Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3128

Balltracking im RoboCup mit der Log-Polar-Transformation

Florian Burger

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. P. Levi
Betreuer:	Andreas Koch
begonnen am:	10. Januar 2011
beendet am:	12. Juli 2011
CR-Klassifikation:	I.2.9, I.2.10, I.4.8

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Verfolgung der Trajektorie von Objekten, dem sogenannten *Tracking*, am Beispiel der Verfolgung von Bällen im *RoboCup*. An Informationen stehen dazu die Bilder einer am Roboter angebrachten 360°-Kamera zur Verfügung. Unter der Voraussetzung, dass die initiale Position des zu verfolgenden Balls bekannt ist, wird dessen weitere Bewegung in den folgenden Kamerabildern bestimmt. Die dabei zur Verfügung stehende Rechenzeit ist stark begrenzt. Um ein günstigeres Abtastverhalten zu erreichen, werden die Bilder im logarithmischen Polarkoordinatensystem (Log-polar) betrachtet. Dadurch werden, relativ zur betrachteten Bildposition, nahe Bereiche feiner abgetastet als weiter entfernte.

Mit Hilfe dieser Betrachtungsweise werden die Kanten des Balls bestimmt. Dazu werden Pixel als Ballpixel oder Nicht-Ballpixel klassifiziert. Diese Klassifizierung basiert auf einem parameterlosen System, dass die speziellen Bedingungen des *RoboCup* berücksichtigt.

Aus den bestimmten Kanten werden dann Größe und Position des Balls im aktuellen Kamerabild ermittelt. Für diesen Vorgang werden zwei mögliche Varianten in dieser Arbeit beschrieben und verglichen. Der eine Ansatz basiert darauf, dass der Rand eines Balls im Log-Polar-Bild genau dann eine Gerade darstellt, wenn der Pol dieses Bildes auf dem Mittelpunkt des Balls liegt. Das liegt daran, dass der Ball alle Winkel bis zu einem gewissen Abstand ausfüllt. Aus der Abweichung des aktuellen Bilds zu diesem Idealbild soll versucht werden, auf die Position des Ballmittelpunkts zu schließen. Der zweite Ansatz basiert auf der Methode der Summe der kleinsten Fehlerquadrate. Dabei werden aus jeweils drei Ballkantenpunkten Kreise bestimmt. Für jeden dieser Kreise wird anschließend die Summe der quadrierten Abstände der anderen Ballkantenpunkte von eben diesem Kreis bestimmt. Der Kreis mit der niedrigsten Summe bestimmt dann die Parameter des Balls.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Überblick

Dieses Dokument beschreibt die Entwicklung eines Tracking-Verfahrens für Bälle zum Einsatz im *RoboCup*. Das Dokument ist in 7 Abschnitte unterteilt.

- Erster Abschnitt: Kurzüberblick.
- Zweiter Abschnitt: Einleitung in das behandelte Thema.
- Dritter Abschnitt: Erläuterung der Grundlagen, die für das restliche Dokument notwendig sind.
- Vierter Abschnitt: Motivation, das hier behandelte Verfahren zu entwickeln, und Beschreibung der Anforderungen, die an dieses gestellt werden.
- Fünfter Abschnitt: Vorstellung des entwickelten Verfahrens inklusive verschiedener Varianten für Teilprobleme und deren Abwägung gegeneinander.
- Sechster Abschnitt: Ergebnisse der Messungen und die Leistungen des Verfahrens in der Praxis.
- Siebter Abschnitt: abschließendes Fazit und Ausblick.

2. Einleitung

Für die Menschheit hat die visuelle Wahrnehmung der Umwelt von jeher große Bedeutung. Für einen Menschen sind seine Augen sein wichtigstes Sinnesorgan. Es ist erstaunlich, was das menschliche Auge zu leisten in der Lage ist. Mindestens ebenso erstaunlich sind die Leistungen des Gehirns, das die Bilder der Augen verarbeitet. Räumliches Sehen ist nur eine der faszinierenden Fähigkeiten, die uns diese Bildverarbeitung ermöglicht.

Mit dem Aufkommen der Kamera- und Computertechnologie erwuchs dementsprechend früh der Wunsch, auch Computern solche Fähigkeiten beizubringen. Dazu genügt es nicht, Kameras zu bauen, die entsprechende Bilder liefern. Mindestens ebenso groß ist die Herausforderung, den Computer diese Bilder verarbeiten zu lassen. Diese enthalten eine Vielzahl von Informationen, die mit Blick auf bestimmte Ziele ausgewertet werden müssen. Oft ist dazu auch ein Filtern dieser Informationen, sprich des Bildes, notwendig. Mit dem Filtern sollen die Informationen hervorgehoben werden, die für die zu erfüllende Aufgabe wesentlich sind, oder umgekehrt nicht benötigte Informationen entfernt werden.

Eine dieser möglichen Aufgaben ist die Verfolgung eines Objekts. Dieser Vorgang wird von uns Menschen täglich durchgeführt. Will man zum Beispiel ein Objekt fangen, das einem zugeworfen wird, so muss das Gehirn die Flugbahn dieses Objekts verfolgen und seine zukünftige Position extrapolieren, um die Hände in die richtige Fangposition zu bringen. Ein anderes Beispiel ist der Straßenverkehr, auch dort müssen ständig Objekte, sprich andere Fahrzeuge oder Verkehrsteilnehmer, verfolgt werden, um Kollisionen zu vermeiden. Allgemein lässt sich sagen, dass die Fähigkeit zur Objektverfolgung in sehr vielen verschiedenen Situationen relevant für einen Menschen ist.

Dementsprechend handelt es sich dabei um eine Aufgabe, die auch in der Bildverarbeitung bei Computern eine wichtige Rolle spielt. Das Verfolgen eines Objektes bezeichnet man hier meist mit dem englischen Begriff *Tracking*. Zentral ist diese Fähigkeit, wie auch das oben genannte Straßenverkehrsbeispiel zeigt, vor allem zur Vermeidung von Kollisionen. Daher hat sie vor allem für mobile Roboter große Bedeutung. Natürlich gibt es aber auch dort Einsatzbereiche für die Objektverfolgung, die über die Kollisionsvermeidung hinausgehen.

Ein aktives Forschungsgebiet in der Robotik ist der Roboterfußball, da es dort mit dem *RoboCup*[1] eine zentrale Organisation zur Veranstaltung von Wettbewerben gibt. Außerdem handelt es sich beim Fußball um ein Szenario, das viele verschiedene Aufgaben der Robotik kombiniert, unter anderem komplexe Bewegungssteuerung, Koordination mehrerer Roboter, Planung und natürlich Bildverarbeitung. Eine der Aufgaben der Bildverarbeitung beim Roboterfußball ist das Verfolgen des Spielballs. Eine genaue Bestimmung der Ballposition ist eine wichtige Voraussetzung für das Gewinnen eines Spiels.

Diese Positionsbestimmung muss auf Grundlage der Bilder der Kamera der Fußballroboter durchgeführt werden. Da Ball und Roboter auf dem Spielfeld beliebig zueinander orientiert sein können, verwendet man hier Kameras, die in der Lage sind, das vollständige Spielfeld, also einen Winkel von 360°, sehen zu können. Das Bild einer solchen Kamera eines Fußballroboters ist in Abbildung 1 zu sehen. Um die notwendige Bandbreite zur

[1] Siehe Unterabschnitt A.1 auf Seite 58

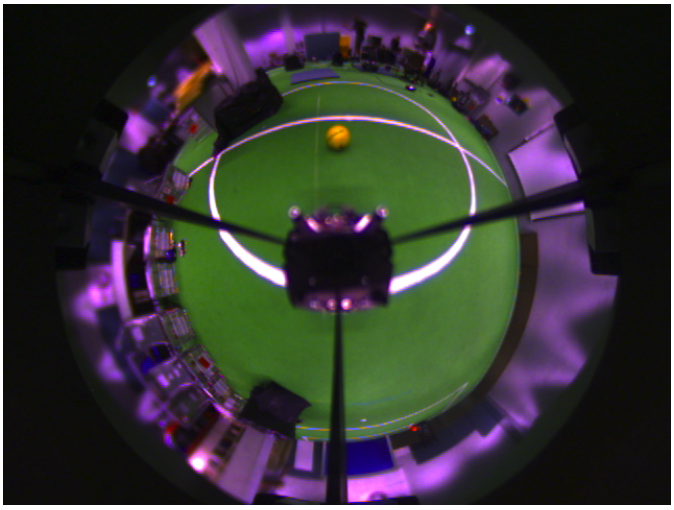


Abbildung 1: Ein Bild der 360°-Kamera eines Fußballroboters. Weitere Informationen zu dieser Art von Kameras sind in Unterabschnitt A.5 auf Seite 61 zu finden.

Übertragung der Bilder einer Kamera zum Roboter gering zu halten, werden die Bilder in der so genannten „4:2:2“-Kodierung des YUV-Farbmodells[2] geliefert.

Eine wichtige Anforderung an ein Verfahren, das die Verfolgung des Spielballs über mehrere Kamerabilder hinweg ermöglicht, ist geringe Laufzeit. Gewöhnlich liefern die verwendeten Kameras 30 Bilder pro Sekunde. Das bedeutet, dass für die gesamte Verarbeitung eines Bildes maximal $\frac{1}{30}$ Sekunde zur Verfügung steht. In dieser Zeit müssen aber noch wesentlich mehr Bildverarbeitungsaufgaben durchgeführt werden, zum Beispiel muss sich der Roboter lokalisieren oder andere Roboter erkennen. Dadurch verringert sich die tatsächlich für das Tracking des Balls zur Verfügung stehende Zeit weiter.

Um die notwendige Geschwindigkeit zu erreichen, muss die Abtastung des Bildes beschleunigt werden. Um das zu erreichen, kann man dieses nicht Pixel für Pixel abtasten, sondern es in Log-Polar-Koordinaten überführen und dann nur bestimmte Pixel betrachten. Diese Transformation und ihre Eigenschaften werden im folgenden Abschnitt 3 genauer beschrieben.

Gegenstand dieser Arbeit ist die Entwicklung und Evaluierung eines Verfahrens, das die Verfolgung eines Spielballs mit Hilfe der Log-Polar-Transformation realisiert.

2.1. Verwandte Arbeiten

Die Verwendung der Log-Polar-Transformation in der Bildverarbeitung war bereits Thema vieler Arbeiten. Bereits 1990 schreiben C. F. Weiman und R. D. Juday über deren Einsatz[12]. Sie stellen ein Verfahren vor, mit dem das Tracking von Raumfahrzeugen ermöglicht wird, um deren Andockvorgang zu unterstützen. Die Arbeit kommt zum Ergeb-

[2]Siehe Unterabschnitt A.4 auf Seite 60

nis, dass die Transformation in Log-Polar-Koordinaten hilft, Laufzeit und Speicherbedarf zu senken.

K. Daniilidis schreibt 1995 über die Verwendung der Transformation zur Nachbildung der Auflösungsverringerung des menschlichen Auges am Rand des wahrgenommenen Bildes[5]. Basierend darauf werden in dieser Arbeit weitere günstige Eigenschaften der Log-Polar-Transformation vorgestellt. Anschließend werden die gewonnenen Erkenntnisse zur Berechnung der Eigenbewegung und zur Kollisionsberechnung bei einem Roboter verwendet.

Die dynamische Fokussierung von Bildbereichen bei Stereo-Kamera-Robotern ist das Thema der Arbeit von C. Capurro et al. von 1997[3]. Sie erörtern, warum die Verwendung von Log-Polar-Bildern in diesem Fall vorteilhaft ist und stellen ein Vergenz-Kontrollsystem vor, dass auf diesen Bildern basiert.

Die Verwendung der Log-Polar-Transformation zur Gesichtserkennung stellen K. Hotta et al. in ihrer Arbeit aus dem Jahre 1998 vor[6]. Das Log-Polar-Bild wird dazu zeilenweise nach skalierungs- und verschiebungsinvarianten Merkmalen durchsucht, die dann zur Erkennung von menschlichen Gesichtern genutzt werden.

Ein Feld der Bildverarbeitung, in dem ebenfalls gern auf die Eigenschaften der Log-Polar-Koordinaten zurückgegriffen wird, ist die Registrierung mehrerer Bilder zueinander. Das beschreiben zum Beispiel G. Wolberg und S. Zokai in ihrer Arbeit aus dem Jahr 2000[14]. Sie stellen ein Registrierungsverfahren vor, dass Skalierung und Rotation von mehreren Bildern zueinander erkennt. Dieses arbeitet mit Log-Polar-Bildern.

Dem Tracking von Objekten im Bild einer beweglichen Kamera unter Zuhilfenahme der Log-Polar-Koordinaten widmen sich N. Okajima et al. im Jahr 2000[8]. Durch Untersuchung der Phasendifferenz von komplexen Wavelet-Transformationen der Bildersequenzen ermitteln sie die Bewegung des Objekts. Ziel ist außerdem, möglichst wenig Laufzeit zu benötigen, um möglichst viele Bilder pro Sekunde verarbeiten zu können.

Den Ansatz der Erfassung und Schätzung von Translationsbewegungen mit der Log-Polar-Transformation untersuchen V. Javier Traver und Filiberto Pla in ihrer Arbeit aus dem Jahre 2001[11]. Dazu verwenden sie ein gradientenbasiertes Minimierungsverfahren.

Il Choi et al. beschäftigen sich 2003 mit einem ähnlichen Thema[4] wie bereits N. Okajima et al.[8]. Ihre Arbeit behandelt das Tracking von sich bewegenden Objekten mit einem Stereo-Kamera-System. Dabei vergleichen sie den Einsatz der Log-Polar-Transformation mit anderen Ansätzen und stellen dessen Überlegenheit fest.

Speziell auf die Bewegungssteuerung eines Roboterkopfes zum Zweck des Trackings von farbigen Objekten gehen G. Metta et al. in ihrem Artikel aus dem Jahre 2004 ein[7]. Basierend auf Log-Polar-Bildern wird die Bewegung des Roboterkopfes bestimmt, die zu vollziehen ist, um das Kamerasystem des Roboters auf das zu verfolgende Objekt auszurichten. Sie stellen außerdem ein Lernverfahren vor, bei dem die Parameter dieses Steuerungssystems selbstüberwacht vom Roboter gelernt werden.

In seiner 2005 verfassten Masterthesis stellt Saikiran S. Thunuguntla ein Objekt-Tracking-Verfahren basierend auf der Log-Polar-Transformation vor[10]. Dieses basiert auf einem Template-Matching-Ansatz und verwendet die Transformation insbesondere dazu, um dieses Matching gegenüber Skalierung und Rotation des zu verfolgenden Objekts robust zu machen.

3. Grundlagen

3.1. Die Log-Polar-Transformation

Logarithmische Polarkoordinaten bauen auf gewöhnlichen Polarkoordinaten auf. Daher empfiehlt sich, zuerst zu erklären, was ein polares Koordinatensystem ist und wie dieses in der Bildverarbeitung verwendet wird.

Polarkoordinaten In einem Polarkoordinatensystem wird die Position eines Punktes auf einer zweidimensionalen Ebene nicht durch zwei orthogonale Vektoren bestimmter Länge, wie in einem kartesischen Koordinatensystem, sondern durch einen Winkel und einen Abstand definiert. Der Abstand, normalerweise mit r bezeichnet, gibt dabei die Entfernung des Punktes vom Ursprung, im Polarkoordinatensystem als *Pol* bezeichnet, wieder. Die Winkelkoordinate, meist mit θ oder ϕ bezeichnet, ist definiert als der Winkel, der von der Polarachse (auch: Null-Grad-Linie) gegen den Uhrzeigersinn anliegt. Statt Abstands- und Winkelkoordinate spricht man auch von Radius und Azimut oder Azimutwinkel. Abbildung 2 zeigt eine Skizze dazu.

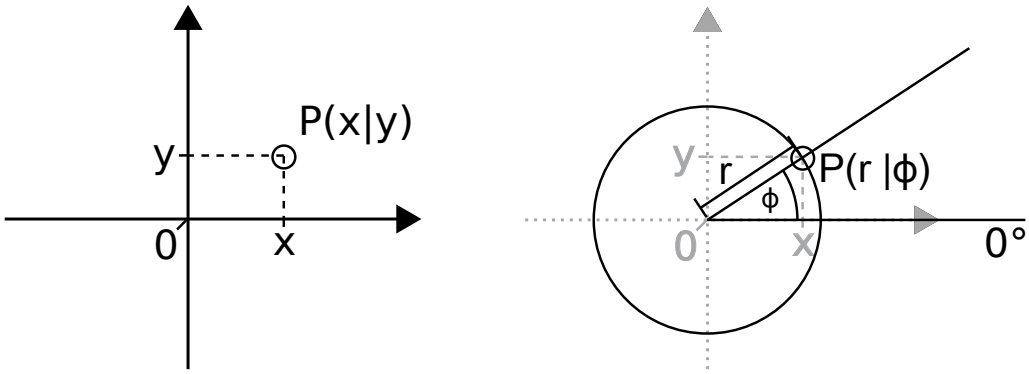


Abbildung 2: Links ist ein Punkt in einem kartesischen Koordinatensystem zu sehen. Rechts der selbe Punkt in Polarkoordinaten.

Das Koordinatenpaar bestimmt einen Punkt in einer zweidimensionalen Ebene unter zwei Voraussetzungen eindeutig: Erstens muss der Azimutwinkel auf ein definiertes Intervall, beispielsweise $[0, 2\pi)$, beschränkt werden. Ansonsten könnte jeder Punkt beliebig viele verschiedene Werte als Winkelkoordinate haben, die sich jeweils um eine volle Periode von 2π unterscheiden. Zweitens muss dem Pol die Azimutkoordinate $\phi = 0$ zugewiesen werden, da diese sonst für $r = 0$ beliebig wäre. Unter diesen Voraussetzungen ist eine Umrechnung von Polarkoordinaten in kartesische Koordinaten problemlos möglich. Dies ist für die Verwendung von Polarkoordinaten in der Bildverarbeitung wichtig, da Datenstrukturen für Bilder üblicherweise mit kartesischen Koordinaten adressiert werden. Liegt die Polarachse gleich mit der x-Achse des kartesischen Koordinatensystems, so ergeben sich folgende Umrechnungsformeln, um Polarkoordinaten in kartesische Ko-

ordinaten umzurechnen:

$$x = r \cdot \cos(\phi) \tag{1}$$
$$y = r \cdot \sin(\phi) \tag{2}$$

Die Umrechnung von kartesischen Koordinaten in Polarkoordinaten gestaltet sich wegen der Berechnung des Azimutwinkels etwas aufwendiger. Der Radius r entspricht hingegen einfach der Länge des x-y-Vektors:

$$r = \sqrt{x^2 + y^2} \tag{3}$$

Die Berechnung des Winkels erfolgt über die Arkustangens-Funktion. Da diese jedoch auf den Wertebereich $(-\frac{\pi}{2}, \frac{\pi}{2})$ beschränkt ist, ist hierfür eine erweiterte Arkustangensfunktion notwendig, die, abhängig von den Werten von x und y, gewisse Fälle unterscheidet (je nach Quadrant, in dem sich der umzuwandelnde Punkt befindet), und damit auf den Wertebereich $[0, 2\pi)$ erweitert wird. Da diese Umwandlung für diese Arbeit keine Rolle spielt, wird diese, auch als $atan2$ bezeichnete, Funktion hier nicht wiedergegeben.

Polarkoordinaten in der Bildverarbeitung Polarkoordinaten sind für die Bildverarbeitung interessant, da sich damit bestimmte Abtastverfahren eines Bildes leichter beschreiben lassen. Wenn von einem zentralen Punkt aus entlang von von diesem ausgehenden Linien ein Bild abgetastet werden soll, lassen sich diese Positionen hervorragend durch Polarkoordinaten ausdrücken. Der Azimutwinkel wird dann immer um den gewünschten Abstand zwischen zwei Abtastlinien erhöht, der Radius einfach hochgezählt. Diese Art der Abtastung ist besonders bei der Objekterkennung und -verfolgung interessant, da so nicht alle Pixel des Bildes betrachtet werden müssen und die Auflösung der Abtastung mit steigendem Abstand vom zentralen Punkt sinkt. So kann das Bild auf der Suche nach einem zu erkennenden Objekt in größeren radialen Abschnitten untersucht werden, und nur dort, wo potentielle Objektpositionen gefunden werden, wird die Suche vertieft.

Abbildung 3: Ein Beispielbild in Originalform (links) und in Polarkoordinaten (rechts). Die Position des Pols des Polarkoordinatenbilds ist mit einem Kreuz im Originalbild markiert.

Logarithmieren der Entfernungen Der einzige Unterschied von Log-Polar-Koordinaten zu gewöhnlichen Polarkoordinaten besteht darin, dass die Abstandskoordinate nicht mehr dem Radius selbst, sondern dessen natürlichem Logarithmus entspricht:

$$\rho = \ln \sqrt{x^2 + y^2} \tag{4}$$

Teilweise wird auch noch ein Skalierungsfaktor M verwendet, beispielsweise in OpenCV[13, S. 277ff]:

$$\rho' = M \cdot \ln \sqrt{x^2 + y^2} \tag{5}$$

Der Azimutwinkel berechnet sich genau wie bei den gewöhnlichen Polarkoordinaten. Für die umgekehrte Transformation, also die Umwandlung von log-polar Koordinaten in kartesische Koordinaten, gilt:

$$x = e^{\rho} \cdot \cos(\phi) \tag{6}$$
$$y = e^{\rho} \cdot \sin(\phi) \tag{7}$$

Mit Skalierungsfaktor M gilt:

$$x = e^{\frac{\rho'}{M}} \cos(\phi) \tag{8}$$
$$y = e^{\frac{\rho'}{M}} \sin(\phi) \tag{9}$$

Mit steigendem Abstand vom zentralen Punkt sinkt also bei der Verwendung von Log-Polar-Koordinaten in der Bildverarbeitung die Abtastrate. Nahe Bereiche sind hingegen höher aufgelöst. Dieses Verhalten wird von der Logarithmierung des Abstandes verursacht. Abbildung 4 verdeutlicht dies an einem Beispielbild.

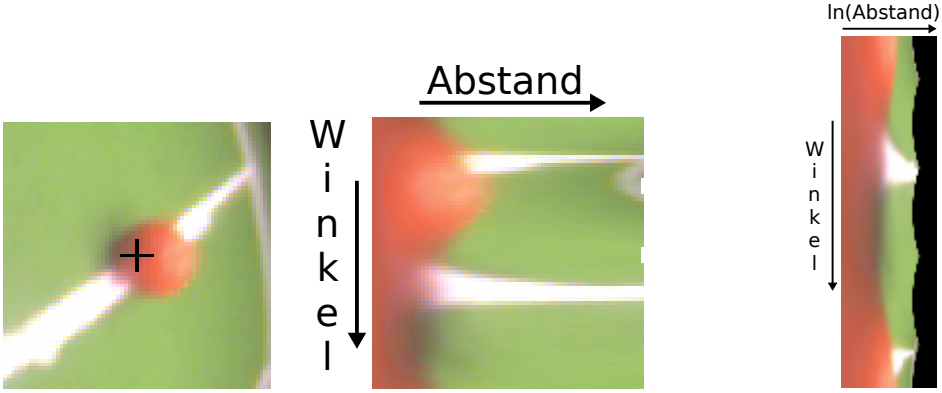


Abbildung 4: Ein Beispielbild in Originalform (links), Polarkoordinaten (Mitte) und in Log-Polar-Darstellung (rechts, Skalierungsfaktor $M = 10$)

3.2. Weitere Grundlagen

Die Erläuterungen zu weiteren Grundlagen sind bei Bedarf im Anhang zu finden. Im Einzelnen finden sich Erläuterungen zum *RoboCup* in Unterabschnitt A.1 auf Seite 58, zur *OpenCV*-Bibliothek in Unterabschnitt A.2 auf Seite 59, zu *Look-Up-Tables* in Unterabschnitt A.3 auf Seite 60, zum YUV-Farbmodell in Unterabschnitt A.4 auf Seite 60 und zu omnidirektionalen Kameras in Unterabschnitt A.5 auf Seite 61.

4. Motivation und Anforderungen

4.1. Tracking des Balls

Um im *RoboCup* erfolgreich sein zu können, ist eine gute und schnelle Erkennung des Spielballs ausgesprochen wichtig. Nur mit einer genauen Lokalisierung des Balls sind die Planungen von Angriffs- oder Verteidigungsstrategien möglich. Auch für die Berechnung der Bewegungssteuerung ist es erforderlich, die genaue Ballposition zu kennen: Der Roboter muss den Ball im richtigen Winkel anfahren, um ihn beispielsweise in sein *Dribbling-Device*[3] zu legen und sich dann mit dem Ball fortbewegen zu können. Für den Torwart ist eine genaue Erkennung des Balls ebenfalls von großer Bedeutung, damit er sich richtig positionieren kann, um gegnerische Angriffe abzuwehren.

Als Tracking bezeichnet man dabei den Vorgang, einen Ball über mehrere Kamerabilder hinweg zu verfolgen. Dazu steht zunächst eine Initialposition zur Verfügung, die von einem globalen Ballsuchverfahren geliefert wird. Natürlich können dabei auch mehrere potentielle Ballpositionen gefunden werden, die dann unabhängig voneinander ausgewertet und, falls ein Ball erkannt wurde, verfolgt werden. Für das Tracking selbst macht dies keinen Unterschied, es wird mehrfach für verschiedene Positionen ausgelöst. Das globale Verfahren wird nicht mehr benötigt, so bald ein Tracking-Vorgang aktiv ist. Stattdessen bekommt das Trackingverfahren als Initialposition im nächsten Schritt die Position des Balls aus dem aktuellen Trackingschritt.

Da es sich beim *RoboCup* um ein dynamisches Szenario handelt, soll das Tracking mit möglichst wenig Rechenzeit realisierbar sein. Nach einer einmaligen globalen Suche, um den Ball initial zu erfassen, muss das Trackingverfahren diesen im jeweils nächsten Bild der Kamera des Roboters in kurzer Zeit lokalisieren. Ausgangspunkt ist dabei die Position des Balls im vorherigen Bild. Als Vorgabe wurde gegeben, dass das Tracking nicht länger als etwa zwei Millisekunden auf der aktuellen *RoboCup*-Robotergeneration dauern soll.

Ein Problem beim Tracking ist die Verdeckung des Balls durch andere Roboter oder sogar den Bezugsroboter selbst, zum Beispiel dann, wenn der Ball im Kamerabild unter den Halterungen des Kameraspiegels verschwindet. Dies ist in Abbildung 5 zu sehen. Das Trackingverfahren muss daher in der Lage sein, den Ball auch dann korrekt zu erkennen, wenn er teilweise verdeckt ist.

Für das *RoboCup*-Szenario gelten gewisse Voraussetzungen, die beim Tracking hilfreich sind. Die Farbe des „Rasens“ ist vorgegeben und auf dem Spielfeld dürfen sich nur die anderen Roboter und ein Ball befinden. Der Ball selbst hat eine kräftige Farbe, meist gelb, und ist so gut von allen anderen Objekten zu unterscheiden. Da dies jedoch nur für Spielsituationen selbst gilt, das Trackingverfahren aber auch in Trainings- oder Testsituationen funktionieren soll, muss es weiteren Hindernissen oder mehreren Bällen gegenüber robust sein.

Um Fehldetektionen auszuschließen, muss das Trackingverfahren Informationen ausgeben, auf deren Basis sich eine Güte des detektierten Balls bestimmen lässt. Da eine globale Ballsuche möglicherweise mehrere potentielle Ballpositionen liefert, muss daraus

[3] Aussparung oder Vorrichtung am Roboter, in die der Ball zum Dribbeln gelegt werden soll

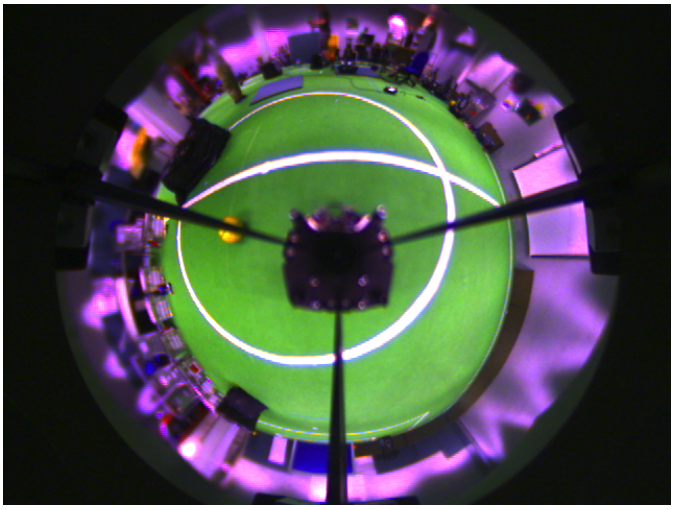


Abbildung 5: Der Ball wird von einer Halterung des Kameraspiegels teilweise verdeckt

die beste und somit wahrscheinlichste Hypothese gewählt werden.

Das Trackingverfahren muss außerdem deterministisch arbeiten: Bei gleichem Eingabebild müssen auch immer identische Ballposition und -größe ermittelt werden. Die stabile Detektion eines ruhig liegenden Balls muss außerdem robust gegenüber dem unvermeidlichen Rauschen im Kamerabild sein, damit nicht fälschlicherweise eine Bewegung erkannt wird.

Eine weitere Anforderung an ein Trackingverfahren ist seine Skalierbarkeit: Steht zukünftig mehr Rechenleistung zur Verfügung, soll es möglich sein, durch Parametereinstellung die Güte der Ergebnisse zu verbessern, da ja dann mehr Rechenzeit aufgewendet werden kann.

4.2. Verwendung der Log-Polar-Transformation

Die Log-Polar-Transformation ist unter zwei Gesichtspunkten für das Tracking interessant. Zum einen durch das Auflösungsverhalten, das sich durch diese ergibt. Dieses ist für ein Trackingverfahren potentiell nützlich. Zum anderen lassen sich aus der Darstellung eines Balles im Log-Polar-Bild Rückschlüsse auf dessen Position ziehen.

Auflösungsverhalten Wie im vorangegangenen Unterabschnitt 4.1, Tracking des Balls, erwähnt, ist hohe Geschwindigkeit für das Trackingverfahren sehr wichtig. Eine Möglichkeit, die Geschwindigkeit eines solchen Verfahrens zu steigern, ist die Anzahl der abgetasteten Punkte zu verringern, also nicht alle Punkte eines Bildes abzutasten. Wie in Unterabschnitt 3.1, Die Log-Polar-Transformation, auf Seite 10, beschrieben, hat die Log-Polar-Transformation hier gleich zwei günstige Eigenschaften:

Strahlenförmige Abtastung. Bei Log-Polar-Koordinaten, wie auch bei gewöhnlichen Po-

larkoordinaten, kann ein Bild strahlenförmig von einem zentralen Punkt ausgehend abgetastet werden, indem jeweils über die Winkel- und Radiuskoordinate iteriert wird. Dadurch wird die Anzahl der abgetasteten Pixel, je nach Schrittweite, erheblich reduziert, obwohl alle Richtungen trotzdem gleich gut berücksichtigt werden.

Auflösungsverhalten entlang der Strahlen. Durch die Logarithmierung des Radius' bei Log-Polar-Koordinaten wird außerdem die Auflösung auch entlang der einzelnen Strahlen mit steigendem Abstand reduziert. Bei Polarkoordinaten würde jedes Pixel entlang eines Strahls betrachtet, bei Log-Polar-Koordinaten hingegen – mit Skalierungsfaktor $M = 1$ – nur das erste ($e^0 = 1$), das dritte ($e^1 \approx 2,718$), das siebte ($e^2 \approx 7,389$) und so weiter.

Balldarstellung im Log-Polar-Bild Interessant für das Tracking ist außerdem das Aussehen eines Balls im Log-Polar-Bild. Liegt der Mittelpunkt eines Balls auf dem zentralen Punkt eines Log-Polar-Bildes, also dem *Pol*, so ergibt sein Rand eine gerade Linie, da alle Punkte des Randes den gleichen Abstand r_{Ball} vom Mittelpunkt haben. Abbildung 6 zeigt das.

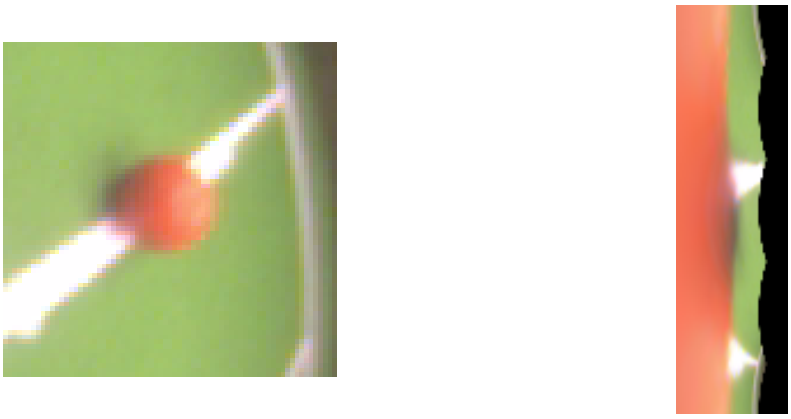


Abbildung 6: Der Ball (links) und das entsprechende Log-Polar-Bild (rechts) mit dem Zentrum im Ballmittelpunkt ($M = 10$)

Ist ein Ball hingegen nicht im Zentrum des Log-Polar-Bildes, so ergibt sein Rand eine Kurve. Dies zeigt Abbildung 7.

Aus dem Verlauf dieser Kurve lassen sich wiederum Rückschlüsse auf die Position des Ballmittelpunkts ziehen, was für ein Trackingverfahren eine potentiell interessante Eigenschaft ist.

4.3. Zu verwendende Technologien

Da das zu entwickelnde Verfahren beim *1. RFC Stuttgart* im Rahmen der *Middle Size League* des *RoboCups* eingesetzt werden soll, ergeben sich einige Anforderungen an die zu verwendenden Technologien. Die Entwicklung findet in der Programmiersprache *C++*

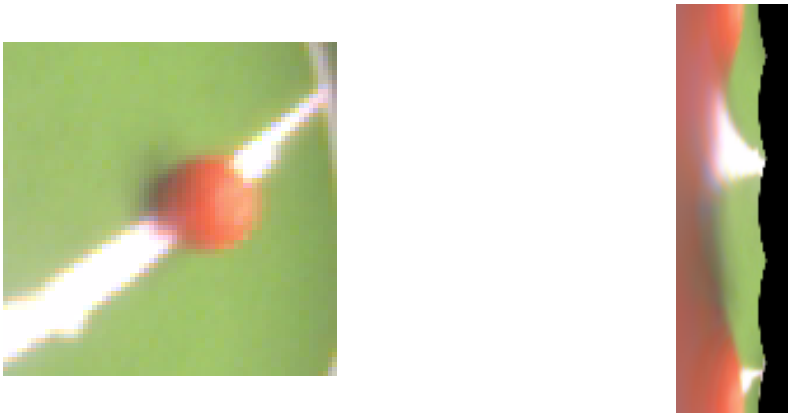


Abbildung 7: Der Ball (links) und das entsprechende Log-Polar-Bild (rechts, $M = 10$)

statt. Einzige Abhängigkeit soll die *OpenCV*-Bibliothek sein. Unter diesen Voraussetzungen kann das entwickelte Verfahren dann problemlos in das RoboCup-Software-Framework des *1. RFC Stuttgart* integriert werden.

5. Balltracking mit der Log-Polar-Transformation

Die Aufgabe des *Tracking* eines Balls besteht darin, ausgehend von einer Startposition in einem Kamerabild, den Ball zu lokalisieren, der in diesem Fall durch einen Kreis mit Mittelpunkt und Radius definiert wird. Dabei gibt es grundsätzlich zwei übergeordnete Probleme zu lösen:

- Wie können die Kanten des Balls gefunden werden?
- Wie können aus diesen Randpositionen der Mittelpunkt und der Radius des Balls bestimmt werden?

Natürlich sind diese beiden Probleme nicht unabhängig voneinander, sondern beeinflussen sich gegenseitig. Je exakter die Kanten des Balls ermittelt werden können, desto einfacher lassen sich daraus dessen Größe und Position bestimmen. Wenn diese Positionsbestimmung allerdings toleranter gegenüber Ausreißern ist, darf man bei der Detektion der Kanten großzügiger sein. Wichtig ist also, dass diese beiden Probleme aufeinander abgestimmt gelöst werden, damit die Ausgabe der Kantendetektion dem Positionsbestimmungsverfahren genügt, um zuverlässig die korrekte Position und Größe des Balls zu bestimmen.

Auf Grund der Forderung nach geringer Laufzeit kann hier nicht beliebig Rechenzeit investiert werden. Beide Probleme müssen mit einem Minimum an Operationen gelöst werden. Hier ist es, wie meist in der Informatik, notwendig, einen Kompromiss zwischen Güte der Ergebnisse und aufgewendeter Laufzeit zu finden.

Im Folgenden werden die entwickelten Lösungsansätze für die beiden Teilprobleme dargestellt und verglichen und anschließend dargelegt, aus welchen Gründen welcher Ansatz schließlich verwendet wurde.

5.1. Detektion der Ballkanten

Um die Kanten eines Balls in einem Kamerabild zu detektieren, ist das grundsätzlich zu lösende Problem, Ballpixel von Nicht-Ballpixeln zu unterscheiden. Anschließend kann der Übergang von einem Nicht-Ballpixel zu einem Ballpixel oder umgekehrt als Kante des Balls klassifiziert werden. Dies allein genügt aber nicht: Hier würden aus mehreren Gründen sehr viele falsche Kanten erkannt. Die Probleme sind im einzelnen:

Verdeckung Bälle, die durch andere Roboter oder andere Hindernisse oder die Aufbauten des Bezugsroboters selbst teilweise verdeckt sind. In diesem Fall würden dann Kanten an der Grenze zum verdeckten Bereich erkannt, was dazu führen würde, dass der Ball erheblich unterschätzt wird.

Bildrauschen Wenn tatsächlich jeder Übergang als Kante gezählt werden würde, so wären allein durch das Kamerarauschen viele Fehlerkennungen enthalten.

Die Detektion der Kanten muss also stabil gegenüber Bildrauschen sein und die Kanten, die an Verdeckungen entstehen, ausschließen.

Im Folgenden ist das entwickelte Verfahren zur Kantendetektion mit seinen einzelnen Schritten beschrieben.

5.1.1. Detektion von Ballpixeln ohne Parameter

Das erste zu lösende Problem beim Tracking ist, wie beschrieben, die Unterscheidung zwischen Ballpixeln und Nicht-Ballpixeln. Im vorgegebenen Szenario soll dabei der Ball nicht vorgegeben sein, genaues Aussehen und Größe sind dem Verfahren also unbekannt. Die einzige zur Verfügung stehende Information ist, dass der Ball, im Gegensatz zu allen anderen Objekten auf dem Spielfeld im *RoboCup farbig* aber nicht grün ist, sprich sich deutlich vom Rasen abhebt. Es muss also ein Weg gefunden werden, wie „bunte“ Pixel von „nicht-bunten“ Pixeln unterschieden werden können.

Die Kamera des Roboters stellt die Bilder in „4:2:2“-Kodierung des YUV-Farbmodells dar (Siehe auch Unterabschnitt A.4, YUV-Farbmodell, auf Seite 60,). Wenn man die einzelnen Komponenten eines Bildes, das mit dem YUV-Farbmodell dargestellt wird, betrachtet, so stellt man fest, dass ein „farbiger“ Ball entweder im U- oder im V-Komponenten-Bild sehr deutlich zu erkennen ist. Die folgende Abbildung 8 verdeutlicht das.

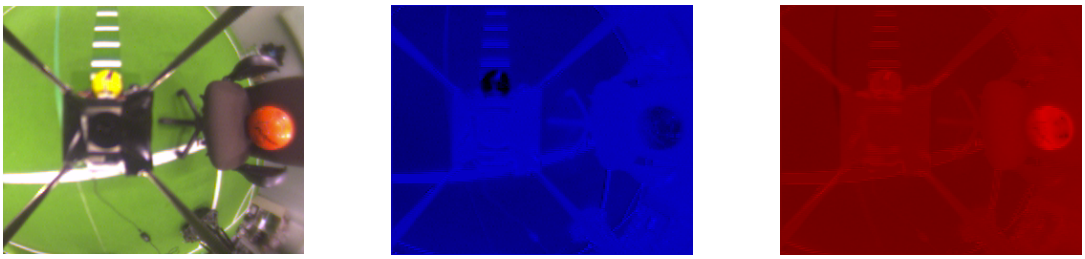


Abbildung 8: Links das Originalbild, in der Mitte die U-Komponente und rechts die V-Komponente. In den beiden rechten Bildern sieht man deutlich, dass jeweils der orangene beziehungsweise der gelbe Ball sehr gut zu erkennen sind.

Bildet man nun das Differenzbild, rechnet also an jedem Pixel $V - U$ aus, so sieht man, dass sich alle Bälle darauf deutlich abheben. In Abbildung 9 ist dies verdeutlicht.

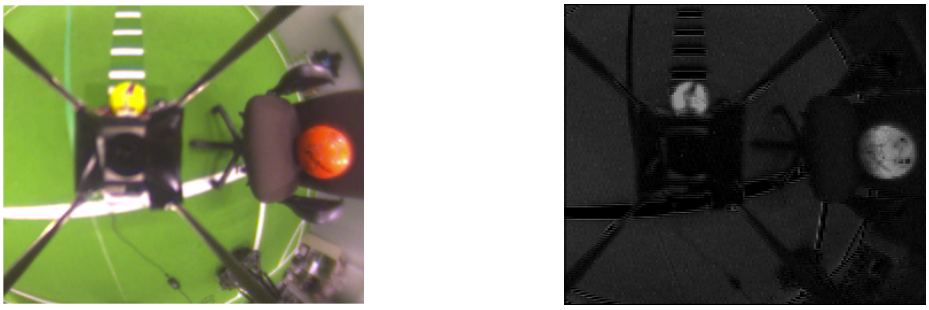


Abbildung 9: Links das Originalbild, rechts das $V - U$-Differenzbild. In diesem sind die Bälle klar zu erkennen.

In diesem Differenzbild heben sich die Bälle deutlich vom Hintergrund ab. Das liegt daran, dass die verwendeten Bälle alle einen erheblich größeren Rot- als Blauanteil in YUV-Darstellung haben. Das unterscheidet sie von allen anderen Pixeln des Bildes. Legt

man über das erzeugte Differenzbild nun noch einen Schwellwertfilter, hat man im daraus resultierenden Bild die Bälle aus dem Originalbild extrahiert. Dies kann man in Abbildung 10 sehen.

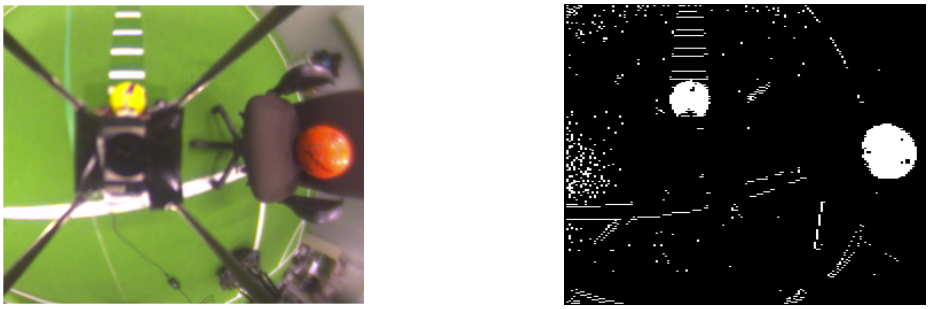


Abbildung 10: Links das Originalbild, rechts das $V - U$-Schwellwertbild (Schwellwert in diesem Fall $T_{vu} = 45$). Die beiden Bälle sind deutlich zu erkennen und nur leichtes Rauschen ist im Schwellwertbild enthalten.

Ein vollständiges Konvertieren des zu untersuchenden Kamerabildes in diese Darstellung ist allerdings zu zeitaufwendig. Da für diese Umwandlung jedoch nur die Information des Pixels selbst, sprich seine U- und V-Komponente, wichtig ist, kann diese Methode auch nur für jedes betrachtete Pixel ausgeführt werden. Ein Pixel gilt also zunächst einmal als Ballpixel, wenn folgende Bedingung erfüllt ist:

$$V_{Pixel} - U_{Pixel} > T_{vu} \tag{10}$$

Der Schwellwert T_{vu} muss dabei natürlich, je nach Eigenschaften und Kalibrierung der Kamera, unterschiedlich gewählt werden.

Detektieren von Feldpixeln Um das Ergebnis der Kantendetektion zu verbessern und das Rauschen im Schwellwertbild zu verringern, bietet es sich an, Spielfeldpixel von vorn herein auszuschließen. Wird also ein Pixel als Spielfeldpixel eingestuft, so kann es sich dabei nicht um ein Ballpixel handeln.

Auch zur Erkennung von „grünen“ Spielfeldpixeln eignet sich das Betrachten der einzelnen Kanäle eines YUV-Bildes. Vergleicht man sowohl die U- als auch die V-Komponente eines Bildes mit einem Schwellwert $T_{\text{grün}}$, so lässt sich das Grün des Spielfelds sehr gut von allen anderen Pixeln unterscheiden. Ein Pixel gilt als grün, wenn folgende Bedingung erfüllt ist:

$$(V_{Pixel} < T_{\text{grün}}) \wedge (U_{Pixel} < T_{\text{grün}}) \tag{11}$$

In Abbildung 11 ist dargestellt, welches Ergebnis diese Grün-Erkennung liefert.

Neben der Erkennung aller grünen Pixel können auch alle weißen Linien des Spielfelds als Ballpixel ausgeschlossen werden. Hier ist das YUV-Farbmodell ebenfalls von Vorteil: Vergleicht man den Wert der Luminanz-Komponente Y mit einem Schwellwert $T_{\text{weiß}}$,

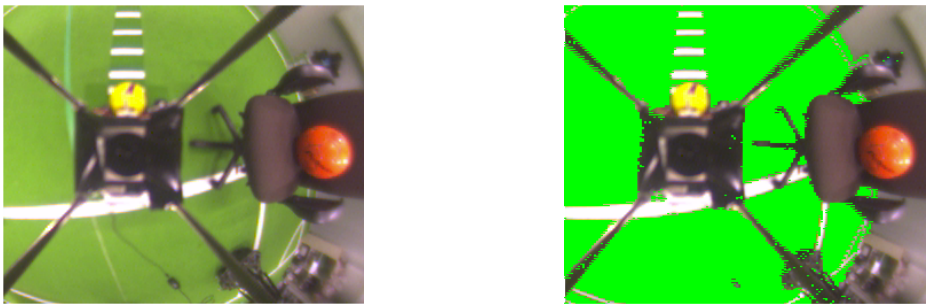


Abbildung 11: Links das Originalbild, rechts sind alle Pixel in grün hervorgehoben, die als Spielfeldpixel erkannt wurden.

kann man diese Linien schnell und zuverlässig erkennen. Ein Pixel wird als Linienpixel klassifiziert, wenn die folgende Bedingung erfüllt ist:

$$Y_{Pixel} > T_{\text{weiß}} \tag{12}$$

Abbildung 12 zeigt das Ergebnis an einem Beispiel. Es ist bei dieser Erkennung allerdings zu beachten, dass der Schwellwert nicht zu weit gesenkt wird, da ansonsten auch Ballpixel als „weiß“ erkannt und somit ausgeschlossen werden.

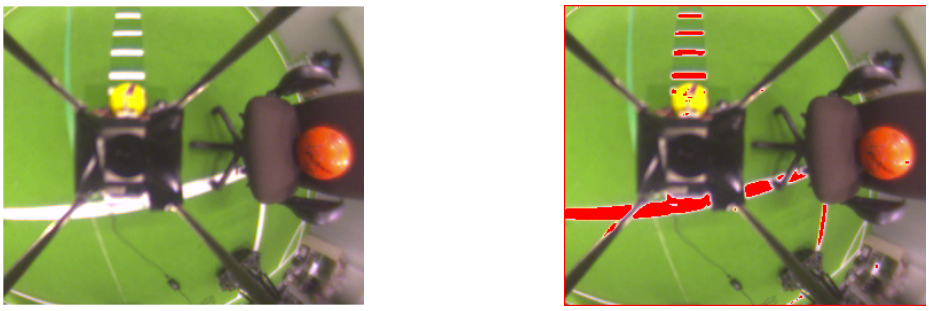


Abbildung 12: Links das Originalbild, rechts sind alle Pixel in rot hervorgehoben, die als weiße Linienpixel erkannt wurden.

Ignorieren von verdeckten Pixeln Bei den Fußballrobotern für den *RoboCup* gibt es noch eine weitere Besonderheit. Da die Roboter über eine Panorama-Kamera mit einem Parabolspiegel verfügen, sieht der Roboter in Teilen des Bildes sich selbst. Insbesondere die Streben der Haltekonstruktion, die den Spiegel über der Kamera fixiert, verdecken einige Pixel auf dem Kamerabild grundsätzlich. Außerdem sieht die Kamera ab einem bestimmten Winkel am Parabolspiegel vorbei und dann nur noch das Schwarz der Haltekonstruktion.

Dort, wo sich diese grundsätzlich verdeckten Pixel befinden, können dementsprechend auch keine Ballpixel erkannt werden, so dass diese Positionen von vorn herein ausgeschlossen werden können. Zu diesem Zweck wird ein Bild erzeugt, das diese Information

enthält: Ein Pixel darin hat den Wert „weiß“, wenn die Sicht der Kamera an dieser Position nicht durch den Roboter selbst eingeschränkt ist, ansonsten ist das Pixel „schwarz“. Dieses Bild wird nach der Kalibrierung des Roboters erstellt. Abbildung 13 zeigt dieses sogenannte *exclusion*-Bild.

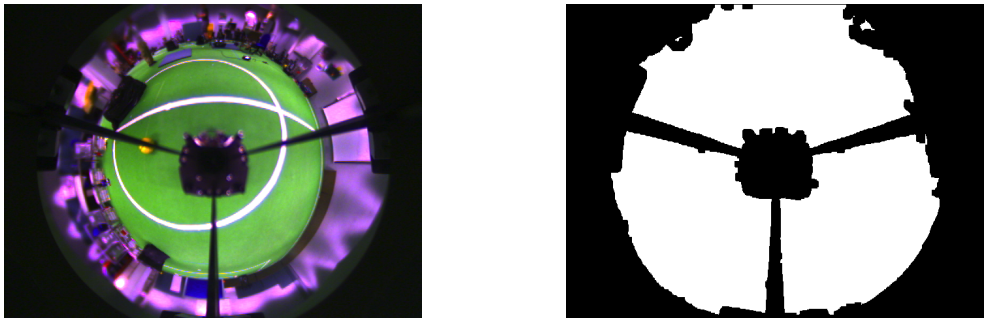


Abbildung 13: Links das Kamerabild eines Roboters, rechts das zugehörige *exclusion*-Bild.

Zusammenfassung: Erkennung eines Ballpixels Zusammenfassend wird ein Pixel mit den Koordinaten $(x_p|y_p)$ genau dann als Ballpixel erkannt, wenn folgende Bedingungen erfüllt sind:

$$exclusion(x_p, y_p) > 0 \quad \text{Das Pixel darf nicht ohnehin verdeckt sein} \tag{13}$$

und

$$Y_{x_p,y_p} <= T_{\text{weiß}} \quad \text{Das Pixel darf nicht weiß sein} \tag{14}$$

und

$$(U_{x_p,y_p} >= T_{\text{grün}}) \vee (V_{x_p,y_p} >= T_{\text{grün}}) \quad \text{Das Pixel darf nicht grün sein} \tag{15}$$

und

$$V_{x_p,y_p} - U_{x_p,y_p} > T_{vu} \quad \text{Das Pixel muss unter die Schwellwertbedingung fallen} \tag{16}$$

Für diese Klassifizierung benötigt man nur Informationen des Pixels selbst und keinerlei Parameter, die den Ball charakterisieren. Je nach Szenario kann es allerdings notwendig sein, beispielsweise „grüne“ Pixel auch als Ballpixel zuzulassen, da diese Auswahl unter Umständen auch tatsächliche Ballpixel ausschließen kann. Im *RoboCup*-Szenario und mit den Robotern des *1. RFC Stuttgart* hat sich die Klassifizierung in dieser Reihenfolge aber als effektiv erwiesen.

5.1.2. Log-Polar-Transformation mit Look-Up-Table

Wie bereits unter Unterabschnitt 4.2, Verwendung der Log-Polar-Transformation, auf Seite 15, geschildert, ist die Verwendung der Log-Polar-Transformation für das Tracking interessant und soll daher im Verfahren Verwendung finden. Das Auflösungsverhalten dieser Transformation in Bezug auf das Originalbild ist für schnelles Tracking hilfreich, da

die Anzahl der abgetasteten Pixel damit erheblich reduziert werden kann. Die Berechnung des Log-Polar-Bildes aus dem Originalbild ist allerdings aufwendig und erfordert somit relativ viel Rechenzeit. Damit wären alle Zeitvorteile, die durch die gröbere Abtastung gewonnen werden, hinfällig.

Um dieses Problem zu lösen, bietet es sich an, eine vorberechnete Look-Up-Table zu verwenden (Siehe Unterabschnitt A.3, Look-Up-Tables, auf Seite 60). Damit lässt sich das Log-Polar-Bild implizit berechnen, ohne die teure Transformation des gesamten Bildes berechnen zu müssen. In der Look-Up-Table wird gespeichert, welche Pixelposition im Originalbild einem Log-Polar-Koordinatenpaar $(\rho|\phi)$ entspricht. Die Look-Up-Table realisiert also die folgende Funktion:

$$LUT : \mathbb{N}^2 \to \mathbb{N}^2, (\rho, \phi) \mapsto (e^{\frac{\rho}{M}} \cos(\phi), e^{\frac{\rho}{M}} \sin(\phi)) \tag{17}$$

Mit dieser Look-Up-Table kann über die Log-Polar-Koordinaten ρ und ϕ iteriert werden, indem die entsprechende Position im Originalbild aus der Tabelle ausgelesen wird. An dieser Stelle wird dann das Originalbild ausgewertet. Das Bild wird folglich strahlenförmig abgetastet, und mit steigender Entfernung vom *Pol* des Log-Polar-Bildes sinkt die Auflösung entlang des Strahls durch die logarithmische Eigenschaft der ρ-Koordinate.

Bei dieser Art der Abtastung des Bildes ergibt sich aus dieser Eigenschaft ein Problem: Beim Iterieren der ρ-Koordinate wird diese nur für ganzzahlige Werte ausgewertet. Mit steigenden Werten vergrößert sich dadurch der Bereich zwischen zwei Abtastpunkten schnell. Stellt sich heraus, dass eine Kante zwischen zwei dieser Abtastpunkte liegt, so kann ihre eigentliche Position irgendwo zwischen diesen beiden liegen, die aber möglicherweise bereits im Bild einen größeren Abstand voneinander haben. Zwischen $e^2 = 7,389$ und $e^3 = 20,086$ liegen beispielsweise bereits 13 Schritte entlang des Strahls. Damit ist unklar, wo genau sich die Kante befindet. Da diese Information für die Ermittlung der Ballposition enorm wichtig ist, darf die Abtastrate des Bildes durch Verwendung der Log-Polar-Position nicht zu schnell zu weit sinken.

Um die Problematik etwas zu entschärfen, kann man den Mittelwert der zwei Abtastpositionen verwenden. Wird eine Kante also zwischen den beiden Werten $\rho_n \in \mathbb{N}$ und $\rho_{n+1} \in \mathbb{N}$ erkannt, so wird als resultierende Entfernung der Kante vom Pol des Log-Polar-Bildes, das die Look-Up-Table darstellt, der Wert

$$r_K = (e^{\frac{\rho_n}{M}} + e^{\frac{\rho_{n+1}}{M}}) \cdot \frac{1}{2} \tag{18}$$

gespeichert, also der Mittelwert der beiden Entfernungen.

Um die Abstände zwischen den abgetasteten Pixeln zu verringern, lässt sich der Skalierungsfaktor M verwenden. Wird dieser erhöht, so steigt die Abtastrate in der Nähe des Pols deutlich, ohne dass sie in größerer Entfernung so stark zunimmt, dass die vorteilhaften Eigenschaften der Log-Polar-Transformation verloren gehen. In Tabelle 1 ist der Verlauf der abgetasteten Entfernungen für zwei verschiedene Wert des Skalierungsfaktors M aufgelistet.

Wie man der Tabelle entnehmen kann, werden die Werte für $M = 1$ bei steigendem ρ schnell sehr groß, während sie für $M = 10$ wesentlich langsamer wachsen. Damit kann

	$e^{\frac{\rho}{M}}$	
ρ	$M = 1$	$M = 10$
0	1	1
1	$\sim 2,718$	$\sim 1,105$
2	$\sim 7,389$	$\sim 1,221$
3	$\sim 20,086$	$\sim 1,350$
4	$\sim 54,598$	$\sim 1,492$
5	$\sim 148,413$	$\sim 1,649$
6	$\sim 403,429$	$\sim 1,822$
7	$\sim 1096,633$	$\sim 2,014$

	$e^{\frac{\rho}{M}}$	
ρ	$M = 1$	$M = 10$
8	$\sim 2980,958$	$\sim 2,226$
9	$\sim 8103,084$	$\sim 2,457$
10	$\sim 22026,466$	$\sim 2,718$
...		
15	$\sim 3,270 \cdot 10^6$	$\sim 4,482$
20	$\sim 4,852 \cdot 10^8$	$\sim 7,389$
25	$\sim 7,200 \cdot 10^{10}$	$\sim 12,182$
30	$\sim 1,069 \cdot 10^{13}$	$\sim 20,086$

Tabelle 1: Auswirkung des Skalierungsfaktors M

man eine wesentlich genauere Abtastung erreichen. Wie bei den höheren Werten zu sehen ist, beginnt das Abtastverhalten auch für den größeren Skalierungsfaktor ab gewissen Werten, wie gewünscht, zu sinken.

Es zeigt sich dabei allerdings ein anderes Problem. Für Werte von ρ von 0 bis 10 nehmen die Werte nur sehr langsam zu, wenn $M = 10$ gilt. Das führt dazu, dass einzelne Pixel mehrfach abgetastet werden, was die Laufzeit des Verfahrens unnötig steigern würde.

Diesem Problem kann man dadurch begegnen, dass ρ nur auf Werte gesetzt wird, die für unterschiedliche Pixel stehen, so dass das selbe Pixel nicht mehr mehrfach betrachtet wird. Ein Wert $\rho = \rho_n$ ist also nur dann zu betrachten wenn gilt:

$$LUT : \mathbb{N}^2 \to \mathbb{N}^2, (\rho, \phi) \mapsto (e^{\frac{\rho}{M}} \cos(\phi), e^{\frac{\rho}{M}} \sin(\phi)) \tag{19}$$

$$LUT(\rho_{n-1}|\phi_0) \neq LUT(\rho_n|\phi_0) \quad \text{mit festem } \phi_0 \tag{20}$$

Um für die Prüfung dieser Bedingung keine Rechenzeit aufwenden zu müssen, kann eine Iterationsliste schon beim Generieren der Look-Up-Table mit erzeugt werden. Dazu wird für alle Werte von ϕ eine solche Liste erzeugt, in die dann Werte für ρ nur dann eingetragen werden, wenn sie die oben genannte Bedingung erfüllen. Beim Abtasten des Bildes selbst wird dann ρ nicht für jeden Wert ausgewertet, sondern nur für alle Werte aus der Liste.

Mit Hilfe dieser Maßnahmen kann die Abtastung des Bildes mit der Log-Polar-Transformation schon erheblich beschleunigt werden. Eine Messung zeigt aber, dass es trotzdem noch viele Pixel des Bildes gibt, die unnötigerweise mehrfach abgetastet werden. Tabelle 2 und Tabelle 3 zeigen das Ergebnis der Messung bei der Abtastung eines 640 Pixel breiten und 480 Pixel hohen Bildes.

Wie man an den Ergebnissen der Messung sehen kann, werden viele Pixel mehrfach abgetastet. Das führt dazu, dass etwa vier mal so viele Abtastvorgänge stattfinden, wie nötig wären. Die Ursache liegt darin, dass durch Rundung viele an sich unterschiedliche Log-Polar-Koordinaten dem selben Pixel zugeordnet werden. Man betrachte das Pixel,

n mal	Anzahl Pixel	n mal	Anzahl Pixel	n mal	Anzahl Pixel	n mal	Anzahl Pixel
0	303.216	10	36	24	12	62	4
1	2.390	12	28	28	8	88	8
2	707	13	12	30	4	108	4
3	191	14	8	31	8	118	4
4	180	15	4	33	4	148	2
5	112	16	28	38	8	149	2
6	60	18	4	39	4	162	4
7	28	19	4	48	8	298	4
8	48	20	12	55	4	315	2
9	24	22	8	58	4	316	2

Tabelle 2: Messung der mehrfach abgetasteten Pixel

Summe der Pixel:	307.200
davon abgetastet:	3.984
Summe der Abtastungen:	16.709
unnötige Abtastungen:	12.725
Abtastungen ($100\,\% \mathrel{\hat{=}} 3.984$):	$\sim 419,4\,\%$

Tabelle 3: Statistik zu Tabelle 2

auf dem sich der Pol befindet, und seine acht Nachbarpixel. Diese Pixel werden von einer Vielzahl von Strahlen mit unterschiedlicher Winkelkoordinate ϕ getroffen und somit mehrfach abgetastet. Abbildung 14 verdeutlicht das Beispiel.

Um diesen Effekt zu vermeiden, bietet es sich an, für bereits besuchte Pixel die Ergebnisse der Abtastung zu speichern und bei der erneuten Abtastung einfach auf diesen Cache zuzugreifen, anstatt das Pixel selbst erneut zu analysieren. Der Zugriff auf den Cache kostet hingegen fast keine Rechenzeit.

Eine Auslassung des bereits abgetasteten Pixels kommt nicht in Frage, da die strahlenförmige Kantensuche dadurch falsche Ergebnisse liefern könnte oder manche Kanten überhaupt nicht gefunden werden würden. Der Grund dafür ist im folgenden Unterunterabschnitt 5.1.3 geschildert. Mit der Einführung besagten Caches wird jegliche Mehrfachabtastung verhindert und die Laufzeit des Verfahrens aufgrund der eingesparten Rechenoperationen erheblich verbessert.

5.1.3. Erkennen von Ballkanten

Nachdem im vorangegangenen Abschnitt beschrieben wurde, wie man Ballpixel und Nicht-Ballpixeln unterscheidet und wie das Bild mit Hilfe der Log-Polar-Transformation abgetastet wird, geht es nun darum, wie und unter welchen Bedingungen Kanten

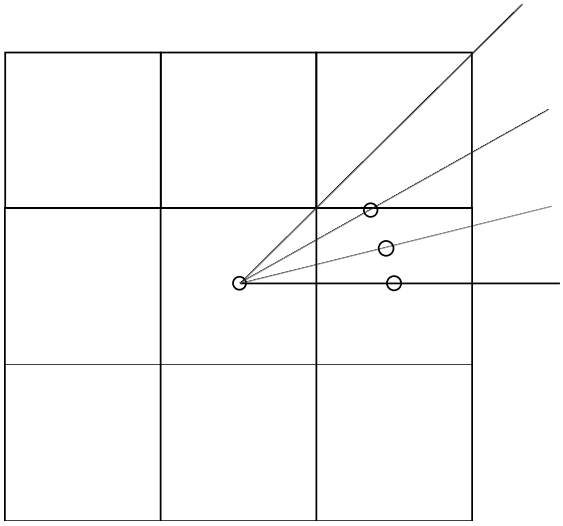


Abbildung 14: In der Skizze ist ein Pixel mit seinen acht Nachbarpixeln zu sehen, also ein Bildausschnitt von 3×3 Pixeln. Das selbe Pixel rechts neben dem Pol wird für eine Vielzahl von Werten der Winkelkoordinate ϕ abgetastet

des Balls erkannt werden.

Grundsätzlich kann sich eine Kante des Balls nur dort befinden, wo entlang eines Abtaststrahls von einem Ball-Pixel auf ein Nicht-Ball-Pixel übergegangen wird oder umgekehrt. Es müssen aber noch zusätzliche Kriterien festgelegt werden, um ein sinnvolles Detektieren von Kanten zu ermöglichen. Eine Einstufung aller Übergänge zwischen Ball und Nicht-Ball als Kanten würde dazu führen, dass man sehr viele falsche Ballkanten erkennen würde. Dies hat verschiedene Ursachen:

Kamerarauschen Das Rauschen der Kamera bewirkt leichte Fluktuationen im Bild, wodurch einzelne Pixel fälschlicherweise als Ball beziehungsweise Nicht-Ball eingestuft werden.

Das Detektionsverfahren Bei der in Unterabschnitt 5.1, Detektion der Ballkanten, auf Seite 18, beschriebenen Erkennung von Ballpixeln können, je nach Parametrierung, auch beispielsweise einzelne Punkte an den Grenzen von Feldlinien irrtümlicherweise als Ballpixel erkannt werden.

Verschattung und Texturierung des Balls Einzelne Pixel des Balls können als Nicht-Ball-Pixel erkannt werden, beispielsweise dann, wenn sie verschattet sind oder der Ball texturiert ist.

Um diesem Problem zu begegnen, wird, wie gesagt, nicht jeder Übergang zwischen Ball- und Nicht-Ball-Pixeln als Kante erkannt. Stattdessen muss dieser weitere Bedingungen erfüllen, um als Kante zugelassen zu werden.

Um die Erkennung von Kanten stabiler gegenüber Kamerarauschen und einzelnen falsch als Ball-Pixeln oder Nicht-Ball-Pixeln erkannten Pixeln zu machen, wird ein Ball- zu Nicht-Ball-Übergang (oder umgekehrt) nur dann als Kante zugelassen, wenn sich davor und danach jeweils eine gewisse Anzahl von gleichen Pixeln befindet. Zum Beispiel wird eine Kante dort erkannt, wo fünf Ball-Pixel auf fünf Nicht-Ball-Pixel folgen. Allgemein müssen die folgenden beiden Bedingungen erfüllt sein, damit eine Kante erkannt wird:

$$n_{\text{gleiche Pixel davor}} \geq T_{\text{minimum gleiche Pixel davor}} \tag{21}$$

$$n_{\text{gleiche Pixel danach}} \geq T_{\text{minimum gleiche Pixel danach}} \tag{22}$$

Damit werden einzelne falsch zugeordnete Pixel als mögliche Kanten ausgeschlossen. Übergänge werden zwischen Folgen gleich zugeordneter Pixel erkannt und nicht zwischen einzelnen Pixeln. Solche Folgen von Pixeln können natürlich auch wieder durch Rauschen fälschlicherweise ausgeschlossen werden, wenn sich, zum Beispiel nahe des Randes des Balls, ein einzelnes Nicht-Ball-Pixel befindet. Diese Überkompensation hat aber in der Praxis kaum Auswirkungen und kann daher verschmerzt werden, da sich durch diese Methode viele Ausreißer ausschließen lassen.

Eine weitere Möglichkeit, die Kantendetektion zu verbessern, ist, grüne Feldpixel zu beachten. Solche grünen Pixel dürfen sich natürlich nur auf der „Nicht-Ball-Seite“ der erkannten Kante befinden, ansonsten ist die Kante als Fehldetektion einzustufen. Damit können weitere Ausreißer von Kantenpositionen ausgeschlossen werden. Die hierbei anfallende Information über Kanten zu grünen Feldpixeln ist im folgenden Unterunterabschnitt 5.1.4 wieder von Bedeutung.

Kanten zu im *exclusion*-Bild (Siehe „Ignorieren von verdeckten Pixeln“ auf Seite 21) als verdeckt markierten Pixeln können ebenfalls von vorn herein ausgeschlossen werden. Befindet sich also ein Übergang an oder in unmittelbarer Nähe einer dieser Stellen, so wird er als mögliche Kante nicht zugelassen.

Um die Laufzeit der Kantenerkennung möglichst gering zu halten, soll jeder Abtaststrahl nur so lange abgelaufen werden, bis die maximale Anzahl von Kanten erkannt wurde. Dabei gibt es drei Möglichkeiten:

1. Es wird keine Kante erkannt.

2. Es wird eine Kante von Ball zu Nicht-Ball erkannt. Damit kann zum nächsten Strahl übergegangen werden, da hier keine weitere Kante mehr folgen kann, denn der Pol muss sich auf dem Ball befunden haben und nach Verlassen des Balls kann keine weitere Kante mehr folgen.

3. Es wird eine Kante von Nicht-Ball zu Ball erkannt. Dann muss der Strahl weiter abgelaufen werden, denn es muss eine weitere Kante von Ball zu Nicht-Ball gefunden werden können, außer diese liegt außerhalb der maximalen Suchdistanz. Das liegt daran, dass der Strahl, wenn er Ball betritt, ihn dann auch wieder verlassen muss. Dieser Fall kann nur eintreten, wenn der Pol des Log-Polar-Bildes nicht auf dem Ball liegt.

In Fall Nummer zwei kann die Abtastung früh abgebrochen werden, was dazu beiträgt, die Laufzeit des Verfahrens gering zu halten.

5.1.4. Klassifikation der Kanten

Die im vorherigen Unterunterabschnitt 5.1.3 beschriebene Erkennung von Kanten erkennt jedoch immer noch falsche Ballkanten, nämlich dort, wo der Ball von Robotern oder anderen Objekten verdeckt ist. An diesen Stellen werden Kanten erkannt, die aber eigentlich nicht zum Rand des Balls gehören. Um diesem Umstand Rechnung zu tragen, gibt es eine einfache Klassifikation, mit der diese Kanten ausgeschlossen werden können.

Die Kanten werden dazu in zwei Klassen unterteilt:

- Kanten zu einer Folge von grünen Feldpixeln
- Andere Kanten

Kanten zu Folgen von grünen Feldpixeln sind auf jeden Fall korrekte Kanten, die zum Rand des Balls gehören, da im *RoboCup* nur das Feld grün sein darf. Sie können daher nicht an Verdeckungen oder dergleichen entstehen. Gleichzeitig kostet diese Unterscheidung praktisch keine Rechenzeit, da die Grün-Eigenschaft von Pixeln in den vorangegangen Schritten Unterunterabschnitt 5.1.1 und Unterunterabschnitt 5.1.3 ohnehin für jedes betrachtete Pixel ermittelt wurde. Mit dieser einfachen Klassifikation lassen sich die Ergebnisse der Kantendetektion erheblich verbessern und nahezu alle Ausreißer ausschließen.

Die bei dieser Klassifikation ausgeschlossenen Kanten werden nicht verworfen. Sollten zu wenig Kanten zu grünen Feldpixeln erkannt werden, um den Ball zu bestimmen, so kann man auf sie zurückgreifen. Es existieren Situationen, in denen der Ball keine Kanten zu grünen Feldpixeln hat, beispielsweise, wenn er vollständig vor einem anderen Roboter oder am Rand des Feldes liegt. In diesen Fällen sind auch die „schlechteren“ Kanten wichtig, um überhaupt eine Ballposition bestimmen zu können.

5.1.5. Einsatz von Importance Sampling

Um die Detektion von Kanten zu verbessern und zu beschleunigen, wurde auch der Einsatz von *Importance Sampling* getestet. Ziel war es, durch dieses weitere Ausreißerkanten auszuschließen. Für alle gefundenen Kanten wurden benachbarte Abtaststrahlen untersucht und nur bei Erkennung einer Kante entlang dieser Strahlen wurde die Ursprungskante zugelassen. Dieser Vorgang wurde rekursiv für die Nachbarkanten wiederholt, bis die gewünschte Anzahl an bestätigten Kanten gefunden wurde. Der Einsatz von *Importance Sampling* hat sich jedoch als nicht praktikabel herausgestellt. Dies hatte hauptsächlich drei Gründe:

1. Durch Kamerarauschen wurden oft eigentlich korrekte Kanten ausgeschlossen, wenn eine der Nachbarpositionen keine Kante ergeben hat.

2. Der Einsatz von *Importance Sampling* führt zu einer Verdichtung der Kantenpositionen in einem bestimmten Bereich während andere Bereiche kaum abgetastet werden. Dadurch wurde die Detektion des Balls erheblich verschlechtert. Eine gleichmäßige Abtastung des Balls hat sich als hilfreich heurausgestellt, was den Einsatz von *Importance Sampling* als unzweckmäßig herausstellt.

3. Auch aus Laufzeiterwägungen wurde das getestete *Importance Sampling* wieder deaktiviert und dessen Einsatz verworfen, da sonst viele zusätzliche Strahlen hätten abgelaufen werden müssen, wenn der Ball trotzdem in alle Richtungen gleichmäßig abgetastet werden soll.

5.2. Berechnung der Position des Ballmittelpunkts und des Ballradius'

Nachdem die Positionen von Kantenpixeln, wie im vorangegangen Unterabschnitt 5.1 beschrieben, ermittelt wurden, muss aus diesen Kantenpositionen die Position und Größe des Balls bestimmt werden. Dafür wurde mit zwei Ansätzen experimentiert.

1. Bestimmung des Ballmittelpunkts durch Bestimmen der Abweichung von einer geraden Linie im Log-Polar-Bild

2. Bestimmung des Ballmittelpunkts und des Ballradius durch Einpassen eines Kreises.

Die beiden Ansätze sind im Folgenden beschrieben und werden anschließend verglichen.

5.2.1. Variante 1: Bestimmen der Abweichung von einer geraden Linie im Log-Polar-Bild

Um aus den ermittelten Kantenpixeln die Parameter des Ball zu bestimmen, kann die in Abschnitt 4.2, Balldarstellung im Log-Polar-Bild, auf Seite 16, beschriebene Eigenschaft des Log-Polar-Bildes genutzt werden, die besagt, dass der Rand eines Balls darin als Gerade beziehungsweise Kurve erscheint. Aus der Form dieser Randkurve im Log-Polar-Bild lässt sich auf die Position des Ballmittelpunktes und den Ballradius schließen. Dies funktioniert zunächst einmal natürlich nur dann, wenn der Pol bereits auf dem Ball liegt. Dies ist in den folgenden Betrachtungen, wenn nicht anders beschrieben, vorausgesetzt.

Zur Bestimmung von Ballposition und -radius stehen, wie beschrieben, die Kantenpixel aus der Kantenerkennung in Log-Polar-Koordinaten zur Verfügung. Um nun den Radius des Balls bestimmen zu können, kann über die Entfernungen der n Kantenpixel gemittelt werden. Dabei ist zu beachten, dass nicht über die Werte ρ_n der Koordinaten selbst gemittelt wird, sondern über die tatsächliche Entfernung e^{ρ_n} beziehungsweise $e^{\frac{\rho_n}{M}}$, wenn $M \neq 1$ gilt. Dies folgt aus den Eigenschaften der Log-Polar-Koordinaten. Der Radius des Balls ergibt sich also wie folgt:

$$r_{\text{Ball}} = \frac{1}{N} \cdot \sum_{k=1}^{N} e^{\frac{\rho_k}{M}} \quad \text{mit } N \text{ Kantenpixeln} \tag{23}$$

Würde sich der Pol auf dem Mittelpunkt des Balls befinden, so ergäbe sich im Log-Polar-Bild also eine Gerade bei $\rho = M \cdot \ln(r_{\text{Ball}})$.

Nun kann für jedes Kantenpixel die Distanz von dieser Gerade berechnet werden, also die Strecke, um die der Pol zu verschieben ist, damit das Kantenpixel im Log-Polar-Bild genau auf dieser Geraden liegt. Diese Distanz ergibt sich für das n-te Kantenpixel wie folgt:

$$d_{n,\text{Verschiebung}} = e^{\frac{\rho_n}{M}} - r_{\text{Ball}} \tag{24}$$

Die Richtung der Verschiebung ist zunächst unklar. Nimmt man nun einfach eine Verschiebung entlang der Geraden von Pol zu Kantenpixel um $d_{n,\text{Verschiebung}}$ an, ergibt sich für die berechnete Verschiebung für das n-te Kantenpixel mit den Log-Polar-Koordinaten $(\rho_n|\phi_n)$:

$$\vec{v}_n = \begin{pmatrix} d_{n,\text{Verschiebung}} \cdot \cos(\phi_n) \\ d_{n,\text{Verschiebung}} \cdot \sin(\phi_n) \end{pmatrix} \tag{25}$$

Um nun die korrekte Gesamt-Verschiebung zu bestimmen, müssen die Vektoren wieder bereinigt werden. Voraussetzung dafür ist, dass *der Ballrand gleichmäßig abgetastet wurde*. Dann heben sich für jeweils Paare von Kantenpositionen falsche Teilkomponenten der Verschiebung auf. Die x-Komponente gleicht sich dabei bei jeweils dem Paar aus, bei dem die Winkelkoordinate an der 90°- beziehungsweise 270°-Achse gespiegelt ist, die y-Komponente entsprechend bei den Paaren, deren Winkelkoordinaten an der 0°- beziehungsweise 180°-Achse gespiegelt sind. Übrig bleiben jeweils nur die Komponenten der Verschiebung, die in die korrekte Richtung weisen.

Nachdem, wie beschrieben, die entsprechenden Paare von Verschiebungsvektoren miteinander verglichen wurden, stellen die bereinigten Vektoren alle die korrekte Verschiebung dar. Um aber der Tatsache Rechnung zu tragen, dass das Bild nicht immer gleichmäßig abgetastet wird und es gewissem Rauschen unterliegt, wird der Mittelwert dieser Vektoren gebildet:

$$\vec{v}_{\text{ges}} = \frac{1}{N} \cdot \sum_{k=1}^{N} \vec{v}_{k,bereinigt} \tag{26}$$

Um diesen Vektor muss der Pol des Log-Polar-Bildes bewegt werden, um mit dem Mittelpunkt des Balls zusammenzufallen. Damit ist der Mittelpunkt des Balls bekannt:

$$\vec{v}_{\text{Ball}} = \begin{pmatrix} x_{\text{Pol}} + x_{\text{ges}} \\ y_{\text{Pol}} + y_{\text{ges}} \end{pmatrix} \tag{27}$$

5.2.2. Variante 2: Einpassen eines Kreises

Eine andere Möglichkeit, aus den detektierten Ballkantenpixeln die Parameter des Balls zu bestimmen, ist einen Kreis so einzupassen, dass möglichst alle der Kantenpixel auf dessen Rand liegen. Dieser Kreis stellt dann den Ball dar, das heißt, Radius und Position des

Mittelpunkts dieses Kreises stimmen mit dem Ball überein. Dafür müssen die ermittelten Kantenpixel, die in Log-Polar-Koordinaten gegeben sind, in kartesische Koordinaten umgewandelt werden. Diese Umwandlung wurde in Abschnitt 3.1, Logarithmieren der Entfernungen, auf Seite 12, beschrieben.

Im Folgenden wird zuerst erläutert, wie mögliche Kreise bestimmt werden können und anschließend, nach welchen Kriterien der schließlich ausgegebene Kreis ermittelt wird.

Berechnen möglicher Kreise Da drei Punkte einen Kreis eindeutig bestimmen, kann aus jedem 3-Tupel von Kantenpixeln ein möglicher Kreis berechnet werden, sofern diese nicht auf einer Geraden liegen. Dies liegt daran, dass ein solches 3-Tupel von Punkten ein Dreieck darstellt und der gesuchte Kreis dann gleich dem Umkreis dieses Dreiecks ist. Dies verdeutlicht Abbildung 15.

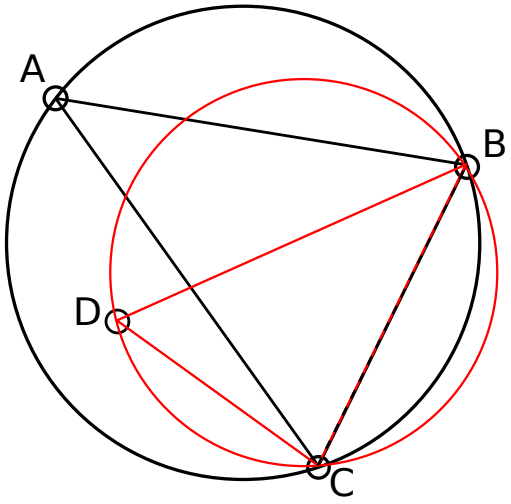


Abbildung 15: Vier Punkte mit zwei möglichen Dreiecken sowie deren Umkreisen.

Der Mittelpunkt des Umkreises eines Dreiecks ist gleichzeitig der Schnittpunkt der Mittelsenkrechten der Seiten dieses Dreiecks. Um also den Kreis zu bestimmen, genügt es, den Schnittpunkt zweier Mittelsenkrechten zu errechnen. Der Radius des Kreises ist dann der Abstand der Punkte von diesem Schnittpunkt. In Abbildung 16 ist dies zu sehen.

Soll also der Kreis, den das 3-Tupel A, B, C von Punkten darstellt, bestimmt werden, so beginnt man mit der Berechnung zweier Mittelsenkrechten des entsprechenden Dreiecks mit den Seiten a, b, c. Die Steigung einer Mittelsenkrechten bestimmt man aus der Steigung der Dreiecksseite. Die Steigung der Dreiecksseite a bestimmt sich wie folgt:

$$\vec{a} = \begin{pmatrix} x_C - x_B \\ y_C - y_B \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{y_C - y_B}{x_C - x_B} \end{pmatrix} = \begin{pmatrix} 1 \\ m_a \end{pmatrix} \tag{28}$$

Die Dreiecksseite a und deren Mittelsenkrechte s_a sind orthogonal zueinander. Mit dem

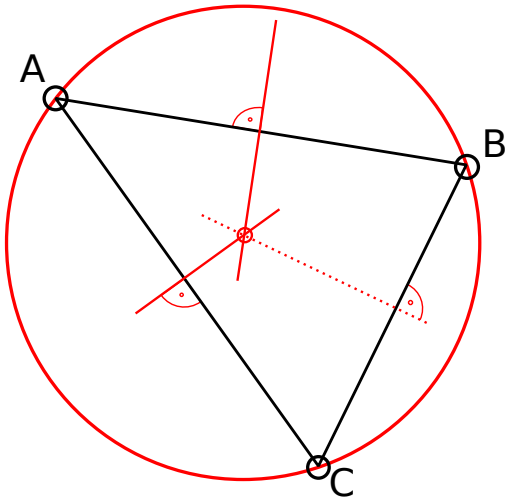


Abbildung 16: Ein Dreieck mit den Mittelsenkrechten der Seiten und dem daraus bestimmten Umkreis.

euklidischen Skalarprodukt gilt also:

$$\vec{a} \cdot \vec{s_a} = |\vec{a}| \cdot |\vec{s_a}| \cdot \cos(90°) = 0 \tag{29}$$

$$\vec{a} \cdot \vec{s_a} = x_a \cdot x_{s_a} + y_a \cdot y_{s_a} = 0 \tag{30}$$

$$1 \cdot 1 + m_a \cdot m_{s_a} = 0 \tag{31}$$

$$m_a \cdot m_{s_a} = -1 \tag{32}$$

$$m_{s_a} = -\frac{1}{m_a} \tag{33}$$

$$m_{s_a} = -\frac{1}{\frac{y_C - y_B}{x_C - x_B}} \tag{34}$$

$$m_{s_a} = \frac{x_B - x_C}{y_C - y_B} \tag{35}$$

Für die Seite b gilt entsprechend:

$$m_{s_b} = \frac{x_A - x_C}{y_C - y_A} \tag{36}$$

Um den Schnittpunkt der beiden Mittelsenkrechten berechnen zu können, braucht man diese allerdings als Geradengleichungen, also muss man auch noch die y-Achsen-Abschnitte bestimmen. Dafür verwendet man die Mittelpunkte der Dreiecksseiten, die ja den Schnittpunkt mit der jeweiligen Mittelsenkrechten darstellen. Diese bestimmt man

wie folgt:

$$\vec{M}_a = \begin{pmatrix} x_{M_a} \\ y_{M_a} \end{pmatrix} = \vec{B} + \frac{\vec{C} - \vec{B}}{2} \tag{37}$$

$$\vec{M}_a = \begin{pmatrix} x_B \\ y_B \end{pmatrix} + \begin{pmatrix} x_C - x_B \\ y_C - y_B \end{pmatrix} \cdot \frac{1}{2} = \begin{pmatrix} x_B \\ y_B \end{pmatrix} + \begin{pmatrix} \frac{x_C - x_B}{2} \\ \frac{y_C - y_B}{2} \end{pmatrix} \tag{38}$$

$$\vec{M}_a = \begin{pmatrix} x_B + \frac{x_C - x_B}{2} \\ y_B + \frac{y_C - y_B}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \cdot (x_C + x_B) \\ \frac{1}{2} \cdot (y_C + y_B) \end{pmatrix} \tag{39}$$

Nun kann durch Einsetzen dieses Seitenmittelpunkts M_a die Geradengleichung der Mittelsenkrechten s_a der Seite a bestimmt werden:

$$s_a(x) = m_{s_a} \cdot x + c_{s_a} \tag{40}$$

$$c_{s_a} = s_a(x) - m_{s_a} \cdot x \tag{41}$$

$$c_{s_a} = \frac{1}{2}(y_C + y_B) - \frac{1}{2} m_{s_a}(x_C + x_B) \quad \text{mit } \vec{M}_a = \begin{pmatrix} \frac{1}{2} \cdot (x_C + x_B) \\ \frac{1}{2} \cdot (y_C + y_B) \end{pmatrix} \tag{42}$$

Entsprechend gilt für c_{s_b}:

$$c_{s_b} = \frac{1}{2}(y_C + y_A) - \frac{1}{2} m_{s_b}(x_C + x_A) \tag{43}$$

Der gesuchte Umkreismittelpunkt M ergibt sich nun durch Gleichsetzen der Geraden:

für $m_{s_a} \neq \infty, m_{s_b} \neq \infty$

$$m_{s_a} x_M + c_{s_a} = m_{s_b} x_M + c_{s_b} \tag{44}$$

$$m_{s_a} x_M = m_{s_b} x_M + c_{s_b} - c_{s_a} \tag{45}$$

$$m_{s_a} x_M - m_{s_b} x_M = c_{s_b} - c_{s_a} \tag{46}$$

$$x_M(m_{s_a} - m_{s_b}) = c_{s_b} - c_{s_a} \tag{47}$$

$$x_M = \frac{c_{s_b} - c_{s_a}}{m_{s_a} - m_{s_b}} \tag{48}$$

$$y_M = m_a x_M + c_a \tag{49}$$

$$\rightarrow \vec{M} = \begin{pmatrix} \frac{c_{s_b} - c_{s_a}}{m_{s_a} - m_{s_b}} \\ m_a \left(\frac{c_{s_b} - c_{s_a}}{m_{s_a} - m_{s_b}} \right) + c_a \end{pmatrix} \tag{50}$$

für $m_{s_a} = \infty, x_a = x_0$ const ($m_{s_b} = \infty$ analog)

$$x_M = x_a \tag{51}$$

$$y_M = m_b x_M + c_b \tag{52}$$

Nachdem der Umkreismittelpunkt M bekannt ist, ergibt sich der Radius r des Umkreises als Abstand zwischen M und einem der drei Dreieckseckpunkte.

$$r = |\vec{MA}| = |\vec{A} - \vec{M}| = \left| \begin{pmatrix} x_A - x_M \\ y_A - y_M \end{pmatrix} \right| = \sqrt{(x_A - x_M)^2 + (y_A - y_M)^2} \tag{53}$$

N	$\binom{N}{3}$
3	1
4	4
5	10
10	120
20	1140
30	4060
50	19600

Tabelle 4: Einige Beispielwerte für den Binomialkoeffizienten

Anzahl möglicher Kreise Wie beschrieben, kann aus drei Punkten ein Kreis eindeutig bestimmt werden. Die Anzahl verschiedener Kreise, die aus der Gesamtanzahl N an gefundenen Kantenpixeln bestimmt werden kann, ist durch den entsprechenden Binomialkoeffizienten gegeben.

$$\binom{N}{3} = \frac{N!}{3! \cdot (N-3)!} \tag{54}$$

Mit steigendem N nimmt dieser Wert schnell zu. Tabelle 4 zeigt einige Beispielwerte.

Wie man in der Tabelle sehen kann, sind die Werte des Binomialkoeffizienten bereits für relativ niedrige Werte von N sehr groß. Die Anzahl der verwendeten 3-Tupel von Punkten muss also beschränkt werden, ansonsten würde die Laufzeit mit steigender Anzahl von detektierten Punkten sehr schnell stark steigen.

Es muss also ein Teil aller möglichen 3-Tupel ausgewählt werden. Dies kann entweder durch ein zufallsbasiertes oder ein deterministisches Verfahren erreicht werden. Es wurde mit beiden Arten der Auswahl experimentiert und schließlich der deterministischen der Vorzug gegeben, um für das selbe Eingabebild auch immer den selben Ball zu erkennen.

Auswahl des besten Kreises Aus der ermittelten Anzahl von potentiellen Kreisen, die aus jeweils einem 3-Tupel von Kantenpixeln bestimmt wurden, ist der bestmögliche Kreis auszuwählen. Dafür muss ein Kriterium gefunden werden. Bezogen auf die ermittelten Kantenpixel ist der beste Kreis derjenige, bei dem diese Pixel am nächsten zum Rand liegen, also deren Abstände am geringsten sind. Damit bietet es sich an, sich hier an der *Methode der kleinsten Fehlerquadrate* oder englisch *Least Squares* zu orientieren. Bei dieser Methode wird, um eine Funktion f optimal an N gegebene Messpunkte $(x_k, y_k), k = 1 \dots N$ anzugleichen, die Summe der Fehlerquadrate

$$\sum_{k=1}^{N} (f(x_k) - y_k)^2 \tag{55}$$

minimiert.

Bei den Kreisen wird diese Summe also gebildet, indem die quadrierten Abstände der Kantenpixel von dem Kreis aufsummiert werden. Für einen Kreis m mit Mittelpunkt $(x_M|y_M)$ und Radius r_m und ein Kantenpixel k mit Position $(x_k|y_k)$ entspricht der Abstand des Pixels vom Kreismittelpunkt der Länge des Vektors $\vec{mk}$. Diese berechnet sich wie folgt:

$$\vec{mk} = \begin{pmatrix} x_k - x_M \\ y_k - y_M \end{pmatrix} \tag{56}$$

$$|\vec{mk}| = \left| \begin{pmatrix} x_k - x_M \\ y_k - y_M \end{pmatrix} \right| = \sqrt{(x_k - x_M)^2 + (y_k - y_M)^2} \tag{57}$$

Der Betrag zum Fehlerwert für dieses Kantenpixel k besteht aus dem quadrierten Abstand dieses Pixels zum Rand des Kreises. Mit dem bereits berechneten Abstand vom Kreismittelpunkt erhält man die Distanz des Pixels vom Rand des Kreises, indem man dessen Radius von diesem Abstand subtrahiert:

$$\eta_k = (|\vec{mk}| - r_m)^2 \tag{58}$$

$$\eta_k = (\sqrt{(x_k - x_M)^2 + (y_k - y_M)^2} - r_m)^2 \tag{59}$$

Abbildung 17 verdeutlicht dieses Prinzip.

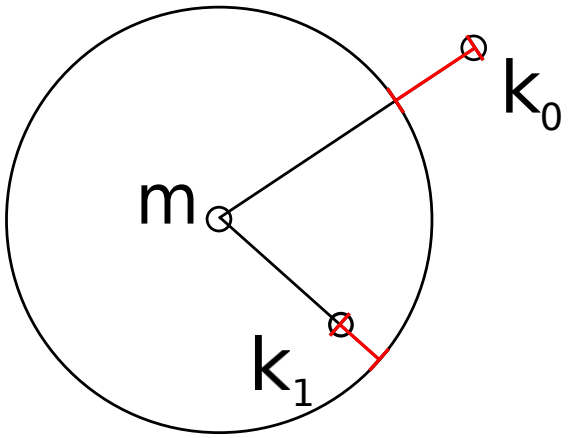


Abbildung 17: In der Skizze ist ein Kreis und zwei Pixel zu sehen. In rot sind jeweils deren Abstände zum Kreis markiert, die dann quadriert deren Beitrag zum Gesamtfehler sind.

Der gesamte Fehlerwert η_m des Kreises m für alle N Kantenpixeln ergibt sich also zu:

$$\eta_m = \sum_{k=1}^{N} \left(\left| \begin{pmatrix} x_k - x_M \\ y_k - y_M \end{pmatrix} \right| - r_m \right)^2 \tag{60}$$

$$\eta_m = \sum_{k=1}^{N} \left(\sqrt{(x_k - x_M)^2 + (y_k - y_M)^2} - r_m \right)^2 \tag{61}$$

Aus der Menge der Kreise K wird der Kreis mit dem niedrigsten η_m als der Beste gewählt.

$$K_{\text{Ergebnis}} = K_m \in K : \eta_m < \eta_k \forall K_k \in K \tag{62}$$

Aus den Parametern dieses Kreises werden dann Ballposition und -radius bestimmt. Da die verwendeten Pixelkoordinaten der Ballkantenpixel als Ursprung den Pol des Log-Polar-Bildes haben, müssen dessen Koordinaten noch addiert werden, um die absolute Position des Ballmittelpunkts zu bekommen. Der Radius des Balls entspricht dem Radius des besten Kreises K_{Ergebnis}.

$$r_{\text{Ball}} = r_{\text{Ergebnis}} \tag{63}$$

$$\vec{v}_{\text{Ball}} = \begin{pmatrix} x_{\text{Pol}} + x_{\text{Ergebnis}} \\ y_{\text{Pol}} + y_{\text{Ergebnis}} \end{pmatrix} \tag{64}$$

Alternative: Median der Fehlerquadrate Es wurde auch damit experimentiert, anstatt der Summe den Median der Fehlerquadrate zu verwenden und dann den Kreis zu wählen, der den geringsten Median hat. Im Vergleich waren die Ergebnisse grundsätzlich robuster gegenüber Ausreißern als bei der Summenberechnung, es gab allerdings Konstellationen, in denen das Verfahren Kreise wählte, die sehr große Summen gehabt hätten, und deren Parameter vom denen des eigentlichen Balls sehr weit entfernt waren. Dies trat beispielsweise zu Tage, wenn Teile des Balls verdeckt waren.

Ein grundsätzliches Problem ist, dass die Berechnung des Medians sehr langsam ist, da hierzu die einzelnen Fehlerquadrate für jeden Kreis gespeichert und anschließend sortiert werden müssen. Auch mit einem schnellen Sortierverfahren ist dies erheblich langsamer als das einfache Aufsummieren.

In dem man die Anzahl der betrachteten Kreise erhöht, konnte bei dann gleicher Laufzeit die Bestimmung mit der Fehlerquadrat-Summe mindestens gleichwertige, meist wesentlich bessere Ergebnisse liefern, als die Bestimmung mit dem Median und weniger betrachteten Kreisen. Daher wurde dieser Ansatz wieder verworfen.

5.2.3. Vergleich der beiden Varianten

Die beiden vorgestellten Varianten, den Mittelpunkt und den Radius des Balls aus den ermittelten Kantenpixeln zu bestimmen, haben beide ihre Stärken und Schwächen. Stärke des ersten Verfahrens ist insbesondere seine geringe Laufzeit. Variante zwei benötigt demgegenüber ein mehrfaches dieser Laufzeit, sie liegt aber immer noch im vertretbaren Rahmen. Bei einem gleichmäßig abgetasteten Ball liefern beide Varianten gute Ergebnisse, wobei die der zweiten Variante trotzdem meist genauer sind.

Variante eins hat aber einen ganz erheblichen Nachteil, der dazu geführt hat, dass sie in der Praxis nicht verwendbar ist. Um vernünftige Ergebnisse zu liefern, muss der Ball in alle Richtungen gleichmäßig abgetastet sein, damit sich die Vektoren gegenseitig bereinigen können. Die ist aber nur gegeben, wenn der Ball vollständig sichtbar ist. Sobald Teile des Balls verdeckt sind, ist eine solche Abtastung kaum mehr möglich. Außerdem ist die Variante deutlich empfindlicher gegenüber Ausreißern als Variante zwei. Zudem

muss für die erste Variante der Pol bereits auf dem Ball selbst liegen, was eine weitere Einschränkung bedeutet.

Variante zwei hingegen liefert nahezu immer gute Ergebnisse, auch wenn Teile des Balls verdeckt sind und es einzelne Ausreißer gibt, an denen Kantenpixel an falschen Positionen erkannt wurden. Auch wenn der Pol sich nicht auf dem Ball befindet, ist dies kein Problem für die zweite Variante.

Diese Faktoren gaben den Ausschlag zum Verwenden der zweiten Variante und Verwerfen der Ersten. Diese basiert zwar auf einer interessanten Idee, hat sich jedoch in der Praxis aufgrund der genannten Einschränkungen als unbrauchbar erwiesen. Auch die Laufzeit ist hier nicht ausschlaggebend, da Variante zwei immer noch ausreichend schnell ist, um die gesetzten Anforderungen zu erfüllen, und gleichzeitig mit ihrer Robustheit und ihren guten Ergebnissen überzeugt.

5.2.4. Weit entfernte Bälle

Sehr weit vom Roboter entfernte Bälle erscheinen im Kamerabild sehr klein. Das führt dazu, dass für diese Punkt zu wenige Kantenpixel gefunden werden, um eine der beiden Varianten vernünftige Ergebnisse produzieren zu lassen. Trotzdem geben diese wenigen Kantenpixel eine Information darüber, wo ungefähr sich der Ball befindet. Bei Bällen in dieser Entfernung ist die exakte Position und Größe des Balls ohnehin nicht so wichtig wie bei nahen Bällen, da eine genauere Positionierung des Roboters bei so weit entfernten Bällen nicht nötig ist. Stattdessen wird der Roboter nur grob in die Richtung des Balls fahren und mit sich verringernder Entfernung vom Ball auch genauere Positionsinformation erhalten.

Um auch in solchen Fällen, in denen der Ball eigentlich zu weit weg ist, eine Information vom Tracking zu erhalten, wird in diesem Fall die gemittelte Position der N gefundenen Kantenpixel und ein Radius von 0 zurückgegeben.

$$\vec{v}_{\text{Ball}} = \frac{1}{N} \sum_{k=1}^{N} \begin{pmatrix} x_N \\ y_N \end{pmatrix} \tag{65}$$

$$r_{\text{Ball}} = 0 \tag{66}$$

5.2.5. Information zur Ermittlung der Güte des gefundenen Balls

Wie in Unterabschnitt 4.1, Tracking des Balls, auf Seite 14, erklärt, ist es wichtig, dass das Tracking-Verfahren auch Informationen über den ermittelten Ball liefert, mit deren Hilfe sich ein Qualitätswert für diesen Ball bestimmen lässt. Nur dann ist man in der Lage, bei mehreren möglichen Ballpositionen die richtige zu bestimmen oder, falls es nur sehr ungenaue Positionsbestimmungen gibt, eine neue globale Suche nach möglichen Ballpositionen anzustoßen.

Zu diesem Zweck gibt das Tracking nach erfolgreichem Finden eines Balls vier verschiedene Informationen zurück.

1. **Art des gefunden Balls.** Der erste zurückgelieferte Wert sagt aus, wie der Ball

lokalisiert wurde. Damit lässt sich eine grobe Qualitätsabstufung in drei Stufen treffen:

Bälle mit Kanten zu grünen Feldpixel. Die besten Bällen sind gleichzeitig der Normalfall, nämlich Bälle, deren Position und Größe ermittelt wurde, in dem die Kantenpixel an Kanten zu grünen Feldpixeln verwendet wurden.

Bälle mit Kanten zu anderen Pixel Qualitativ wesentlich schlechter sind die Bälle, bei denen nicht genügend Kanten zu grünen Feldpixeln vorhanden waren und bei denen auch die anderen gefunden Kantenpixel zur Positionsbestimmung verwendet wurden. Diese sind mit einer wesentlich größeren Unsicherheit behaftet, als die Kantenpixel zu grünen Feldpixeln. Daher sind für diese Bälle die ermittelten Ballparameter tendenziell weniger exakt.

Sehr weit entfernte Bälle Wie im vorherigen Unterunterabschnitt 5.2.4 beschrieben, gibt es auch den Fall, wenn es zu wenig Kantenpixel für jede genaue Positionsbestimmung gibt. In diesem Fall wird nur der Mittelwert der Punkt und ein Radius von 0 zurückgegeben. Bei so bestimmten Ballparametern handelt es sich um die am wenigsten genauen.

2. **Anzahl der gefunden Kantenpositionen.** Die Anzahl der gefundenen Kantenpixel lässt ebenfalls auf die Güte des ermittelten Balls schließen, da mehr gefundene Kantenpositionen auch normalerweise eine exaktere Positionsbestimmung ermöglichen. Insbesondere gilt der umgekehrte Fall, je weniger Kantenpixel gefunden wurden, desto weniger exakt ist die Positionsbestimmung.

3. **Summe der quadrierten Abweichungen.** Die Summe der quadrierten Abweichungen desjenigen Kreises, der dann zurückgegeben wird, wird ebenfalls ausgegeben. Wie diese bestimmt wird, kann in Abschnitt 5.2.2, Auswahl des besten Kreises, auf Seite 34, nachgelesen werden. Auch aus diesem Wert lässt sich auf die Güte der ermittelten Ballparameter schließen

4. **Alle anderen ermittelten Kreise.** Die, wie in Unterunterabschnitt 5.2.2, Variante 2: Einpassen eines Kreises, auf Seite 30, beschrieben, anderen Kreise, die aus den Kantenpixeln ermittelt und als mögliche Ballposition in Betracht gezogen wurden, werden ebenfalls ausgegeben. Falls die vom Verfahren ermittelten Ballparameter ausgeschlossen werden können, kann als Rückfallwert hieraus ein anderer Kreis ausgewählt werden.

Durch die Ausgabe dieser vier Parameter ist von dort, von wo das Tracking aufgerufen wird, eine Gütebestimmung für die ermittelten Ballparameter möglich.

5.3. Zusammenfassung und Beispielablauf

In den voran gegangenen Unterabschnitten wurde erläutert, wie in einem Bild Kantenpixel gefunden werden können und wie man aus diesen Positionen dann die Parameter des

Balls bestimmen kann. Um einen Überblick über den Ablauf eines kompletten Tracking-Schrittes zu geben, werden im Folgenden die dazu nötigen Schritte einzeln und in der entsprechenden Reihenfolge aufgelistet.

Zunächst steht dem Tracking-Verfahren wie in Unterabschnitt 4.1, Tracking des Balls, auf Seite 14, erläutert eine Initial-Position zur Verfügung. Dabei handelt es sich entweder um die Position des Balls im letzten Kamerabild oder, falls diese nicht bekannt ist, um die Ausgabe eines globalen Balldetekionsverfahrens, dass gewisse Regionen mit potentiellen Bällen ermittelt. Von dieser Initialposition aus werden dann alle Schritte des Trackings durchgeführt. Abbildung 18 zeigt ein Beispiel für ein gesehenes Kamerabild und die gelieferte Initialposition.

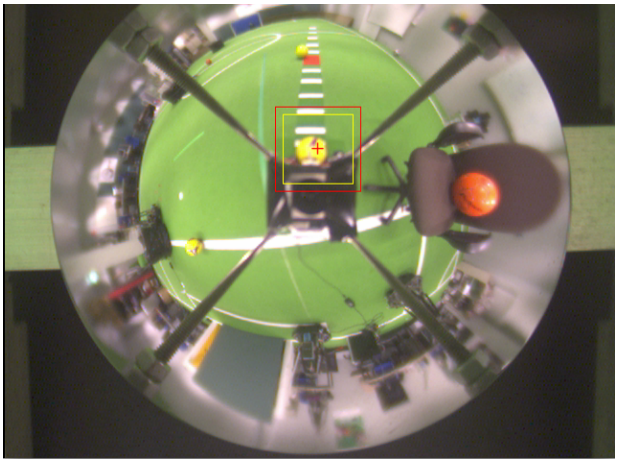


Abbildung 18: Tracking-Beispiel: Das Bild, das der Roboter sieht. Mit einem roten Kreuz ist die Initialposition markiert.

Ausgehend von dieser Position wird nun das Bild abgelaufen, so wie in Unterunterabschnitt 5.1.2, Log-Polar-Transformation mit Look-Up-Table, auf Seite 22, beschrieben. Dabei werden die Punkte in Nicht-Ball-Pixel und Ball-Pixel unterteilt und Kantenpixel gesucht, dies wurde in Unterunterabschnitt 5.1.1, Detektion von Ballpixeln ohne Parameter, auf Seite 19, beschrieben. Dabei wird für jeden untersuchten Pixel der Wert von $V - U$ betrachtet und mit einem Schwellwert T_{vu} verglichen, Abbildung 19 zeigt das entsprechende Bild.

Die gefunden Kantenpixel werden dann klassifiziert, je nachdem, ob sie an Kanten zu grünen Feldpixeln liegen oder nicht. Solche Kantenpixel werden bevorzugt behandelt und andere gefundene Kantenpixel werden nur dann verwendet, wenn die Zahl der Kanten zu Feldpixeln nicht ausreicht. Abbildung 20 zeigt die im Beispiel als grün erkannten Pixel.

In der darauf folgenden Abbildung 21 sind die gefunden Kantenpixel zu sehen.

Damit ist die Kantendetektion abgeschlossen. Nun müssen aus den ermittelten Kantenpixeln die Position des Ballmittelpunkts und der Ballradius bestimmt werden. Wie das im Einzelnen gemacht wird, wurde in Unterabschnitt 5.2, Berechnung der Position des Ballmittelpunkts und des Ballradius', auf Seite 29, erklärt. Dabei wird aus einer Menge

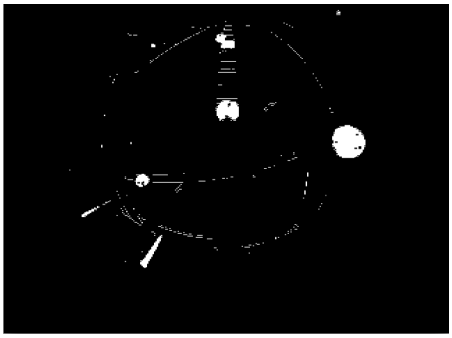


Abbildung 19: Tracking-Beispiel: Im Bild aus Abbildung 18 wurde für jeden Pixel $V - U$ mit einem Schwellwert T_{vu} verglichen. Pixel mit $V - U > T_{vu}$ sind weiß, alle anderen schwarz.

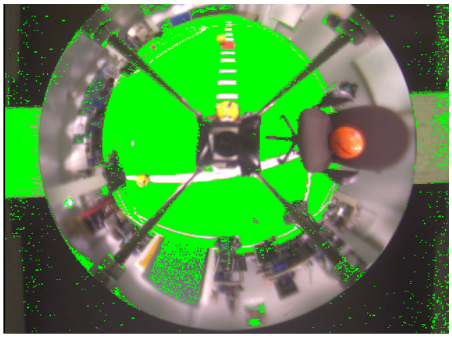


Abbildung 20: Tracking-Beispiel: Im Bild aus Abbildung 18 wurden alle als grüne Feldpixel erkannten Pixel mit einem kräftigen Grün markiert.

von Kreisen der bestmögliche ausgewählt. Wie diese Kreise im Beispiel aussehen, ist in Abbildung 22 zu sehen.

Als Ergebnis liefert das Tracking-Verfahren dann die Ballposition und den Ballradius. Abbildung 23 zeigt dies für das Beispiel.

Damit ist ein Durchlauf des Trackingverfahrens abgeschlossen.

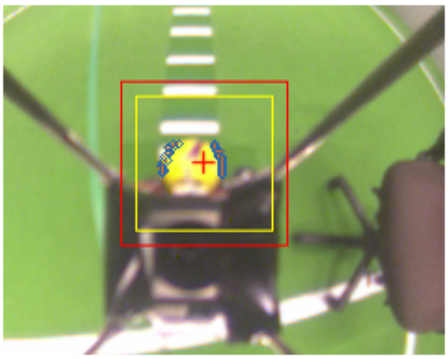


Abbildung 21: Tracking-Beispiel: Die im Bild aus Abbildung 18 erkannten Kantenpixel.

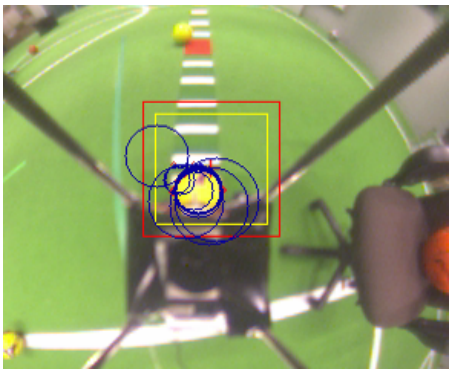


Abbildung 22: Tracking-Beispiel: Die mit den Kantenpixeln aus Abbildung 21 bestimmten möglichen Bälle (*Bemerkung:* andere Initialposition als bei den restlichen Beispielbildern).

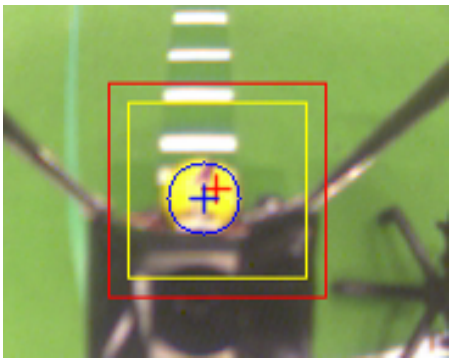


Abbildung 23: Tracking-Beispiel: Der gefundene Ball.

6. Einsatzszenario, Messungen und Ergebnisse

Nachdem nun ein Verfahren entwickelt wurde, dass das Auffinden des Balls in einem Kamerabild ausgehend von einer Initialposition erlaubt, soll dieses in der Praxis zum Tracking eingesetzt werden. Wie bereits beschrieben, soll es bei den Fußballrobotern des *1. RFC Stuttgart* verwendet werden, die im *RoboCup* in der *Middle Size League* spielen. Im Folgenden wird kurz beschrieben, wie diese Roboter aufgebaut sind, siehe dazu Unterabschnitt 6.1.

Anschließend wird gezeigt, wie das im vorherigen Abschnitt 5 entwickelte Tracking-Verfahren in das Software-Framework integriert wurde, das auf den Robotern eingesetzt wird, um diese zu steuern. Dies wird in Unterabschnitt 6.2 erklärt.

Mit den so konfigurierten Robotern wurden schließlich verschieden Messungen mit dem fertig entwickelten Verfahren durchgeführt, um dessen Leistungen zu dokumentieren. Diese Messungen und deren Ergebnisse werden in Unterabschnitt 6.3 erläutert.

6.1. Aufbau der Roboter

Die Fußballroboter des *1. RFC Stuttgart* für die *Middle Size League* des *RoboCups* sind rund einen Meter hoch und haben eine Grundfläche von etwa 50 cm × 50 cm. In Abbildung 24 ist ein solcher Roboter zu sehen. Die Roboter besitzen drei angetriebene Räder

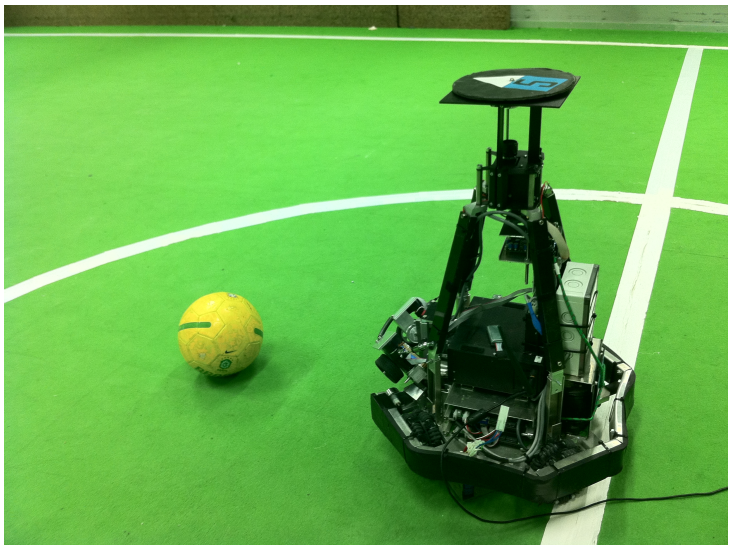


Abbildung 24: Ein Fußballroboter des *1. RFC Stuttgart*

mit quer installieren Rollen auf der Lauffläche. Dadurch können sie sich in jede Richtung drehen und gleichzeitig in jede Richtung fahren, es handelt sich also um einen omnidirektionalen Antrieb. Dieser kann den Roboter mit Geschwindigkeiten von bis zu 5 $^{m}/_{s}$ bewegen[1]. Zur Stromversorgung sind im Boden des Roboters Einschübe für mehrere Akku-Packs vorhanden.

Jeder Roboter besitzt außerdem ein sogenanntes *Dribbling Device*, eine Aussparung knapp über Spielfeldhöhe, teilweise mit zusätzlichen Rädern. Diese Vorrichtung dient dazu, dass der Roboter in der Lage ist, mit dem Ball zu dribbeln. Dazu fährt der Roboter den Ball so an, dass er in diese Aussparung gelegt wird, und durch die zwei Räder wird der Ball gedreht, damit er in der Aussparung liegen bleibt.

In der Mitte des Roboters ist der Steuerrechner angebracht. Dabei handelt es sich um handelsübliche PC-Komponenten, die in ein kleines Gehäuse gepackt wurden. Auf diesen läuft ein normales Linux-Betriebssystem. Er verfügt über konventionelle Anschlüsse für einen Bildschirm, Eingabegeräte und weitere Peripheriegeräte. Der Rechner hat außerdem ein WLAN-Modul, das die Kommunikation mit den anderen Robotern oder den Zugriff von außen auf den Steuerungsrechner ermöglicht. Am Gehäuse des Steuerrechners befindet sich ebenfalls ein Anschluss für ein Laptop-Netzteil, um den Roboter mit Strom zu versorgen, ohne geladene Akkus zu benötigen. Im mobilen Betrieb wird der Steuerrechner von einem eigenen Akku mit Strom versorgt[1].

Ganz oben am Roboter befindet sich die Panorama-Kamera. Sie verfügt über ein Auflösungsvermögen von 640×480 Pixeln und überträgt ihre Bilder in $UYVY$-Kodierung. Die Kamera ist senkrecht nach oben ausgerichtet und blickt auf einen Parabolspiegel direkt über sich, der sich auf einer Haltekonstruktion befindet. Diese Haltekonstruktion ist auch dafür verantwortlich, dass, wie in Unterabschnitt 4.1 auf Seite 14, erklärt, Teile des Bildes immer verdeckt sind. Die ältere Version dieser Konstruktion hatte vier Streben, inzwischen kommt man mit drei schmäleren Streben aus, dies spiegelt sich zum Teil in den Bildern in dieser Arbeit wider.

Dies ist natürlich nur ein sehr grober Überblick über die einzelnen Teile des Roboters. Weitere Details lassen sich auf der Webseite des *1. RFC Stuttgart* finden[1].

6.2. Integration des Tracking-Verfahrens

Um das entwickelte Tracking-Verfahren auf den Robotern einzusetzen, musste es in das Software-Framework eingebunden werden, das auf den Robotern eingesetzt wird. Ausgangsstand war, dass in jedem Kamerabild global nach Pixeln gesucht wurde, die die Ballbedingung erfüllen, also für die

$$V - U > T_{vu} \tag{67}$$

ist. Wenn von diesen Pixeln eine gewisse Anzahl benachbart sind, wird eine gewissen Region um diese gefundenen Pixel herum als „enthält potentiell einen Ball“ markiert und gespeichert. Das Tracking-Verfahren wurde dann so integriert, dass es für jede dieser Region aufgerufen wurde. Wurde dabei mindestens ein Ball gefunden, fand ein erneuter Aufruf der globalen Suche nach Ballpixeln erst wieder statt, wenn zu einem späteren Zeitpunkt kein Ball mehr gefunden wurde. Zusätzlich wurde diese globale Suche auch nach einer gewisse Anzahl von Kamerabildern erneut durchgeführt, um zu verhindern, dass das Tracking ewig nur ein und denselben Ball verfolgt.

Da das Verfahren für jede der Regionen aufgerufen wird, kann es natürlich vorkommen, dass mehrere Bälle gefunden werden. Wenn dieser Fall eintritt, muss ausgewählt werden, welchem der gefundenen Bälle gefolgt werden soll. Hierfür ist eine Qualitätsabschätzung

wichtig, die besagt, welcher Ball der am besten erkannte ist, der dann verfolgt werden soll. Die Qualitätsabschätzung ist auch wichtig, um zu ermitteln, wann erneut eine globale Ballsuche ausgeführt werden soll, also wann der Zeitpunkt erreicht ist, ab dem ein noch erkannter Ball als qualitativ nicht ausreichend klassifiziert wird.

Diese Qualitätsabschätzung wird im Software-Framework mit den vom Tracking-Verfahren gelieferten Informationen durchgeführt. Diese wurden unter Unterunterabschnitt 5.2.5, Information zur Ermittlung der Güte des gefundenen Balls, auf Seite 37, beschrieben. Der Gütewert eines Balls berechnet sich nach folgender Formel, wobei sich alle Pixelanzahlen auf die Pixel innerhalb des Kreises beziehen, der vom Tracking-Verfahren geliefert wird:

$$\eta_{\text{Ball}} = \frac{n_{\text{Ballpixel}} - n_{\text{Grünpixel}}}{n_{\text{Gesamtpixel}}} \cdot r_{\text{Ball}} \tag{68}$$

Mit dem Radius wird multipliziert, um größere und somit nähere Bälle höher zu gewichten. Außerdem wird bei der Berechnung der Anzahl von Gesamtpixeln nur der Teil der Pixel beachtet, der nicht ohnehin durch die Roboteraufbauten verdeckt ist, siehe dazu Abschnitt 5.1.1, Ignorieren von verdeckten Pixeln, auf Seite 21.

Der gefundene Ball mit dem höchsten Qualitätswert wird dann vom Tracking verfolgt, sprich dessen Position wird dem Tracking-Verfahren für den nächsten Rechenschritt als Initialwert vorgegeben.

Dieser Gütewert hat einen weiteren Nutzen, denn man kann damit die Qualität der Balldetektion in verschiedenen Entfernungen vergleichen. Basierend darauf wurden verschiedene Messungen durchgeführt, die im Folgenden beschrieben sind.

6.3. Messungen

6.3.1. Erkennen des Balls in verschiedenen Positionen

Eine wichtige Leistung, die das Tracking zu erbringen hat, ist das Erkennen von verschiedenen Arten von Bällen in verschiedener Entfernung. Bei der Auflösung der Kameras der Roboter und der Panorama-Konstruktion sollte eine Erkennung bis etwa $5\,m$ Entfernung möglich sein, dies würde den Anforderungen zum Einsatz im *RoboCup* genügen. Auf diese Entfernung ist nur noch ein Tracking des Balls möglich, die globale Suche nach Ballregionen erkennt Bälle in dieser Entfernung nicht mehr, da sonst die Gefahr zu groß wäre, Fehldetektionen zu unterliegen. Durch den bereits beschriebenen Qualitätswert, der sich nach der Formel

$$\eta_{\text{Ball}} = \frac{n_{\text{Ballpixel}} - n_{\text{Grünpixel}}}{n_{\text{Gesamtpixel}}} \cdot r_{\text{Ball}} \tag{69}$$

berechnet, ist es möglich, auch eine Aussage darüber zu treffen, wie gut ein Ball in einer gewissen Entfernung im Vergleich zu anderen Bällen und anderen Entfernungen erfasst wurde.

Um die Leistung des Trackingverfahrens zu überprüfen, wurde eine entsprechende Messung durchgeführt. Dazu wurde ein Roboter, auf dem das Tracking-Verfahren installiert

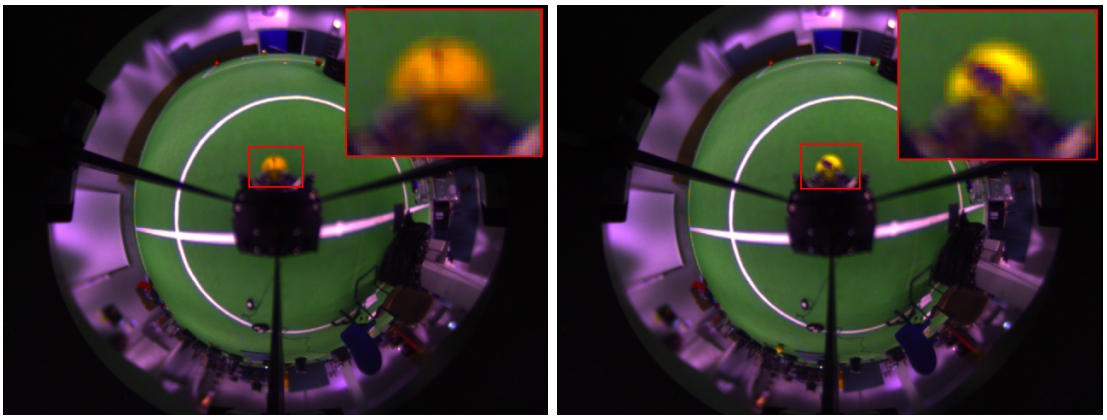
wurde, etwa in der Mitte des Spielfelds positioniert. Dann wurden zwei Bälle in verschiedenen Entfernungen von $0\,m$ bis $5\,m$ positioniert, und die Qualitätswerte aus 100 Kamerabildern gemittelt. Da die Kameras immer mit einem gewissen Rauschen behaftet sind, wurde einem Mittelwert der Vorzug gegenüber einer einzelnen Messung gegeben.

Die beiden Bälle, die verwendet wurden, sind in Abbildung 25 zu sehen. Dabei handelt es sich einmal um den sogenannten „Tournament“-Ball. Dieser ist gleichmäßig gelb und enthält nur wenig Muster. Seinen Namen trägt er, weil er oft für *RoboCup*-Wettbewerbe Verwendung findet. Zum Vergleich wurde ein zweiter Ball benutzt, der gelb mit lila Texturierung ist. Dieser ist durch die Mehrfarbigkeit grundsätzlich etwas schwieriger zu Erkennen, als der „Tournament“-Ball.

Abbildung 25: Die beiden für die folgende Messung verwendeten Bälle. Links der „Tournament“-Ball, der oft bei *RoboCup*-Wettbewerben verwendet wird. Rechts ein etwas stärker texturierter Ball.

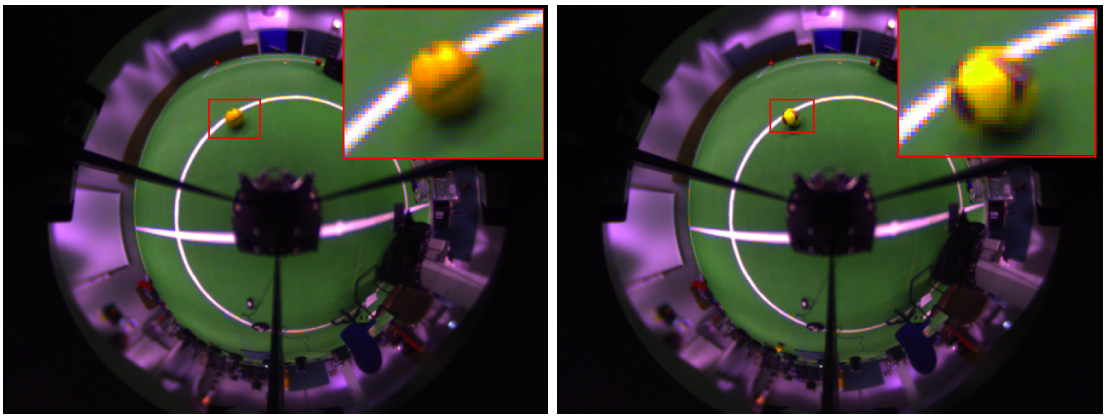
Die Messung wurde durchgeführt, in dem nacheinander die beiden Bälle in der entsprechenden Entfernung positioniert wurden. Dann wurden die Qualitätswerte für den Ball aus 100 Kamerabildern gemittelt. Anschließend wurde noch ein Bild aufgenommen, diese werden im Folgenden zur Illustration verwendet, um einen Eindruck dafür zu geben, wie groß Bälle in gewisser Entfernung im Bild der Panorama-Kamera des Roboters erscheinen.

Im Folgenden sind die Ergebnisse der einzelnen Messungen mit besagten Bildern aufgelistet. Danach folgt eine Zusammenfassung und Bewertung der Ergebnisse der Messungen.



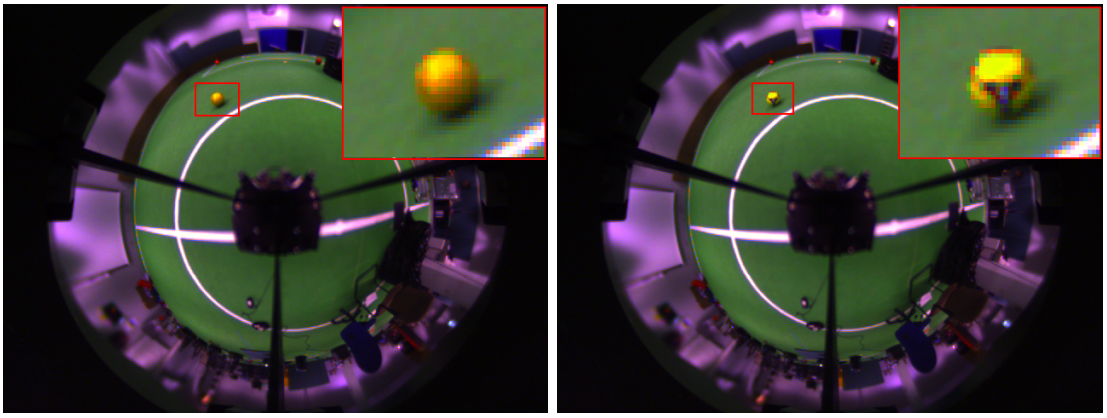
Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$0\,m$ *(direkt am Roboter)*	$12,1$	$11,2$

Abbildung 26: Messungen mit verschiedenen Bällen direkt am Roboter



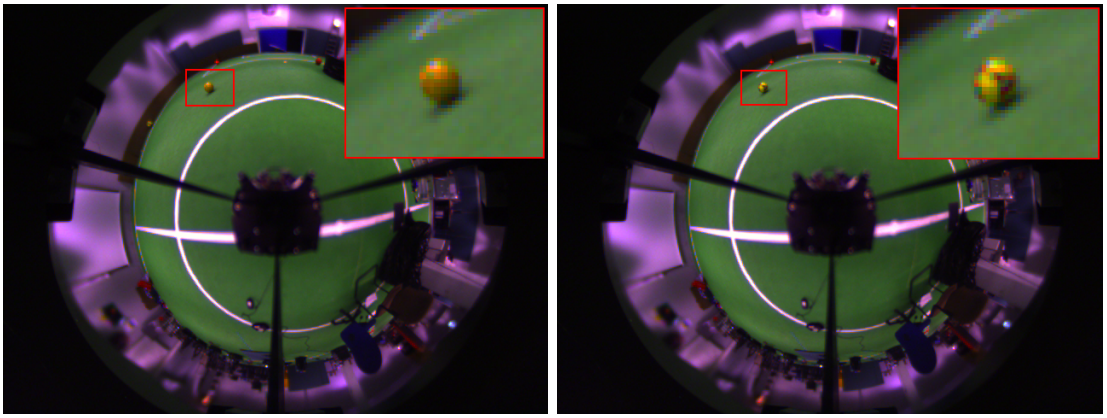
Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$1\,m$	$10,3$	$8,2$

Abbildung 27: Messungen mit verschiedenen Bällen in einem Meter Distanz zum Roboter



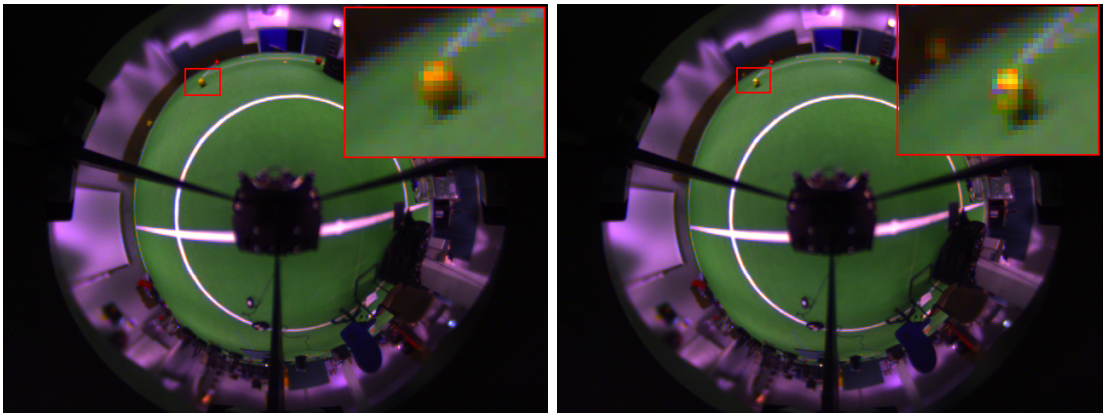
Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$2\,m$	$7,2$	$5,1$

Abbildung 28: Messungen mit verschiedenen Bällen in zwei Metern Distanz zum Roboter



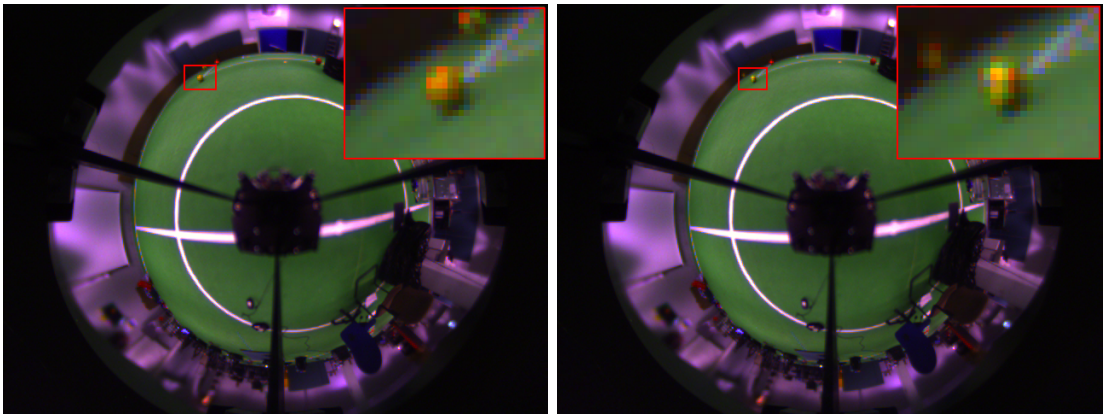
Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$3\,m$	$5,0$	$3,8$

Abbildung 29: Messungen mit verschiedenen Bällen in drei Metern Distanz zum Roboter



Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$4\,m$	$3,1$	$2,8$

Abbildung 30: Messungen mit verschiedenen Bällen in vier Metern Distanz zum Roboter



Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$5\,m$	$1,5$	—

Abbildung 31: Messungen mit verschiedenen Bällen in fünf Metern Distanz zum Roboter

Bewertung der Ergebnisse Die Messungen zeigen, dass das Tracking-Verfahren den geforderten Anforderungen genügt. Bis $5\,m$ wird der Wettbewerbs-Ball zuverlässig erkannt. Der texturierte Ball wird etwas schlechter gefunden, dieser wird ab einer Entfernung von etwa $4,5\,m$ nicht mehr erkannt. In Tabelle 5 sind die Ergebnisse der Messung nochmals zusammengefasst.

Entfernung	Ø Qualität *Tournament*-Ball	Ø Qualität texturierter Ball
$0\,m$	$12,1$	$11,2$
$1\,m$	$10,3$	$8,2$
$2\,m$	$7,2$	$5,1$
$3\,m$	$5,0$	$3,8$
$4\,m$	$3,1$	$2,8$
$5\,m$	$1,5$	—

Tabelle 5: Zusammenfassung der Messungen der Ballqualität des erkannten Balls in verschiedenen Entfernungen und mit verschiedenen Bällen.

Der Verlauf der Qualitätswerte zeigt, dass die Bälle wie gewünscht bewertet werden: Bälle, die sich näher am Roboter befinden, werden höher bewertet als Bälle, die weiter entfernt sind. Damit würde der Roboter im Zweifelsfall immer den näheren Ball wählen, würde er mehrere Bälle sehen.

Die Qualitätsberechnung geschieht wie beschrieben nach der folgenden Formel:

$$\eta_{\text{Ball}} = \frac{n_{\text{Ballpixel}} - n_{\text{Grünpixel}}}{n_{\text{Gesamtpixel}}} \cdot r_{\text{Ball}} \tag{70}$$

Diese enthält den Radius des gefundenen Balls als multiplikativen Faktor. Da die Qualität des erkannten Balls in etwa mit der Entfernung skaliert, ohne größere Einbrüche zu haben, lässt sich daraus schließen, dass weiter entfernte Bälle bis etwa $4 - 5\,m$ (je nach Ball) nahezu ebenso gut erkannt werden wie weniger weit entfernte.

6.3.2. Laufzeitmessungen

Wie bereits in Unterabschnitt 4.1, Tracking des Balls, auf Seite 14, beschrieben, stellt geringe Laufzeit ebenfalls eine wichtige Anforderung an das Tracking-Verfahren dar. Während der Entwicklung und Implementierung des Verfahrens war möglichst geringe Laufzeit daher immer oberste Priorität. Auch im praktischen Einsatz auf den Fußballrobotern ist die Laufzeit enorm wichtig. Sie muss gering sein, da ja zwischen jeweils zwei Kamerabildern nur ca. $30\,ms$ an Zeit zur Verfügung stehen. In dieser Zeit müssen unter Umständen mehrere Bereiche bei der Suche nach Bällen ausgewertet werden.

Um die Laufzeit des Verfahrens zu messen, wurden die *Google Performance Tools*[4]

[4] `http://code.google.com/p/google-perftools/`

verwendet. Mit diesen wurde die Gesamtlaufzeit der Bildverarbeitungskomponente des Roboter-Software-Frameworks gemessen und den Ergebnis-Graphen dann der Anteil entnommen, der auf das Tracking entfiel. Der Testaufbau war ähnlich wie in Unterunterabschnitt 6.3.1, Erkennen des Balls in verschiedenen Positionen. Der „Tournament"-Ball (Siehe Abbildung 25 auf Seite 45) wurde in verschiedenen Entfernungen zum Roboter positioniert, anschließend wurde die Laufzeit der BV-Komponente für 100 Kamerabilder gemessen.

Entfernung	Ø gemessene Laufzeit BV-Komponente	Anteil Tracking	Laufzeit Tracking
$0\,m$	$4,0\,ms$	$29,7\,\%$	$\sim 1,19\,ms$
$1\,m$	$3,8\,ms$	$28,6\,\%$	$\sim 1,09\,ms$
$2\,m$	$4,3\,ms$	$30,2\,\%$	$\sim 1,30\,ms$
$3\,m$	$4,6\,ms$	$32,2\,\%$	$\sim 1,48\,ms$
$4\,m$	$5,5\,ms$	$35,8\,\%$	$\sim 1,97\,ms$

Tabelle 6: Laufzeit des Trackings bei verschiedenen Entfernungen des Balls.

Wie man an den Ergebnissen der Messung in Tabelle 6 erkennen kann, liegt die Laufzeit auch bei weiter entfernten Bällen unter den $2\,ms$, die für die Fußballroboter des *1. RFC Stuttgart* gefordert wurde. Außerdem sieht man, dass die Laufzeit mit steigender Entfernung des Balles vom Roboter zunimmt. Dies liegt daran, dass bei weiter entferntem Ball länger nach Kantenpixeln gesucht wird, da die gewünschte Anzahl an Kantenpixeln nicht so schnell gefunden wird, wie bei einem im Bild großen, nahen Ball.

Wichtig zu bemerken ist, dass in der Messung darauf geachtet wurde, dass sich keine anderen Objekte auf dem Spielfeld befinden, die als Bälle erkannt werden, also insbesondere auch keine anderen Bälle. Die gemessene Laufzeit bezieht sich also auf exakt einen Schritt des Tracking-Verfahrens, also der Suche nach einem Ball in einer Region des Kamerabildes. Gibt es mehr Regionen mit potentiellen Bällen so wird das Tracking mehrfach aufgerufen, was sich in höherer Laufzeit niederschlägt.

6.3.3. Erkennen des Balls in ungünstigen Positionen

Die Erkennung von Bällen, die teilweise verdeckt sind, ist eine wichtige Anforderung an ein Tracking-Verfahren im *RoboCup*. Die Fähigkeit des vorgestellten Verfahrens, solche Bälle zu erkennen, soll exemplarisch an zwei wichtigen Positionen gezeigt werden. Bei diesen beiden Positionen handelt es sich einmal um die teilweise Verdeckung des Balls durch den Roboter selbst, wenn sich der Ball direkt am Roboter, also in der Dribbling-Position befindet. Dies wird in Abbildung 32 gezeigt.

Wie in der Abbildung zu sehen ist, ist der Ball etwa zur Hälfte verdeckt. Trotzdem wird seine Position und Größe korrekt erkannt.

Ein weiterer Fall ist die Verdeckung des Balls durch die Halterungen der Kameraauf-

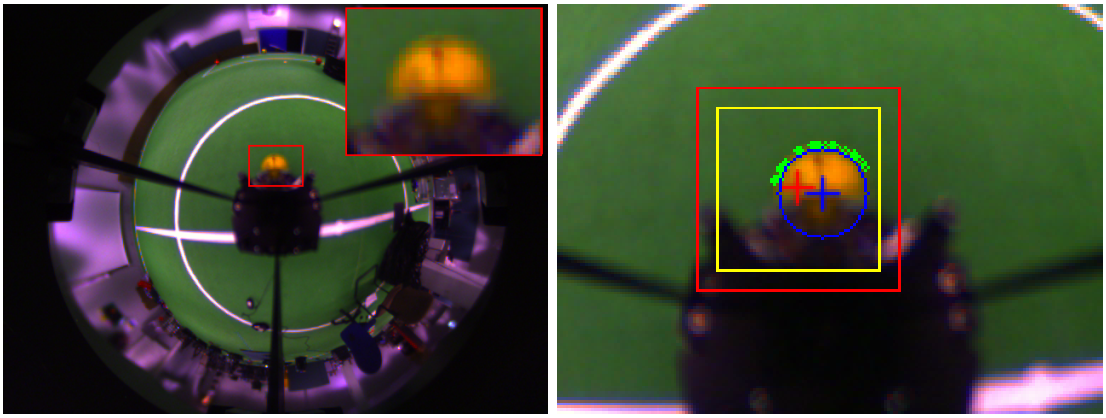


Abbildung 32: Links ist das Kamerabild des Roboters zu sehen, rechts der vom Verfahren erkannte Ball. In Grün die erkannten Ballkantenpixel.

bauten des Roboters. Diese wird in Abbildung 33 gezeigt. Auch hier ist zu sehen, dass

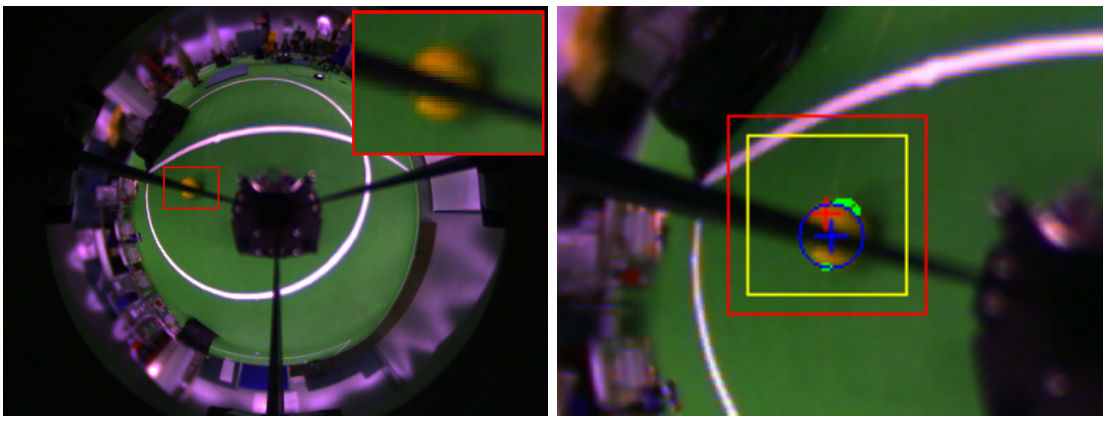


Abbildung 33: Links ist das Kamerabild des Roboters zu sehen, rechts der vom Verfahren erkannte Ball. In Grün die erkannten Ballkantenpixel.

der Ball korrekt erkannt wird, obwohl er zu einem nicht unerheblichen Teil durch die Strebe der Kamera-Aufbauten verdeckt wird.

7. Fazit und Ausblick

Wie sich durch die Messungen, die in Unterabschnitt 6.3 vorgestellt wurden, gezeigt hat, ist das entwickelte Tracking-Verfahren in der Lage, die Position des Balls schnell, exakt und robust zu ermitteln. Dabei genügt es den in Abschnitt 4, Motivation und Anforderungen, auf Seite 14, formulierten Anforderungen. Es ist schnell genug, um auf den Fußballrobotern des *1. RFC Stuttgart* im Spielbetrieb eingesetzt zu werden und seine Ergebnisse sind qualitativ ausreichend, um den Ball zuverlässig anzufahren beziehungsweise Spielzüge abhängig von der Position des Balls zu planen.

Grundsätzlich haben sich die getroffenen Annahmen und verwendeten Konzepte also als tauglich erwiesen, ein schnelles und robustes Tracking-Verfahren umzusetzen.

Mit dem beschriebenen Verfahren ist es möglich, den Ball in nahezu allen Positionen und bis in akzeptable Distanz zu erkennen. Dies hängt natürlich von der Güte und dem Auflösungsvermögen der verwendeten Kamera ab. Da die bei den Messungen eingesetzte Kamera mit einer Auflösung von 640×480 und einem eher ungünstigen Rauschverhalten aber eher am unteren Rand der Skala zu verorten ist, und damit bereits zufriedenstellende Ergebnisse erzielt werden können, ist das Verfahren nicht von besonders hochwertiger Hardware abhängig. Es kann auch mit niedrig aufgelösten und verrauschten Bildern umgehen, was für die Praxis durchaus relevant ist. Kameras sind teure Bauteile, und bieten somit ein großes Sparpotenzial, wenn man auf günstigere Modelle zurückgreifen kann und das Tracking trotzdem möglich ist.

Außerdem hat sich gezeigt, dass das entwickelte Verfahren auf der Referenzhardware, also den Steuerrechnern der Fußballroboter des *1. RFC Stuttgart*, die im durchschnittlichen Leistungsbereich zu verorten ist, in ausreichend geringer Zeit zu berechnen ist. Dies wurde insbesondere durch die strahlenförmige Abtastung des Bildes ausgehend vom zentralen Pol, die konsequente Verwendung von *Look-Up-Tables* und die Verwendung möglichst schnell zu berechnender mathematischer Formeln erreicht.

Für den Einsatz im *RoboCup* ist besonders wichtig, dass die Bälle auch erkannt werden, wenn sie beispielsweise teilweise verdeckt sind. Dies kann durch den Roboter selbst, seine Kamera-Aufbauten, oder durch andere Roboter auf dem Spielfeld der Fall sein. Auch hier hat sich gezeigt, dass das entwickelte Verfahren in der Lage ist, Bälle auch in ungünstigen Positionen zu erkennen, siehe dazu Unterunterabschnitt 6.3.3.

Im Folgenden wird auf zwei Aspekte genauer eingegangen. Dabei handelt es sich um die parameterlose Ballerkennung und die Verwendung der Log-Polar-Transformation. Diese beiden Eigenschaften bilden die Grundkonzepte für das entwickelte Tracking-Verfahren und deren Eignung sollte untersucht werden. Daher werden diese in eigenen Unterabschnitten genauer besprochen.

Daran anschließend folgt eine zusammenfassende Bewertung des hier gezeigten Verfahrens, eine Darstellung von Erweiterungsmöglichkeiten und zum Abschluss ein Ausblick auf die zukünftige Verwendung des Verfahrens.

7.1. Verwendung der parameterlosen Ballsuche

Beim Tracking eines Balls stellt sich das Problem, dass eine Möglichkeit gefunden werden muss, Ballpixel eindeutig zu identifizieren. Dazu müssen gewöhnlich Informationen wie Farbe oder Texturierung über den Ball zur Verfügung stehen. So ein Verfahren ist allerdings dann von der Kalibrierung abhängig und muss bei Änderung des zu verfolgenden Balles neu eingestellt werden.

Um diesen Nachteilen aus dem Weg zu gehen, wird beim in dieser Arbeit besprochenen Verfahren auf eine Parametrierung des Balls verzichtet. Stattdessen soll der Ball dadurch gefunden werden, dass er sich, salopp gesprochen, von der Umgebung abhebt, weil er im Gegensatz zu allen anderen Objekten auf dem Spielfeld *bunt* ist. Die Eigenschaft des *bunt*-Seins ist hier formal so definiert, dass ein Pixel dann als Ballpixel klassifiziert wird, wenn für seine Chrominanzkomponenten U und V

$$V - U > T_{vu} \tag{71}$$

gilt, wobei T_{vu} ein Schwellwert ist. Außerdem gibt es noch einige andere Möglichkeiten, wann Pixel als potentielle Ballpixel ausgeschlossen werden. Eine genaue Beschreibung der Ballpixelerkennung ist in Unterunterabschnitt 5.1.1, Detektion von Ballpixeln ohne Parameter, auf Seite 19, nachzulesen. Hierbei werden die Eigenschaften des YUV-Farbmodells genutzt, in dem auch die Bilder der Kamera ausgegeben werden. Dadurch ist auch keine unnötige Konvertierung notwendig.

Die parameterlose Detektion von Ballpixeln hat sich für das *RoboCup*-Szenario als geeignet erwiesen. Bis auf einige Fehlerkennungen wegen Kamerarauschens und an den Grenzen zu weißen Linien, liefert die Ballpixeldetektion absolut zufriedenstellende Ergebnisse. Echte Ballpixel werden zuverlässig erkannt.

Gleichzeitig ist die Pixelklassifizierung schnell zu berechnen und kommt mit den Informationen des Pixels selbst aus, ohne Informationen aus der Nachbarschaft oder aus Kalibrierungsparametern zu erhalten, von einigen Schwellwerten abgesehen.

Ein weiterer Vorteil ist, dass diese Art der Ballpixeldetektion relativ unabhängig von der konkreten Beleuchtung des Feldes ist, da die Luminanzkomponente Y nicht in die Detektion einfließt.

Es ist jedoch festzuhalten, dass diese Art der Ballerkennung nur im *RoboCup*-Szenario funktioniert. Dort ist fest vorgegeben, dass nur die Bälle auf dem Feld *bunt* sein dürfen, ansonsten gibt es festgelegte Farben für den Rasen und die Roboter. In einem Szenario, in dem diese Voraussetzung nicht gegeben ist, würde diese Art der Ballpixelerkennung nicht funktionieren. Dies zeigt sich auch daran, dass bunte Objekte an den Wänden des Spielfelds auch als mögliche Bälle eingestuft werden können. Diese werden aber dann durch die Qualitätsberechnung schnell verworfen und sind dadurch kein Problem.

Die Komponente der Ballpixelerkennung ist jedoch austauschbar, ohne dass dies Seiteneffekte auf andere Teile des Verfahrens hat. Sollte man das Tracking-Verfahren also einsetzen wollen, ohne dass die eben genannte Bedingung erfüllt ist, so müsste man einfach diesen Teil austauschen. Für die Verwendung im *RoboCup* hat sich diese Art der Ballerkennung aber als zweckmäßig erwiesen.

7.2. Die Log-Polar-Transformation beim Ball-Tracking

Das in dieser Arbeit vorgestellte Verfahren basiert zentral auf der Verwendung der Log-Polar-Transformation für das Balltracking. Dies ist einer der Ansätze, die untersucht werden sollen. Wie bereits im Vorherigen hervorgehoben, ist das Tracking-Verfahren in der Lage, die gestellten Anforderungen zu erfüllen. Grundsätzlich muss also festgehalten werden, dass die Verwendung der Log-Polar-Transformation für das Tracking geeignet ist.

Trotzdem hinterlässt die Log-Polar-Transformation ein zwiespältiges Bild. Zunächst erzeugt ihre Verwendung grundsätzlich einen gewissen Overhead, da nicht einfach über das betrachtete Kamerabild gelaufen werden kann, sondern die entsprechende Log-Polar-Position betrachtet werden muss. Dazu muss entweder das Bild komplett transformiert werden, was allerdings, wie in Unterunterabschnitt 5.1.2, Log-Polar-Transformation mit Look-Up-Table, auf Seite 22, beschrieben, für die Anforderungen beim Ball-Tracking im *RoboCup* ungeeignet ist, da dieser Vorgang zu rechenzeitintensiv ist. Im selben Unterabschnitt wird aber auch die Lösung für dieses Problem gegeben. Durch Verwendung einer *Look-Up-Table* (siehe auch Unterabschnitt A.3, Look-Up-Tables, auf Seite 60) kann dieser Overhead nahezu vollständig beseitigt werden. Aus Sicht der Laufzeit spricht damit nichts mehr gegen die Verwendung der Log-Polar-Transformation.

Der Grund für den zwiespältigen Eindruck ist ein anderer. Wie ebenfalls in Unterunterabschnitt 5.1.2, Log-Polar-Transformation mit Look-Up-Table, auf Seite 22, erläutert, kann der Bereich zwischen zwei Abtastpunkten bei der Abtastung des Bildes mit der Log-Polar-Transformation sehr groß werden. Gegensteuern lässt sich hier durch Anpassen des Skalierungsfaktors M, was neue Probleme aufwirft. Dabei kann eine Mehrfachabtastung entstehen, die dann wieder mit geeigneten Mitteln beseitigt werden muss. Auch das wurde in besagtem Unterabschnitt beschrieben. Bei den anschließenden Tests und Messungen auf dem Roboter wurden auch die Parameter des Verfahrens angepasst, unter anderem der Skalierungsfaktor M. Es hat sich dabei gezeigt, dass die Leistung des Verfahrens für höhere Werte von M besser wird, bis zu einem gewissen Punkt. Dieser Punkt ist dann erreicht, wenn M so groß ist, dass im Bereich des Balls und der unmittelbaren Umgebung einfach jeder Punkt abgetastet wird. Mit einer weiteren Erhöhung von M ist dann natürlich keine Verbesserung mehr zu erreichen, da keine zusätzlichen Abtastungen stattfinden. Diese werden, wie beschrieben, von der Mehrfachabtastungs-Prävention verhindert.

Wenn M aber nun soweit erhöht wurde, dass jeder Pixel entlang der Abtaststrahlen betrachtet wird, so ist die verwendete Log-Polar-Transformation in diesem Bereich absolut identisch zu gewöhnlichen Polarkoordinaten. Das bedeutet, in der Praxis hat sich gezeigt, dass unter Verwendung von Polarkoordinaten und Begrenzung des Suchbereichs auf einen Teil des Bildes die besten Ergebnisse zu erreichen sind, ohne die Laufzeit erheblich zu vergrößern. Genau das ist, was das hier entwickelte Verfahren in der praktischen Verwendung dann, bis auf einen kleineren Teil der Außenbereiche des Suchradius, durchführt.

Dass die Vorteile der Log-Polar-Transformation bei dem Problem des Balltrackings im *RoboCup* nicht zur Geltung kommen liegt daran, dass die Kanten eines Balls sehr

genau ermittelt werden müssen, um Position und Größe des Balls zu bestimmen. Hier ist die potentiell große Distanz zwischen zwei Abtastpunkten schon in geringer Entfernung zum Pol von Nachteil. Bereits um wenige Pixel falsch detektierte Ballkantenpixel können das Ergebnis deutlich verschlechtern, wenn es sich dabei um einen systematischen Fehler handelt.

Bei anderen Tracking- oder Bildverarbeitungsproblemen, bei denen eine genaue Kanten- oder Konturenerkennung nicht notwendig ist, sind die Auswirkungen der Vorteile aber durchaus denkbar. Insbesondere für eine Feature-Erkennung erscheint das Log-Polar-Bild geeignet. Im Bereich Objekt-Erkennung lassen sich also sicher interessante Ergebnisse erzielen, und von der Verwendung der Log-Polar-Transformation in diesem Kontext ist nicht grundsätzlich abzuraten.

Für das spezielle Problem, das Ball-Tracking im *RoboCup*, ist man allerdings mit gewöhnlichen Polarkoordinaten genauso gut bedient und spart sich bei der Implementierung die etwas komplexeren Transformationsregeln und einen Teil der Mehrfachabtastungs-Prävention. Festzuhalten bleibt aber, dass das hier vorgestellte Verfahren bei entsprechender Parametereinstellung absolut äquivalent zur Verwendung von Polarkoordinaten ist, insbesondere was die Laufzeit betrifft. Das liegt daran, dass alle entstehende Redundanz durch die Präventionsverfahren beseitigt wird, diese müssen aber nur einmalig zum Programmstart ausgerechnet werden und nicht mehr während das eigentliche Tracking durchgeführt wird.

7.3. Bewertung des Verfahrens

In den beiden vorangegangenen Unterabschnitten wurde bereits dargelegt, dass das Verfahren geeignet ist, die daran gestellten Anforderungen zu erfüllen. Es ist in der Lage, in relativ kurzer Rechenzeit zufriedenstellende Ergebnisse zu liefern. Dabei ist es, wie sich in den Messungen gezeigt hat, robust gegenüber den im *RoboCup* auftretenden Problemen, wie beispielsweise der Verdeckung des Balls durch den Roboter. Dabei ist es weder auf außergewöhnlich leistungsstarke Hardware noch auf qualitativ hochwertige Kamerabilder angewiesen, sondern kann auch mit der verwendeten, durchschnittlich leistungsfähigen Technik umgehen.

Grundsätzlich ist das Verfahren relativ einfach zu implementieren, wobei seine Geschwindigkeit erheblich von der eigentlichen Implementierung abhängt. Für die in den Messungen verwendete Umsetzung wurde erhebliche Optimierungsarbeit geleistet, um die Rechenzeit so weit wie möglich zu senken. Insbesondere durch die konsequente Verwendung von *Look-Up-Tables* konnte die Laufzeit gering gehalten werden.

Als weiterer Vorteil ergibt sich, dass eine Umsetzung des Verfahrens ohne Abhängigkeiten nach außen auskommt, von *OpenCV* abgesehen. Sonst wurden neben den *C++*-Standardbibliotheken keine weiteren Bibliotheken genutzt. Auch *OpenCV* lässt sich problemlos ersetzen, das Verfahren selbst ist von den verwendeten Datenstrukturen unabhängig.

Für den Einsatz des Verfahrens im *RoboCup* ist besonders die Tatsache interessant, dass der Ball selbst nicht konfiguriert werden muss oder dessen Parameter gelernt werden müssen, sondern dass durch die parameterlose Ballpixeldetektion jeder Ball erkannt wird, der die gesetzten Bedingungen erfüllt. Auch wenn sich die Log-Polar-Transformation

selbst nicht als unbedingt für die Lösung dieses konkreten Tracking-Problems als hilfreich erwiesen hat, so ist das für das Verfahren kein Nachteil, da das Verhalten normaler Polarkoordinaten durch Parametrierung hergestellt werden kann.

Insgesamt steht mit dem entwickelten Verfahren also eine schnelle und relativ robuste Methode zum Ball-Tracking zur Verfügung.

Natürlich muss aber auch festgehalten werden, dass bei der Entwicklung und Implementierung des Verfahrens stets Kompromisse zwischen Laufzeit und Güte der Ergebnisse gemacht wurden. Die Qualität der Ergebnisse steht letztlich hinter anderen Suchverfahren, die auf den Fußballrobotern verwendet wurden, zurück, insbesondere, was die Erkennung von Ballpixeln betrifft. Dafür ist das Verfahren aber um ein mehrfaches schneller, und kann somit auch für verschiedene Suchregionen aufgerufen werden, ohne dass zu viel Rechenzeit dafür benötigt werden würde.

Die Zeit, die zur Entwicklung des Verfahrens zur Verfügung stand, war auf den Zeitrahmen einer Diplomarbeit begrenzt. Das Verfahren bietet aber in mehreren Richtungen erhebliches Verbesserungs- und Erweiterungspotential. Unter anderem darauf wird im folgenden Unterabschnitt 7.4 eingegangen.

7.4. Weiterentwicklung

Das Verfahren bietet an mehreren Stellen Verbesserungspotential, das aufgrund der begrenzten Entwicklungszeit ungenutzt blieb.

Zum Beispiel ist ein bekannter Parameter des Balls dessen echte Größe. Bei einem kalibrierten Roboter lässt sich außerdem die Entfernung eines Bildpunktes vom Roboter bestimmen. Mit diesen beiden Informationen könnte man eine *Look-Up-Table* aufstellen, die für jede Position im Bild den Radius des Balls, wenn er auf dem Boden liegt, liefert. Dieser Radius könnte dann benutzt werden, um die Bestimmung des besten Kreises verbessern, indem Kreise bevorzugt werden würden, deren Radius nah an diesem berechneten Radius liegt.

Verbesserungspotenzial besteht auch in der Detektion von Ballkantenpixeln. Anstatt hier nur zu ermitteln, ob ein Pixel zum Ball gehört oder nicht, und dann später noch zu prüfen, ob es grün ist, könnte man hier eine echte Pixelklassifizierung durchführen. Dabei könnte man die Pixel direkt in mehrere Klassen wie *Ball*, *grünes Feldpixel*, *weißes Feldpixel*, *Hindernis*, usw. einteilen.

Es wäre außerdem denkbar, die Qualitätsprüfung der Ergebnisse direkt in das Tracking-Verfahren zu verlagern. Dort stehen alle Informationen, die gewonnen wurden, zur Verfügung, und nicht nur der Teil, der in Variablen zurückgegeben wird. Damit ließe sich eine wesentlich feinere Qualitätsbestimmung durchführen, deren Ergebnisse direkt zur Optimierung der Ausgabe genutzt werden können. Hier könnten ungeeignete Kreise frühzeitig ausgeschlossen und für die Bestimmung des besten Kreises gar nicht in Betracht gezogen werden.

7.5. Ausblick

Aufgrund der Tatsache, dass das Verfahren, beziehungsweise seine Implementierung, bei den Fußballrobotern des *1. RFC Stuttgart* eingesetzt wird, ist eine Erweiterung und Weiterentwicklung des Verfahrens sehr wahrscheinlich. Beim *1. RFC Stuttgart* wird ständig aktiv an der Robotersoftware gearbeitet und in Wettbewerben neu gewonnene Erkenntnisse werden zur Verbesserung der Verfahren eingesetzt. Die Implementierung des beschriebenen Tracking-Verfahrens wurde in diese Software integriert und wird somit von den Mitarbeitern der beteiligten Institute weiterentwickelt werden.

Wie beschrieben sind die Roboter des *1. RFC Stuttgart* mit relativ niedrig auflösenden und relativ stark verrauschten Kameras ausgestattet, das lässt sich auch an den Beispielbildern in dieser Arbeit sehen. Diese sollen aber in naher Zukunft durch höherauflösende, hochwertigere Kameras ersetzt werden. Durch den Einsatz dieser Kameras sollten sich die Leistungen des Verfahrens erheblich verbessern lassen, insbesondere, was die Entfernung betrifft, in der Bälle zuverlässig erkannt werden.

Für die weitere Verwendung der gewonnenen Erkenntnisse ist nochmals festzuhalten, dass das hier entwickelte Verfahren im Prinzip aus zwei Teilen besteht, die unabhängig voneinander verwendet werden können. Zum einen die Detektion von Ballpixeln und, darauf aufbauend, die Erkennung von Ballkanten und zum zweiten die Berechnung von Mittelpunkt und Größe des Balls aus den Kantenpositionen. Natürlich benötigt die Positionsbestimmung als Eingabe die bestimmten Ballkantenpositionen, jedoch könnte man das Detektionsverfahren durch ein beliebiges anderes ersetzen, so lange es diese Positionen liefert. Umgekehrt kann natürlich auch die Ausgabe des Detektionsverfahren für eine andere Positionsbestimmung verwendet werden.

Das Verfahren wurde im ersten Teil des Jahres 2011 entwickelt und einige Wochen vor der *RoboCup*-Veranstaltung in Istanbul im Juli 2011 fertiggestellt. Es wurde auf den Fußballrobotern des *1. RFC Stuttgart* implementiert und zeigte in Tests und Simulationen gute Leistungen. Daher soll es auf dieser Veranstaltung das erste Mal in der Praxis in einer tatsächlichen Spielsituation eingesetzt werden. Wie gut die Implementierung mit allen Eventualitäten und unvorhergesehenen Situationen zurecht kommt, wird sich dort zeigen.

A. Anhang: Weitere Grundlagen

A.1. RoboCup

Beim RoboCup handelt es sich um eine wissenschaftliche Initiative, die gegründet wurde, um zur Verbesserung von intelligenten Robotern beizutragen. Die Organisation selbst definiert sich auf ihrer Website wie folgt[9]:

> RoboCup is an international scientific initiative with the goal to advance the state of the art of intelligent robots. When established in 1997, the original mission was to field a team of robots capable of winning against the human soccer World Cup champions by 2050. While that mission remains, RoboCup has since expanded into other relevant application domains based on the needs of modern society.
>
> (`http://www.robocup.org/about-robocup/`)

Unter dem Dach des *RoboCups* werden verschiedene Wettbewerbe durchgeführt, in denen sich intelligente Roboter in verschiedenen Bereichen messen müssen. Einer davon, die auch die ursprüngliche Disziplin des *RoboCups* ist, ist der Roboterfußball. In verschiedenen Ligen treten hier Teams von – je nach Liga – autonomen Robotern gegeneinander an. Der Roboterfußball ist für die Robotik eine interessante Spielwiese. Um Fußball spielen zu können, müssen die Roboter verschiedenste Aufgaben erfüllen können, die alle auch für Anwendungen in anderen Bereichen eine wichtige Rolle spielen. Einige Beispiele für Fähigkeiten, die im Roboterfußball von Bedeutung sind:

- Erfassen der **dynamischen** Umgebung, der Gegner und des Spielgeräts durch Sensoren, insbesondere Kameras (→ Objekterkennung und -verfolgung)
- Erfassen des Zustands des Roboters selbst
- Selbstlokalisierung auf dem Spielfeld
- Schnelle und flexible Fortbewegung mit – je nach Liga – verschiedenen Antrieben
- Kooperation und Kommunikation zwischen mehreren, autonomen Robotern
- Taktisches und strategisches Vorgehen zum Erzielen eigener beziehungsweise Verhindern gegnerischer Tore (→Planung)

Roboterfußball ist also nicht nur Selbstzweck, alle Entwicklungen können auch für praxisrelevante Aufgaben Verwendung finden. Trotzdem bietet der Roboterfußball ein festgelegtes Szenario, in dem es möglich ist, neue Entwicklungen gezielt und im Vergleich mit anderen Lösungen auszuprobieren. Aus den Ergebnissen dieser Tests können dann natürlich Rückschlüsse darauf gezogen werden, ob ein im *RoboCup* eingesetztes Verfahren, eine Technik oder ein Bauteil auch in echten Anwendungen funktionieren würde.

Anmerkung: In dieser Arbeit werden RoboCup *und Roboterfußball gleichbedeutend verwendet, auch wenn der* RoboCup *wie beschrieben inzwischen weitere Disziplinen umfasst.*

Middle Size League Eine der Ligen des *RoboCups* ist die *Middle Size League*, in der das in dieser Arbeit entwickelte Verfahren zum Einsatz kommen soll. Daher wurden auch alle Messungen mit Robotern des *1. RFC Stuttgart*[5] der Universität Stuttgart durchgeführt, der an den Wettbewerben dieser Liga teilnimmt. Für die *Middle Size League* gelten folgende Regularien[9]:

> Middle-sized robots of no more than 50 cm diameter play soccer in teams of up to 6 robots with regular size FIFA soccer ball on a field similar to a scaled human soccer field. All sensors are on-board. Robots can use wireless networking to communicate. The research focus is on full autonomy and cooperation at plan and perception levels.

A.2. Die OpenCV-Bibliothek

OpenCV ist eine freie und offene Bibliothek für Bildverarbeitung und maschinelles Sehen (engl. *Computer Vision*), insbesondere in Echtzeit[2]. Die Bibliothek wird unter einer BSD-Lizenz[6] veröffentlicht, und ist damit sowohl für den akademischen als auch für den kommerziellen Gebrauch vollständig kostenlos nutzbar. *OpenCV* wurde ursprünglich von *Intel* entwickelt, inzwischen liegen die Entwicklungstätigkeiten jedoch bei dem Robotertechnik-Unternehmen *Willow Garage*. Die zur Erstellungszeit dieser Arbeit aktuelle Version trägt die Versionsnummer `2.2`, auf diese Version beziehen sich alle Aussagen.

Ursprünglich wurde *OpenCV* vollständig in der Programmiersprache *C* entwickelt, inzwischen gibt es jedoch ein nahezu vollständiges *C++*-Interface und auch alle weitere Entwicklung wird in dieser Sprache stattfinden[2]. Es gibt verschiedene Wrapper in anderen Sprachen, die jedoch zum Großteil nicht immer alle Funktionen der Original-Bibliothek abbilden. *OpenCV* ist unter nahezu allen gängigen Desktop- und mobilen Betriebssystemen verfügbar[2].

OpenCV stellt eine Vielzahl von Funktionen und Algorithmen bereit. Als Basis für diese Algorithmen werden in der *opencv-core*-Bibliothek grundlegende Datenstrukturen definiert, mit denen beispielsweise Bilder, Punkte oder Vektoren innerhalb von *OpenCV* verwaltet werden können. Das Auslesen von Bildern beziehungsweise Bildfolgen einer Kamera ist ebenfalls Teil von *OpenCV*. Auf Bildern, die in den *OpenCV*-Datenstrukturen vorliegen, können dann eine Vielzahl von mitgelieferten Operationen durchgeführt werden, beispielsweise einfache wie die Invertierung oder Konvertierung in einen anderen Farbraum oder Filteroperationen wie Sobel[7] oder Canny[8]. Ebenfalls gibt es Zeichenoperationen, mit denen zum Beispiel Linien oder Kreise in die Bilder gezeichnet werden können. *OpenCV* deckt aber auch andere Bereiche der Bildverarbeitung und des maschinellen Sehens ab, wie zum Beispiel Objekt- oder Gesichts-Erkennung. Außerdem beinhaltet es eine Bibliothek für maschinelles Lernen, mit der einige der anderen Verfahren unterstützt und verbessert werden können[2]. Eine einfache grafische Oberfläche ist ebenfalls Teil von *OpenCV*, mit der beispielsweise Bilder angezeigt werden können.

[5] `http://robocup.informatik.uni-stuttgart.de/rfc/www/`
[6] `http://opensource.org/licenses/bsd-license.php`
[7] `http://en.wikipedia.org/wiki/Sobel_operator`
[8] `http://en.wikipedia.org/wiki/Canny_edge_detector`

Zur Implementierung aller Bildverarbeitungsaufgaben, auf die in dieser Arbeit eingegangen wird, wurden die *OpenCV*-Datenstrukturen und ein Teil der *OpenCV*-Funktionen genutzt.

A.3. Look-Up-Tables

Unter einer sogenannten *Look-Up-Table* versteht man eine Datenstruktur, die eine Funktion abbildet. Die *Look-Up-Table* liefert also für eine Belegung von Eingangsparametern einen Wert oder eine Datenstruktur zurück. Diese *Look-Up-Table* kann dazu einmal bei Programmstart initialisiert oder aus Dateien oder einer Datenbank geladen werden. Der große Vorteil ist, dass die Ergebnisse der Funktion nun direkt mit einem einzigen *Look-Up* aus der *Look-Up-Table* gelesen werden können, ohne dass sie erst dynamisch berechnet werden müssen. Dies bedeutet, bei nicht-trivialen Funktionen, einen erheblichen Geschwindigkeitsvorteil.

Je nach Definitionsbereich der abgebildeten Funktion kann so eine *Look-Up-Table* sehr groß werden und somit viel Speicher belegen, dies muss bei deren Verwendung beachtet werden. Außerdem dauert das Anlegen der *Look-Up-Table* unter Umständen sehr lange, was den Programmstart sehr stark verlangsamen kann. Hier bietet es sich an, die *Look-Up-Table* in eine Datenbank oder Dateien auszulagern und sie dann beim Programmstart nur noch zu Laden und nicht mehr neu zu berechnen. Diese Methode ist auch dann notwendig, wenn die *Look-Up-Table* keine mathematische Funktion abbildet, die dynamisch für alle Belegungen von Eingangsparametern berechnet werden kann.

Die *Look-Up-Table* ist oft als *Array* umgesetzt, jedoch nicht an diese Datenstruktur gebunden. Jede Datenstruktur mit schnellem *Random Access*[9] ist prinzipiell zum Umsetzen einer *Look-Up-Table* geeignet.

A.4. YUV-Farbmodell

Beim YUV-Farbmodell wird die Farbe eines Pixels durch seine Helligkeit (Luminanz) Y und die Farbanteile (Chrominanzwerte) U und V ausgedrückt. U repräsentiert hierbei den Blauanteil, V den Rotanteil der Farbinformation.

Da das menschliche Auge auf Änderungen der Helligkeit wesentlich empfindlicher reagiert, als auf Änderungen der Farbe, und diese Informationen im YUV-Farbmodell getrennt vorliegen, kann man die Werte komprimieren, ohne dass damit ein wesentlicher Qualitätsverlust verbunden ist. So werden üblicherweise für jeweils 2 oder 4 Pixel gemeinsame Chrominanzwerte U und V gespeichert, aber ein Helligkeitswert Y für jedes einzelne Pixel. Diese Art der Komprimierung wird dann als „4:2:2“- beziehungsweise „4:1:1“-Kodierung bezeichnet. Verwendet man zum Beispiel die „4:2:2“-Kodierung, so kann man ein dreikanaliges RGB-Bild in zwei Kanälen abspeichern, wobei im einen Kanal die Helligkeitswerte Y und im anderen Kanal abwechselnd U und V gespeichert werden. Daher wird diese Kodierung auch als „UYVY“-Kodierung bezeichnet. Auf Grund dieser Eigenschaften ist das YUV-Modell für die Ausgabe von Videokameras beliebt.

Abbildung 34 zeigt ein Beispiel für die einzelnen Komponenten eines Bildes.

[9] `http://en.wikipedia.org/wiki/Random_access`

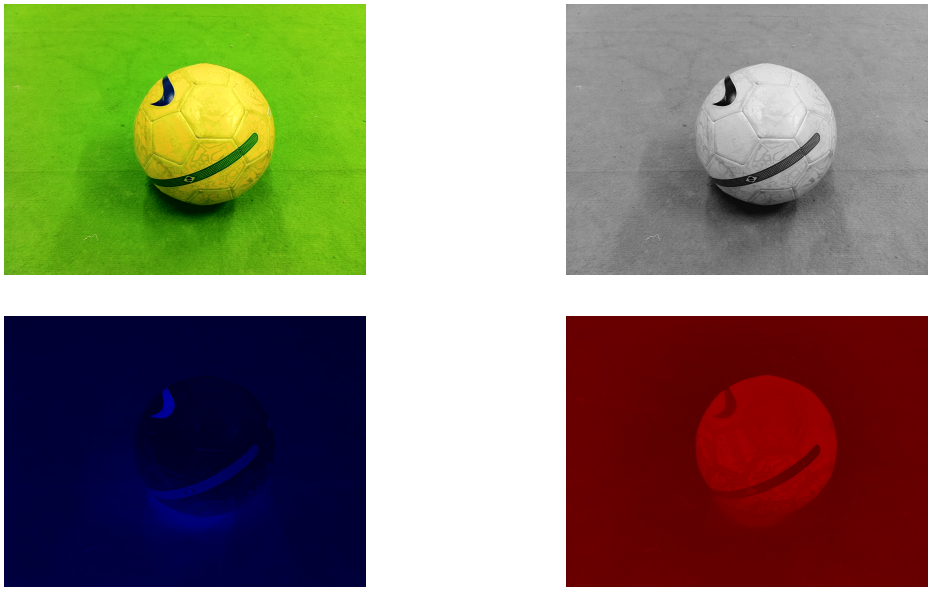


Abbildung 34: Ein Beispielbild: links oben das Original RGB-Bild, rechts oben die Helligkeit Y als Graustufenbild, unten links die Chrominanz-Komponente U und rechts unten die Chrominanz-Komponente V

A.5. Omnidirektionale Kameras

Unter einer omnidirektionalen Kamera versteht man eine Kamera, die in der Lage ist, Bilder aufzunehmen, die die gesamte Sphäre um die Kamera herum einnehmen. Um dies zu erreichen, wird ein komplexes System aus zwei Spiegeln verwendet. Die Kamera blickt meist vertikal nach oben direkt auf einen nach oben gewölbten Hohlspiegel, der die Kamerasicht dann auf einen zweiten Spiegel umlenkt, der die Kamera umgibt. Dieser lenkt die Kamerasicht dann in die gesamte Sphäre.

So eine Konstruktion besitzt einen blinden Fleck, den sie konstruktionsbedingt nicht sehen kann. Dabei handelt es sich um die Bereiche der Sphäre, die durch den oberen Hohlspiegel verdeckt sind.

Neben diesen echten omnidirektionalen Kameras gibt es auch so genannte Panorama-Kamera-Systeme, die nur eine Hemisphäre sehen können. Deren Konstruktion ist wesentlich einfacher, hierfür muss lediglich ein einzelner nach unten gewölbter Hohlspiegel über der Kamera positioniert werden. Solche Kamerasysteme werden zum Teil ebenfalls als „omnidirektional“ bezeichnet.

Literatur

[1] 1. RFC Stuttgart: Offizielle Webseite. 2011
`http://robocup.informatik.uni-stuttgart.de/rfc/www/`

[2] Bradski, Gary; Kaehler, Adrian: *Learning OpenCV: Computer Vision with the OpenCV Library*. Sebastopol: O'Reilly Media 2008

[3] Capurro, C.; Panerai, F.; Sandini, G.: Dynamic vergence using log-polar images. *International Journal of Computer Vision 1.24*, 79–94 (1997)

[4] Choi, Il; Yoon, Jong-Gun; Lee, Young-Beum; Chien, Sung-Il: Stereo system for tracking moving object using log-polar transformation and zero disparity filtering. In: Petkov, Nicolai; Westenberg, Michel (eds.): *Computer Analysis of Images and Patterns CAIP*, Groningen 2003 (Lecture Notes in Computer Science 2756, 182–189), Berlin Heidelberg: Springer 2003

[5] Daniilidis, Konstantinos: Attentive visual motion processing: Computations in the log-polar plane. In: Kropatsch, Walter G.; Klette, Reinhard; Solina, Franc (eds.): *Proceedings of the 7th TFCV on Theoretical Foundations of Computer Vision*, Dagstuhl 1994 (Computing Supplement 11, 1–20), London UK: Springer 1996

[6] Hotta, K.; Kurita, T.; Mishima, T.: Scale invariant face detection method using higher-order local autocorrelation features extracted from log-polar image. In: *Proceedings: Third IEEE International Conference on Automatic Face and Gesture Recognition*, Nara (Japan) 1998 (FG'98, 70–75), New York City: Institute of Electrical & Electronics Engineering 1998

[7] Metta, Giorgio; Gasteratos, Antonios; Sandini, Gulio: Learning to track colored objects with log-polar vision. *Mechatronics* 14.9, 989–1006 (2004)

[8] Okajima, N.; Nitta, N.; Mitsuhashi, W.: Motion estimation and target tracking in the log-polar geometry. *Seventeenth Sensor Symposium.* Kawasaki (Japan), 2000

[9] The RoboCup Federation: Offizielle Webseite. 2011
`http://www.robocup.org/`

[10] Thunuguntla, Saikiran S.: *Object Tracking using Log-Polar Transformation*. Master's thesis, Louisiana State University and Agricultural and Mechanical College. 2005

[11] Traver, V.; Pla, Filiberto: An optimization approach for translational motion estimation in log-polar domain. In: Skarbek, Wladyslaw ed.: *Computer Analysis of Images and Patterns CAIP*, Warschau 2001 (Lecture Notes in Computer Science 2124, 365–373), Berlin Heidelberg: Springer 2001

[12] Weiman, C. F.; Juday, R. D.: Tracking algorithms using log-polar-mapped image coordinates. In: D. P. Casasent ed.: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, Philadelphia 1989 (SPIE Conference Series 1192, Intelligent Robots and Computer Vision VIII: Algorithms and Techniques, 843–853), Bellingham: International Society for Optics and Photonics 1990

[13] Willow Garage Inc.: *OpenCV Reference Manual Version 2.2*. 2011

[14] Wolberg, G.; Zokai, S.: Robust image registration using log-polar transform. In: *Proceedings: 2000 International Conference on Image Processing*, Vancouver 2000 (Image Processing ICIP 2000 International Conference 4 Volume Set, 493–496), New York City: Institute of Electrical & Electronics Engineering 2000

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Florian Burger)