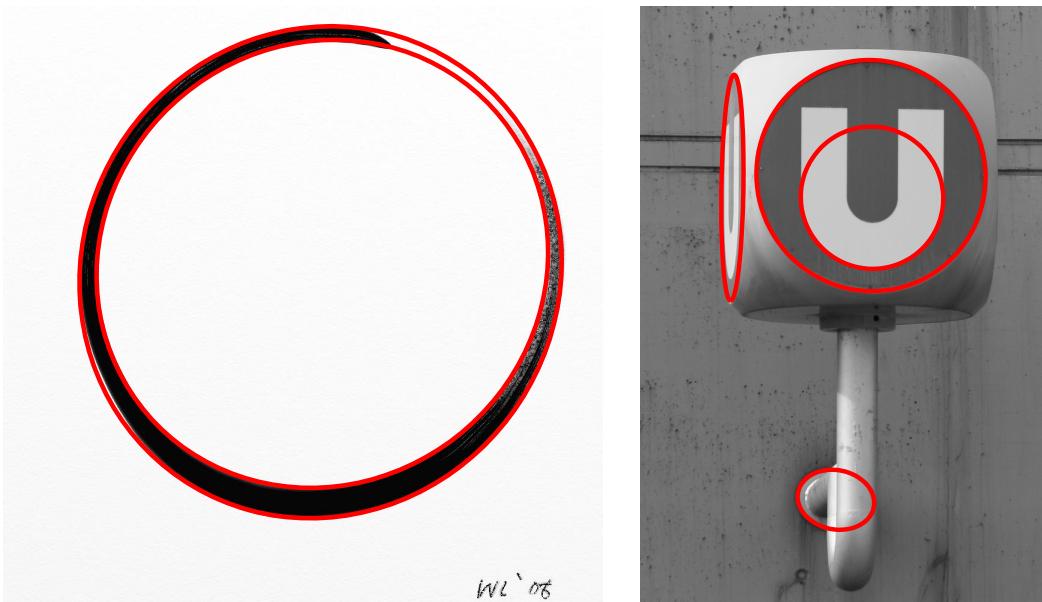


Abbildung 7.1: Anwendung außerhalb des RoboCups

Quelle links: Kobi Nistel <http://www.mathworks.com/matlabcentral/fileexchange/35223-circle-detection-using-hough-transforms/content/CircleFinder/circlefinder.m>

Quelle rechts: Wikipedia <http://commons.wikimedia.org/wiki/File:U-bahn-wien.jpg>

Literaturverzeichnis

- [1] VON GIOI, RAFAEL GROMPONE, JEREMIE JAKUBOWICZ, JEAN-MICHEL MOREL und GREGORY RANDALL: *LSD : A Line Segment Detector*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(4):1–10, 2010. III, 5, 48
- [2] ROBOCUP.ORG: *RoboCup Objective*. <http://www.robocup.org/about-robocup/objective/>, 3 2013. 1
- [3] KAUFMAN, RACHEL: *RoboCup 2010: Could Robot versus Human Be Far Behind?* Scientific American, 6 2010. 1
- [4] ROBOCUP: *Standard Platform League*. <http://www.tzi.de/spl/bin/view/Website/WebHome>, 3 2013. 2
- [5] COMMITTEE, ROBOCUP TECHNICAL: *RoboCup Standard Platform League (Nao) Rule Book*. <http://www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf>, 2 2013. 2, 4, 5
- [6] ROBOTICS, ALDEBARAN: *Datasheet NAO Next Gen – H21/H25 Model – English version*. Aldebaran Robotics. 2
- [7] CHIA, A.Y.-S., S. RAHARDJA, D. RAJAN und M. K H LEUNG: *Structural Descriptors for Category Level Object Detection*. Multimedia, IEEE Transactions on, 11(8):1407–1421, 2009. 2
- [8] CHIA, A.Y., S. RAHARDJA, D. RAJAN und KARHANG LEUNG: *Object recognition by discriminative combinations of line segments and ellipses*. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, Seiten 2225–2232, 2010. 2, 6
- [9] LIU, YANGXING, TAKESHI IKENAGA und SATOSHI GOTO: *Geometrical, Physical and Text/Symbol Analysis Based Approach of Traffic Sign Detection System*. IEICE Transactions, 90-D(1):208–216, 2007. 2

- [10] KUMAR, N., S. KOHLBECHER und E. SCHNEIDER: *A novel approach to video-based pupil tracking*. In: *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, Seiten 1255–1262, 2009. 2
- [11] BELL, A.A., G. HERBERICH, D. MEYER-EBRECHT, A. BOCKING und T. AACH: *Segmentation and Detection of Nuclei in Silver Stained Cell Specimens for Early Cancer Diagnosis*. In: *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, Band 6, Seiten VI – 49–VI – 52, 2007. 2
- [12] COMMITTEE, ROBOCUP TECHNICAL: *RoboCup Standard Platform League (Nao) Rule Book*. <http://www.tzi.de/spl/pub/Website/Downloads/Rules2012.pdf>, 5 2012. 4
- [13] GANDER, WALTER, GENE H GOLUB und ROLF STREBEL: *Least-squares fitting of circles and ellipses*. BIT, 34:558–578, 1994. 8
- [14] WONG, C.Y., S. C F LIN, T. R. REN und N. M. KWOK: *A survey on ellipse detection methods*. In: *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*, Seiten 1105–1110, 2012. 10
- [15] HOUGH, PAUL: *Method and Means for Recognizing Complex Patterns*. U.S. Patent 3.069.654, Dezember 1962. 10
- [16] XIE, YONGHONG und QIANG JI: *A new efficient ellipse detection method*. In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, Band 2, Seiten 957–960 vol.2, 2002. 10
- [17] FISCHLER, MARTIN A. und ROBERT C. BOLLES: *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. Commun. ACM, 24(6):381–395, Juni 1981. 24

Abbildungsverzeichnis

2.1	Spielfeld der Standard-Plattform-League	5
2.2	Resultat des LSDs	6
2.3	Gradient einer Kontur	6
2.4	Region Growing	6
2.5	Rechtecke der eingefärbten Regionen	7
2.6	Projektion eines Würfels	8
2.7	Kegelschnitte	9
3.1	Konturextraktion	12
3.2	Grüntöne des Spielfeldes	12
3.3	Linienfarbe	13
3.4	Projektion des Mittelkreises	13
3.5	Orientierung kurzer Strecken	14
3.6	Entfernung und Orientierung einer Linie zur Ellipse	15
3.7	Irrelevante und verfälschte Ellipsen	16
3.8	Verifizierung eines unvollständigen Mittelkreises	16
4.1	Abtastmuster zur Farbprüfung einer Kontur	19
4.2	Ketten aus Strecken	20
4.3	Abstand und Winkel zweier Strecken	20
4.4	Graphkonstruktion	21
4.5	Kettenunterbrechungen	22
4.6	Ketten, die eine Strecke der Mittellinie enthält	22
4.7	Verfälschung durch Ausreißer	22
4.8	Schatten von Ellipse und Kettengliedern	25
4.9	Veränderung der Ellipse durch Hinzufügen einer neuen Strecke	26
4.10	Krümmung einer Pixelregion und die dazu platzierte Strecke	27
4.11	Unterstützungsmenge eines Modells	28

4.12 PFE-Feld	30
4.13 Geometrische Distanz	30
4.14 Integration des PFE-Feldes entlang der Strecke $\overline{P_1P_2}$	30
4.15 Fixierung des Fehlers an den Punkte P und Q	32
4.16 PFE-Felder für unterschiedliche δ -Werte	32
4.17 Schnittpunkte der Geraden mit der Ellipse	33
4.18 Schritte der Abtastung	36
4.19 Abgetasteten Mittelkreise	36
5.1 Beziehung der MATLAB-Funktionen	39
6.1 Verzögerung der Berechnung durch Tornetz	49
6.2 Feldfarbe nicht korrekt erkannt	51
6.3 Schlechte Übereinstimmung zwischen Ellipsen und Strecken	51
6.4 Falsche Zusammenführung von Ketten des Mittelkreises	52
6.5 Ausreißer werden nicht erkannt	53
6.6 Ausreißer werden nicht erkannt	53
6.7 Verworfene Mittelkreise	54
6.8 Überprüfung der weißen Linie des Mittelkreises	55
6.9 Erkannte Mittelkreise	56
7.1 Anwendung außerhalb des RoboCups	58

Tabellenverzeichnis

5.1	MATLAB-Funktionen	38
6.1	Berechnungszeiten	48
6.2	Erkennungsraten	49

Abkürzungsverzeichnis

CCD	Charge-coupled Device – Lichtempfindliches Bauelement
CFE	Chain Focal-Point Error – Brennpunktfehler mehrerer Strecken
EPS	Encapsulated Postscript – Grafikformat
JPG	Joint Photographic Experts Group – Grafikformat
LFE	Line Focal-Point Error
LSD	Line Segment Detector
RANSAC	Random Sample Consensus
PFE	Point Focal-Point Error
PNG	Portable Network Graphics – Grafikformt
RGB	Additiver Farbraum der Grundfarben Rot, Grün und Blau
SVG	Scalable Vector Graphics – Grafikformat