Technische Universität Hamburg-Harburg
Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Formbasierte Ballerkennung in Echtzeit auf dem humanoiden NAO-Robotiksystem

Bachelorarbeit

Thomas Schattschneider

23. November 2015

Eidesstattliche Erklärung

Ich, Thomas Schattschneider, geboren am 25.08.1991 in Hamburg, versichere hiermit an Eides statt, dass ich diese von mir bei der Technischen Universität Hamburg-Harburg (TUHH) vorgelegte Bachelorarbeit selbstständig verfasst habe. Ich habe ausschließlich die angegebenen Quellen und Hilfsmittel benutzt.

Ort und Datum	Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Roboter sind heutzutage aus der Industrie moderner Gesellschaften nicht mehr wegzudenken. Sie spielen nicht nur in Produktionsprozessen eine entscheidende Rolle, sondern auch bei logistischen Aufgaben im kleinen sowie großen Maßstab. Überall dort, wo Arbeitsabläufe mit immer gleichen Schritten oder unter Einhaltung strikter Regeln und Bedingungen stattfinden, ist der Einsatz von Robotern zur potentiellen Steigerung der Produktivität denkbar. Besonders die Nutzung von Robotern für die hocheffiziente Produktion von großen Stückzahlen gleichbleibender Bauteile ist längst keine Neuheit mehr. Andere Beispiele sind Maschinen wie Roboterarm-Manipulatoren am Fließband zur Fertigung von Autoteilen oder auch diverse Anlagen für den Transport und die Lagerung von Schiffscontainern an Hafen-Terminals. Die Einbindung von Robotern in den Betrieb scheint für viele Anlagenbetreiber erstrebenswert, da Sie gegenüber menschlichen Arbeiten dauerhaften Einsatz bei gesteigerter Produktivität versprechen. Hierdurch entstehen aktuell Arbeitssituationen, in denen Roboter und Menschen parallel in einem Betrieb arbeiten.

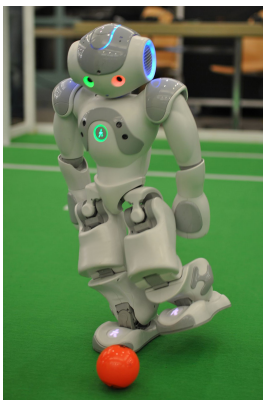
Doch was bedeutet das Wort *Roboter* genau? Die Erwähnung von Robotern erweckt wohl bei den meisten Menschen eine spontane Vorstellung darüber, welche Art von Maschinen hiermit gemeint sind. Tatsächlich gibt es jedoch unterschiedliche Definitionen, von denen hier zwei genannt werden sollen. Zum Einen wäre da die Definition als „(mit Greifarmen ausgerüsteter) Automat, der ferngesteuert oder nach Sensorsignalen bzw. einprogrammierten Befehlsfolgen anstelle eines Menschen bestimmte mechanische Tätigkeiten verrichtet“, was dem vorhergehend beschriebenen industriellem Einsatz entspricht, und zum Anderen als „(der menschlichen Gestalt nachgebildete) Apparatur, die bestimmte Funktionen eines Menschen ausführen kann; Maschinenmensch“ [Dud15]. Besonders jene Art von Robotern, welche menschenähnlich handeln, strahlt auch außerhalb der Industrie eine für viele Personen faszinierende Anziehungskraft aus. Schon seit Jahrzehnten erträumen visionäre Ingenieure die Inbetriebnahme von autonomen Robotern in vielfältigen Bereichen des menschlichen Gesellschaftslebens. Wurden diese Ideen mit dem Aufkommen der Automatisierungstechnik zunächst meist nur in Litera-

tur und Film festgehalten, kann inzwischen beobachtet werden, dass sich Vorstellung und Wirklichkeit mit zunehmenden Anstrengungen auf dem Gebiet der Robotik immer weiter einander annähern.

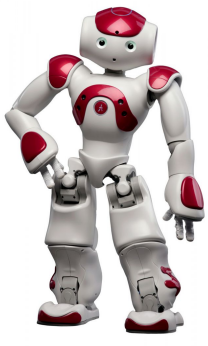
Hier kommt das *NAO-Robotiksystem* (kurz: NAO) von *Aldebaran Robotics* ins Spiel: ein 58 Zentimeter kleiner, humanoider und zur Autonomie fähiger Roboter, welcher mit vielfältigsten Technologien ausgestattet ist [Rob15f]. Für Studenten und Forscher auf dem Gebiet humanoider Robotik ist der NAO als Fertiglösung gut geeignet, um Erfahrung zu sammeln und neueste Entwicklungen zu erproben. Solch ein NAO-Roboter, auf dessen Hardware und Möglichkeiten im Rahmen dieser Arbeit noch näher eingegangen werden soll, wird häufig in Szenarien mit schwer vorhersagbaren und kompliziert strukturierten Prozeduren konfrontiert. Für die erfolgreiche Bewältigung diverser Aufgaben müssen in Bereichen der mechanischen Bewegungen des Roboters, über die künstliche Intelligenz bis hin zu komplexer Bildverarbeitung viele Probleme gelöst werden.

Eine Plattform für die Forschung an Problemlösungen bietet der sogenannte *RoboCup*, ein jährlich stattfindender Wettbewerb, bei dem sich die Teilnehmer in verschiedenen Disziplinen den Herausforderungen diverser Themen der Robotik stellen. Diese umfassen beispielsweise logistische und produktionstechnische Abläufe, an Rettungseinsätze angelehnte Szenarien sowie das Fußballspiel mit autonomen Robotern. Letzteres ist das Einsatzgebiet, in welchem das NAO-Robotiksystem verwendet wird (siehe Abbildung 1.1a).

Das studentische Team *HULKs* der Technischen Universität Hamburg-Harburg nimmt ebenfalls an den RoboCup-Wettbewerben in der sogenannten *Standard Platform League* (kurz: SPL) teil und erarbeitet hierfür neue Software-Lösungen, welche die fußballerischen Fähigkeiten der NAOs verbessern sollen [HUL15]. Die vorliegende Bachelorarbeit ist Teil dieser Anstrengungen.



(a)



(b)

Abbildung 1.1: Ein NAO-Roboter (a) beim Fußballspiel und (b) in Pose [Rob15f].

1.1 Motivation

Um erfolgreich beim Roboterfußball abschneiden zu können, muss ein am RoboCup teilnehmendes Team in einem weitgefächerten Problemgebiet Lösungsansätze entwickeln und austesten. Die zu bewältigenden Aufgaben lassen sich, wie bereits erwähnt, grob in die Bereiche *Bewegung*, *Wahrnehmung* und *künstliche Intelligenz* einteilen. Dabei können zwischen diesen Bereichen durchaus Abhängigkeiten bestehen. Das in dieser Arbeit behandelte Thema gehört zu dem Aspekt der Wahrnehmung mithilfe von Bildverarbeitung auf Grundlage von Kamerabildern. Der NAO besitzt zwar zahlreiche Druck-, Lage-, Kapazitiv-, Infrarot- und Ultraschallsensoren sowie mehrere Mikrofone, jedoch stellen die Kameras des NAO die wohl wichtigste Quelle für Informationen über seine Umwelt dar. Das 6 Meter breite und 9 Meter lange Miniatur-Fußballfeld bietet keine externen Signalquellen für die Roboter an, somit sind die NAOs - auf ähnliche Art und Weise wie Menschen - größtenteils auf die Erfassung der Umgebungsmerkmale über die Licht- und Farbinformationen angewiesen [Com15].

Jedes Jahr werden einige Teile der Spielumgebung oder auch Regeln der Standard Platform League geändert, um für die Teilnehmer ein stetiges Maß an Herausforderung zu bieten und die Forschung voranzutreiben. Allen voran im Bereich der Bildinformationen werden kontinuierlich Aspekte, wie zB. die Team-Markierungen an den Körpern der NAOs oder auch die Farbcodierung der verschiedenen Torpfosten, verändert. Dies zwingt die Programmierer der Teams zum Umdenken und zum Verwenden neuer Kriterien zum Klassifizieren der wahrgenommenen Umgebungscharakteristika.

Im Jahr 2015 wurde vom technischen Komitee der SPL beschlossen, dass der für das Fußballspiel verwendete Spielball nicht mehr eine strahlend und einheitliche rote Farbe haben sollte, sondern dass stattdessen ab 2016 ein realistischer Spielball in Miniaturformat verwendet wird. Somit nähert sich dieser Teil der Spielumgebung dem Fußballspiel der Menschen an. Diese Regeländerung dient als Motivation zur Erprobung eines neuen Ballerkennungsalgorithmus, bei welchem nicht mehr die Farbe des Balls, sondern seine Form die hauptsächliche Rolle spielt.

1.2 Zielsetzung

Für die Anfertigung dieser Bachelorarbeit werden zunächst einige Ziele festgesetzt und nachfolgend erläutert. Es soll ein Algorithmus auf dem NAO implementiert werden, welcher als Hauptkriterium zur Findung eines Balls dessen Form verwendet. Dies soll zu einer Ballerkennung führen, welche unabhängig vom speziell verwendeten Ball funktioniert. Hierbei soll das Programm annähernd in Echtzeit ablaufen, das heißt mindestens zehn Bilder pro Sekunde und bestenfalls mehr verarbeiten können. Nach erfolgter Implementierung soll der Algorithmus auf seine Leistungsfähigkeit im Kontext des NAO-Fußballspiels hin überprüft werden.

Zum Erreichen dieser Ziele fiel die Wahl auf den *pixel2pixel*-Algorithmus, welcher 2005 von Scaramuzza et al. speziell für die Detektion von Bällen mittels mobiler Roboterplattformen vorgestellt wurde [SPV05, S. 1573-1578].

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich hauptsächlich in vier Teile. Zunächst soll der theoretische Hintergrund der technischen Gegebenheiten des Roboters und die Umstände der Spielumgebung beleuchtet werden. Hierzu gehört die Beschreibung des NAO, des Roboterfußballs und der verwendeten Bilddaten. Anschließend wird das zu lösende Problem erläutert, die bisherige Standardmethode präsentiert sowie der neue Lösungsansatz erklärt. Einige Details der Implementierung werden noch erklärt, bevor dann die Evaluation der Ballerkennung und die Diskussion der Ergebnisse erfolgt.
Zuletzt wird ein Fazit gezogen und ein Ausblick für weitergehende Entwicklungen gegeben.

Kapitel 2

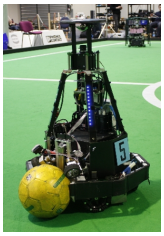
Theoretischer Hintergrund

Verglichen mit üblichen Standards in der komplexen Bildverarbeitung, ist der NAO-Roboter ein System, welches nur über geringe Rechenleistung verfügt. Aus diesem Grund kommen in der Steuersoftware häufig Algorithmen zum Einsatz, welche speziell für den gegebenen Anwendungsfall entwickelt und optimiert sind. Der in dieser Arbeit verwendete Algorithmus ist zielgerichtet auf die Detektion von Bällen in Echtzeit hin entworfen, implementiert und angepasst worden. Aus diesem Grund sollen im Folgenden einige theoretische Grundlagen über die vorherrschenden Bedingungen sowie über die Datengrundlage erläutert werden, um ein Verständnis für das Problem und den hierfür gewählten Lösungsansatz vermitteln zu können.

2.1 RoboCup-Wettbewerb

Der Name *RoboCup* steht kurz für *Robot World Cup Initiative*, eine jährliche Veranstaltung, welche 1995 von Forschern auf dem Gebiet der Robotik ins Leben gerufen wurde und welche 1997 zum ersten Mal stattgefunden hat. Der RoboCup wurde zu dem Zwecke gegründet, die Forschung speziell im Bereich der künstlichen Intelligenz und intelligenter Roboter im Allgemeinen voranzutreiben [Kit+95]. Ursprünglich war der RoboCup nur für Roboterfußball vorgesehen, daher erfährt dieser Teil der Veranstaltung auch immer noch den größten Zulauf durch Teilnehmer. Die Fußball-Wettkämpfe sind nach Größe bzw. Roboter-Kategorie eingeteilt. Hier gibt es die mit der *Small Size*- und *Middle Size*-Liga zwei Ligen für fahrende Roboter, sowie die *Humanoid*-Liga für selbstgebaute humanoide Roboter (siehe Abbildung 2.1). Die größte der Fußball-Ligen ist die *Standard Platform League* (SPL), in welcher die NAO-Roboter zum Einsatz kommen.Weitere bedeutende Wettbewerbe neben dem Fußball sind die *Robot Rescue League*, bei welcher ferngesteuerte Roboter Rettungseinsätze durchführen, die *RoboCup@Home*-Liga, welche den Robotern das Navigieren und Arbeiten im Haushalt abverlangt, sowie die *Logistics*-Liga als Herausforderung für industrielle Ro-


(a)


(b)


(c)

Abbildung 2.1: Abgebildet sind Roboter und Spielbälle der (a) Small-Size-, (b) Middle-Size- und (c) Humanoid-Kid-Size-Ligen [CS15][Rob15a][Ore15].

botik [Bon15]. Mit dem *RoboCup-Junior* wird auch jungen Schülern die Möglichkeit zum Präsentieren eigens erdachter Roboter-Vorführungen geboten.
Neben den Wettkämpfen findet auch stets ein Symposium als Forum für den Wissensaustausch zwischen den Teilnehmern statt.

2.1.1 *Standard Platform League*

In der Standard Platform League treten zwei Teams mit je fünf NAOs gegeneinander an (siehe Abbildung 2.2). Für die Verwendung der NAOs entschied sich das organisatorische Komitee bereits im Jahre 2008. Bis zu dem Jahr kamen noch die vierbeinigen Roboterhunde *Aibo* von Sony zum Einsatz, dessen Produktion jedoch 2006 eingestellt wurde [Hei15]. Es ist denkbar, dass in naher Zukunft auch andere Robotiksysteme anstelle des NAOs in der SPL verwendet werden, da fortlaufende professionelle Wartungsmöglichkeiten für kommerzielle Robotersysteme notwendigerweise von den wirtschaftlichen Interessen der involvierten Unternehmen abhängen.

Ein SPL-Fußballspiel dauert zwanzig Minuten und besteht aus zwei Halbzeiten zu je 10 Minuten, während der die Roboter autonom das Spiel bestreiten müssen. Das heißt, es darf keine Einwirkung von außen geben und alle Berechnungen müssen direkt auf den spielenden Systemen stattfinden. Die einzige externe Steuerinstanz ist der sogennante *Game-Controller*, ein Schiedsrichter, welcher mithilfe eines Computers über ein W-LAN-Netzwerk Zeitstrafen verteilen kann oder Spielstatus-Kommandos wie zB. „Anstoß“, „Tor“ oder „Halbzeit“ an die Roboter sendet. Untereinander dürfen die Roboter innerhalb eines definierten Protokollrahmens pro Sekunde und Team fünf *SPL-Messages* genannte Datenpakete mit allerlei Informationen austauschen. Die NAOs selber dürfen nicht modifiziert werden, da es sich bei der SPL nur um ein Wettmessen der besten Softwarelösungen handeln soll.

Die aktuellen SPL-Regeln (2015) schreiben eine Farbcodierung vor, nach welcher das Feld grün, die Linien und das Tor weiß, sowie der Ball rot sein müssen. Wie bereits zuvor erwähnt, werden die Regeln jährlich dahingehend angepasst, dass die Her-

(a) (b)

Abbildung 2.2: Die NAO-Roboter (a) vor Beginn eines Spiels und (b) währenddessen.

ausforderungen für die Teams wachsen. So waren die Tore bis zum Jahr 2014 noch farbcodiert, zunächst in blau bzw. gelb für die jeweiligen Feldseiten, anschließend nur noch gelb. Inzwischen sind die Torpfosten sogar komplett weiß. Außerdem waren spezielle Team-Trikotfarben vorgeschrieben. Und wie in Sektion 1.1 zuvor erwähnt, wird der aktuell Spielball der SPL, ein roter Hockeyball (siehe Abbildung 1.1a) durch einen realistischeren Spielball ersetzt. Es ist zu erkennen, dass die Regeländerungen Teil einer Bestrebung sind, die Roboter in ihrer Autonomie unabhängiger von farbcodierten Umgebungen werden zu lassen [Com15].

2.2 NAO-Robotiksystem

Der NAO wurde im Jahre 2006 in seiner ersten Version veröffentlicht. Bei der Entwicklung des Roboters lag das Hauptaugenmerk auf dem späteren Anwendungsgebiet der Mensch-Roboter-Interaktion, daher wurde der Roboter mit einer Vielzahl von Sensoren sowie mehreren Ausgabemöglichkeiten ausgestattet [Rob15f]. Somit kann der NAO im Auslieferungszustand sehen und hören, aber auch mittels Bewegungen und Sprachausgabe auf seine Umwelt reagieren. Aufgrund des beabsichtigten Umgangs mit Menschen wurde dem NAO zudem ein recht rundes, freundliches und fast spielzeughaft wirkendes Aussehen gegeben (siehe Abbildung 1.1b). Im Gegensatz zu vielen heutzutage existierenden Robotern wirkt der NAO hierdurch weder zu künstlich, noch zu technisch und keineswegs gefährlich, sondern lädt stattdessen zur Interaktion ein.

Besonderen Anklang fand die Roboterplattform im Bildungsbereich: viele Bildungseinrichtungen kauften den NAO, um Schülern und Studenten den Umgang mit Software und Hardware näherzubringen. Hierdurch ist der Roboter inzwischen in über 70 Ländern in Institutionen wie Grundschulen, aber auch Universitäten vertreten. Aktuell arbeitet Aldebaran Robotics daran, den NAO für Endbenutzer und somit einer breiten Öffentlichkeit anzubieten [Rob15f]. Gründe für die große Popularität des Roboters dürf-

ten neben seinem Erscheinungsbild auch die hohe Verarbeitungsqualität sowie die große Auswahl an technischen Features sein. Eine der größten Hürden der humanoiden Robotik - nämlich die Konstruktion eines Roboters, der zu einem aufrechten Gang fähig ist - wird den Käufern abgenommen. Und während unerfahrene Anwender gerne Gebrauch von der mitgelieferten Herstellersoftware machen, welche das einfache Erstellen von Programmen erlaubt, wissen fortgeschrittene Entwickler das Potenzial der Plattform zu schätzen. Somit stellt das NAO-Robotiksystem für Universitäten eine besonders attraktive Investitionsmöglichkeit dar, um Studenten in den Bereich der humanoiden Robotik einzuführen. State-of-the-Art Softwarelösungen aus dem Bereich der Bewegungssteuerung, künstlichen Intelligenz und Bildverarbeitung können mithilfe des NAO erprobt oder auch neu entwickelt werden. Universitätsteams, welche für die Teilnahme am RoboCup qualifiziert sind, können die Roboter zu einem Stückpreis von 3.200 € erwerben [UB15].

Im Folgenden wird näher auf die Hardware und speziell die Kamera des NAOs eingegangen, sowie eine Einführung in das für die Programmierung in dieser Arbeit verwendete Software-Framework gegeben.

2.2.1 Hardware-Überblick

Im Laufe der letzten Jahre wurden mehrere Versionen des NAO-Roboters entwickelt. Die Unterschiede liegen meist in kleinen Verbesserungen wie zB. Kameras mit höherer Auflösung oder stabileren Zahnrädern. Für diese Arbeit wurden NAO-Roboter der aktuellsten Version *V5* verwendet. Alle folgenden Angaben beziehen sich auf diese Version des Robotiksystems.

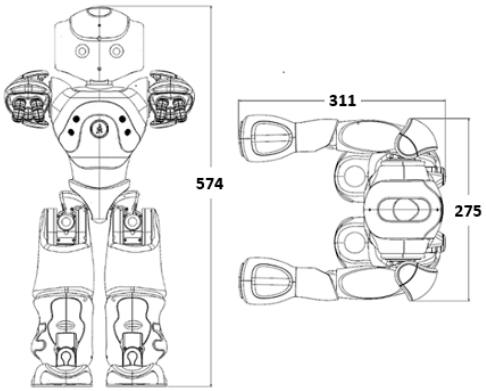


Abbildung 2.3: Die Maße des NAO-Roboters [Rob15c].

Ein NAO-Roboter ist ca. 57 cm hoch und wiegt 5,4 kg (siehe Abbildung 2.3). Er besitzt eine eigenständige Stromversorgung mittels austauschbarem Lithium-Ionen-Akku, welcher bei normaler Verwendung ca. eine Stunde lang hält. Auf dem Mainboard, das

sich im austauschbaren Kopf des Roboters befindet - sind eine Intel ATOM Z530 CPU mit 1,6 GHz, 1 GB RAM sowie 2 GB Flash-Speicher integriert. Der Speicher ist mit einer SD-Karte auf bis zu 10 GB erweiterbar. Für Netzwerkverbindungen wird Ethernet mit integriertem Port sowie IEEE 802.11a/b/g/n WIFI unterstützt. Des Weiteren sind im Kopf zwei Kameras, zwei Lautsprecher, zwei Infrarot-Sensoren sowie vier rundherum angeordnete Mikrofone und zahlreiche LEDs integriert. Im Torso des Roboters befinden sich je zwei Ultraschall-Sender und -Empfänger und zudem ein Inertialsensor, bestehend aus Gyroskop und Beschleunigungssensor.

Der NAO verfügt insgesamt über 25 Freiheitsgrade, welche es dem Roboter ermöglichen, seine Arme und Beine, Hände und Finger, seine Füße, seinen Körper und seinen Kopf in vielfältige Positionen zu bringen [Rob15e].

2.2.2 Die Kameras

In der Vorderseite des Kopfes eingelassen befinden sich zwei Farbkameras zur Warnehmung der vor dem Roboter befindlichen Umgebungsobjekte. Sie sind übereinander angeordnet und bieten zwei Sichtfelder, welche sich nur wenig überschneiden und dadurch keine Stereo-Sicht ermöglichen. Durch diese Anordnung ist es dem Roboter aber möglich, ohne Kopfbewegung einen vertikalen Sichtbereich von über dem Horizont bis zu seinen Füßen wahrzunehmen. Der horizontale Sichtkegel jedoch ist mit 60,97° hierfür ein wenig eingeschränkt (siehe Abbildung 2.4).

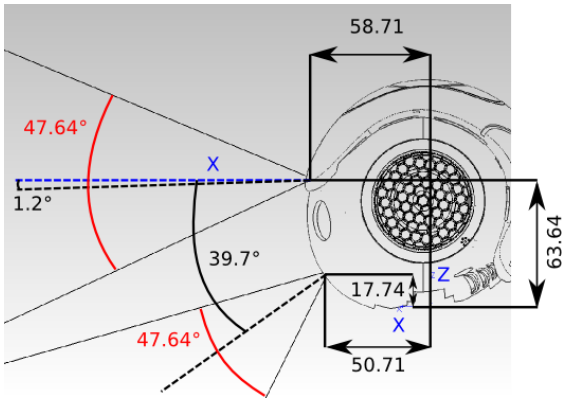


Abbildung 2.4: Anordnung der Kameras im Kopfteil des Roboters [Rob15d].

Die Kameras liefern in der feinsten Einstellung 29 Bilder pro Sekunde bei einer Auflösung von 1280 x 960 Pixeln. Der integrierte Bildsensor im *MT9M114*-SoC (System-on-a-Chip) bietet aber auch die Möglichkeit, andere Ausgabe-Auflösungen zu wählen, welche in Tabelle 2.1 aufgelistet sind.
Im Rahmen dieser Arbeit wurde mit einer Auflösung von 640 Pixeln in der Breite und 480 Pixeln in der Höhe gearbeitet. Dies liegt darin begründet, dass der Balldetekti-

Auflösung [Pixel]	Bilder pro Sekunde
160 x 120	30
320 x 240	30
640 x 480	30
1280 x 960	29

Tabelle 2.1: Vom NAO-Kamerasensor unterstützte Auflösungen. Pixelangaben sind in Breite x Höhe [Rob15e].

onsalgorithmus in einem Programmmodul als Teil des bereits bestehenden HULKs-Framework implementiert wird. Den Bildverarbeitungs-Modulen stehen hierbei pro Sekunde 30 Bilder der Auflösung 640 x 480 zur Verfügung. Diese Auflösung ist für die Bewältigung der meisten Erkennungsaufgaben im wenige Meter umfassenden Nahbereichs des Roboters ausreichend und erzeugt im Vergleich zur höchsten Auflösung 75% weniger Daten, die verarbeitet werden müssen.

Der Kameratreiber erlaubt jedoch nicht nur das Wählen einer Auflösung, sondern auch das Setzen einer Vielzahl von Parametern, welche das Erscheinungsbild der aufgenommenen Bilder deutlich beeinflussen. So können Einstellungen wie die Belichtungszeit, die digitale Verstärkung, Sättigung, Schärfe und Weißabgleich geändert werden. Für die Aufnahme der Bilder in dieser Arbeit wurden die Kameraparameter so konfiguriert, dass die Bilder möglichst natürlich aussehen, ohne besonders ausgeprägte Bewegungsunschärfe aufzuweisen.

Betrachtet werden im Folgenden nur solche Bildern, welche mit der oberen Kamera des NAOs aufgenommen wurden. Begründen lässt sich diese Vorgehensweise damit, dass die Bilder aus der oberen Kamera den größten Teil des Sichtfeldes abdecken, während die untere Kamera meist nur einen kleinen Bereich vor den Füßen des NAOs beobachtet. Außerdem sind die Erkennungsprobleme auf Bildern der oberen Kamera schwieriger zu lösen, da sie oftmals viele Hintergrundobjekte mit aufnimmt. Daher ist bei zufriedenstellender Problemlösung unter Nutzung der oberen Kamera davon auszugehen, dass ein entwickelter Algorithmus sich ohne weitere Probleme für die untere Kamera anpassen ließe.

Bei dem Bildsensor handelt es sich um einen typischen RGB-*Active Pixel Sensor*. Dies ist ein flaches Bauteil, dessen Oberfläche mit einer Matrix aus Halbleiterdetektoren besetzt wurde. Das einfallende Licht bewirkt auf jedem getroffenen Sensorelement das Aufkommen einer Spannung, welche Proportional zu der Lichtintensität ist. Auf dem Sensor sind verschiedene Elemente für verschiedene Lichtfarben vorhanden. Bei diesem RGB-Sensor sind es die Farben Rot, Grün und Blau. Angeordnet sind die Elemente im Muster der sogenannten *Bayer-Matrix*, welche den menschlichen Wahrnehmungsgegebenheiten angepasst ist (siehe Abbildung 2.5 [Bay75]. Hierbei ist die Empfindlichkeit

für grüne Bestandteile des Lichts höher als für die roten und blauen.

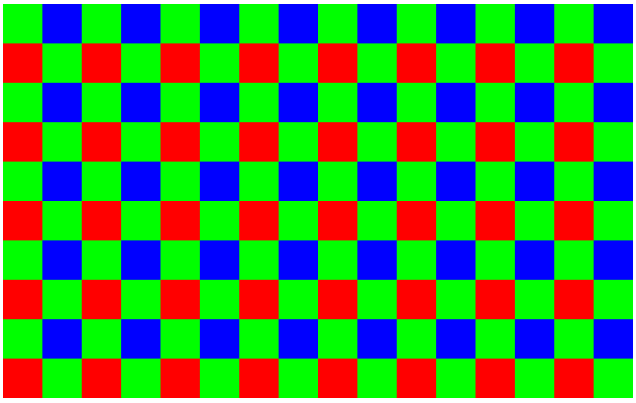


Abbildung 2.5: Anordnung der lichtempfindlichen Farb-Detektorelemente auf dem Bildsensor in Form einer Bayer-Matrix.

Die Datenausgabe des Kamerachips ist jedoch nicht im typischen 3 * 8 Bit RGB-Datenformat (Drei Kanäle, acht Bit pro Kanal für je 256 Farbabstufungen). Stattdessen gibt der Hersteller als Format *YUV422* an, in dem zwar auch 3 * 8 Bit Daten pro Pixel vorkommen, dessen Farbmodell jedoch ein Anderes ist. Verwendet wird YUV422 aufgrund einiger Vorteile, welche dieser Farbraum bei Einsatz auf dem NAO bietet. Das Farbmodell sowie dessen Vorteile werden nachfolgend erläutert.

2.2.2.1 YCbCr-Farbmodell

Zunächst einmal ist darauf hinzuweisen, dass die Abkürzung *YUV* - welche für einen Farbraum in Analogfernsehsystemen steht - häufig fälschlicherweise auch für den digitalen Farbraum *YCbCr* verwendet wird. Trotz des als YUV422 angegebenen Formats wird daher im Folgenden nur noch die korrekte Bezeichnung *YCbCr* für die digitale Darstellung von Farben verwendet [Poy03b].

Während sich im RGB-Farbraum alle enthaltenen Farben additiv aus den drei Farbkomponenten Rot, Grün und Blau ergeben, ist der YCbCr-Farbraum anders aufgeteilt. Farbdaten werden hier durch ihre Helligkeit (Luma) und Farbigkeit (Chroma) definiert. Die Bilddaten im YCbCr-Raum werden aus den RGB-Daten heraus berechnet. Nach der Umrechnung eines Bildes von RGB zu YCbCr enthält dessen Y-Kanal nur noch die Information über die Helligkeit der Pixel, der Cb- und Cr-Kanal hingegen über die Abweichung von der Neutralfarbe (Grau) in Richtung Blau/Gelb bzw. Rot/Türkis. Ermittelt werden die korrekten Werte mithilfe der Formeln 2.1 - 2.3 [Bey12].

$$Y := 0,299 * R + 0,587 * G + 0,114 * B \tag{2.1}$$

$$Cb := 128 - 0,168736 * R - 0,331264 * G + 0,5 * B \tag{2.2}$$

$$Cr := 128 + 0,5 * R - 0,418688 * G - 0,081312 * B \tag{2.3}$$

Der YCbCr-Farbraum lässt sich für gegebene Y-Werte als eine durch Cb- und Cr-Werte aufgespannte Ebene visualisieren (siehe Abbildung 2.6).

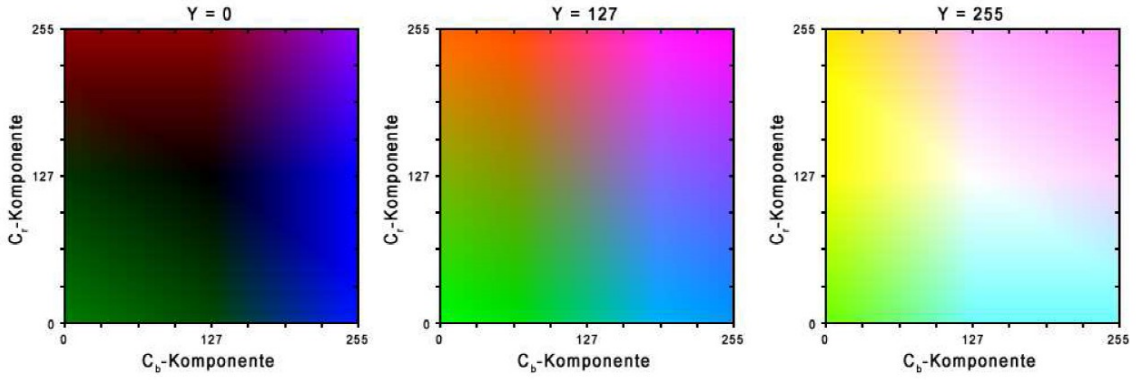


Abbildung 2.6: Dargestellt ist die CbCr-Ebene bei jeweils verschiedenen Werten für Y [Rei11, S. 18-19].

Hierbei ist nun leicht zu erkennen, dass sich die Position der Farben innerhalb der Ebenen auch bei verschiedenen Helligkeiten nicht stark verändern. Außerdem liegen die Farben Grün, Rot, Türkis und Magenta in gegenüberliegenden Ecken der Ebenen. Da dies Farben sind, welche beim RoboCup-Roboterfußball zur Codierung einer Spielelemente dienen, ist die gute Trennbarkeit der Farben für die Klassifizierung von Objekten bei variablen Lichtbedingungen hilfreich.

Ein weiterer Aspekt ist die Möglichkeit der Unterabtastung des Farbraums bei Beibehaltung aller Helligkeitsinformationen. Bei der als 4:2:2 oder nur 422 genannten Arbeitsweise wird nur für jeden zweiten Bildpunkt in horizontaler Richtung die Farbinformation gespeichert, wodurch die Datenmenge verringert, das entstehende Bild aber trotzdem noch dem menschlichen Seherlebnis nachempfunden bleibt. Dieses Verfahren wird als Chroma-Sub-Sampling bezeichnet [Poy03a].

Abschließend sei zu der Wahl des Farbmodells für die Bildverarbeitung auf dem NAO-Robotiksystem noch gesagt, dass die Verwendung von einem anderen als dem YCbCr-Modell eine manuelle Umrechnung für jeden Bildpixel per Software erfordern würde, während das Kamerasystem bereits integriert die Umwandlung der RGB-Rohdaten in YCbCr vornimmt. Somit bedeutet die Wahl eines abweichenden Farbraums stets einen zusätzlichen Rechenaufwand.

2.3 Software-Überblick

Wird ein NAO im Neuzustand beim Hersteller erworben, so wird dieser standardmäßig mit einer Betriebssoftware ausgeliefert, welche auch unerfahrenen Benutzern das Erstellen von Programmen erlaubt. Da die verschiedenen benötigten Algorithmen aber mit hoher Effizienz implementiert werden müssen, ist der Einsatz einer Programmiersprache wie zB. C++ notwendig. Das Linux-basierte Hersteller-Betriebssystem *NAOqi* erlaubt es, eigene Programm-Module in Sprachen wie Java, Python und C++ zu schreiben und auszuführen. Zusätzlich wird ein SDK (Software Development Kit) zur Verfügung gestellt, welches Zugriff auf vom Hersteller bereitgestellte Funktionalitäten ermöglicht. Das Team HULKs entwickelt den kompletten Code für den NAO in C++. Ein knapper Überblick darüber, welche Funktionalität der Hersteller auf dem NAO bietet und wie selbstgeschriebene Software für den NAO entwickelt wird, soll nachfolgend gegeben werden.

2.3.1 NAOqi-SDK

Das C++-SDK des Herstellers ist äußerst umfangreich und bietet Funktionen von Bilderfassung über Audioausgabe bis hin zu Gefühlserkennung in Gesichtern von Menschen [Rob15b]. Ein Großteil der gebotenen Funktionalität wird aber im Kontext des Roboterfußballs entweder nicht benötigt, oder aber ist - wie zB. das bereits vorhandene Laufmuster - zu langsam. Trotzdem bietet das SDK immer noch nützliche Aspekte, wie zB. die einfache Realisierung einer Sprachausgabe mit dem Roboter. Dies kann unter anderem nützlich dazu sein, Statusinformationen auszugeben. Obwohl es in gewissem Rahmen möglich gewesen wäre, wurden für diese Bachelorarbeit jedoch keine Teile des SDK verwendet. Benötigte Grundfunktionen wie Bilderfassung und Debug-Möglichkeiten sind schon im HULKs-Framework enthalten.

2.3.2 Das *HULKs*-Framework

Die Arbeit am HULKs-Framework begann im April 2013 mit dem Ziel, eine komplett selbständig entwickelte Codebase zu erschaffen [PBK14]. Der Aufbau des Frameworks ist schematisch in Abbildung 2.7 dargestellt. Es ist in drei große Teile gegliedert: *Brain*, *Vision* und *Motion*. Eingebettet sind diese Module in das *tuhhSDK*, welches modulübergreifende Funktionalitäten bietet. Die Namen der Module erklären bereits, wie die Unterteilung der Funktionalität in drei Untermodule vorgenommen wurde. *Brain* ist hierbei für die komplette Spiellogik und alle dies betreffenden Entscheidungen unter Nutzung von Zustandsautomaten, sowie für die Kommunikation zwischen den Robotern zuständig. *Motion* führt die Bewegungsbefehle aus und stellt die zahlreichen dafür notwendigen Berechnungen an. *Vision* liefert lediglich Ergebnisse aus der Verarbeitung

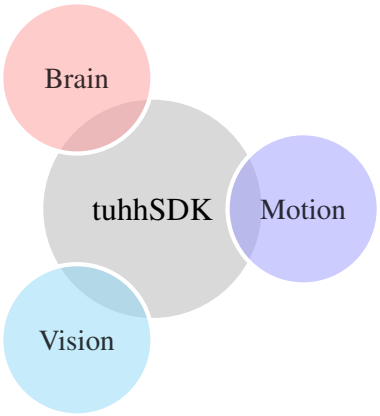


Abbildung 2.7: Der Aufbau des HULKs-Frameworks.

der Kamerabilder. Diese Unterteilung erlaubt auch das gezielte Einsetzen von studentischen Programmierern mit bestimmter Expertise und Wissen in den entsprechenden Bereichen. Des Weiteren können einzelne Module hierdurch bei gleichbleibender Schnittstelle auch ausgetauscht oder komplett überarbeitet werden, ohne die anderen Module zu beeinflussen.

Der Code für diese Arbeit wurde als Teilmodul des Vision-Moduls entwickelt. Jedes Teilmodul von Vision hat Zugriff auf die Kamerabilder und die Ergebnisse anderer Module und besitzt darüber hinaus die Option zum Senden von Debug-Daten über eine Netzwerkverbindung. Jedes Vision-Teilmodul wird pro Programmzyklus einmal mit bereitgestellten aktuellen Bilddaten aufgerufen.

Der Arbeitsablauf mit dem Framework und den Robotern lässt sich grob in einigen wenigen Schritte zusammenfassen: Zunächst wird der Programmcode lokal in einer C++-IDE geschrieben, danach wird durch das Kompilieren des Codes pro Hauptmo-

Abbildung 2.8: Das Debug-Webinterface für die NAO-Softwareentwicklung.

dul eine Programmbibliothek erzeugt, welche im Anschluss jeweils auf den Roboter hochgeladen wird. Per Fernzugriff wird das Betriebssystem des Roboters in der Linux-Umgebung gestartet. Während des Starts werden die eigens registrierten Module in den Speicher geladen und ausgeführt. Läuft das so entwickelte und ausgeführte Programm nun auf dem NAO, bietet sich die Möglichkeit, per Netzwerkverbindung und Konsole bzw. Web-Interface den Programmablauf mithilfe von Debug-Ausgaben zu beobachten (siehe Abbildung 2.8.

2.4 Bild-Datenformat

Bevor eine detaillierte Betrachtung der Bildverarbeitungsalgorithmen erfolgen kann, ist es zunächst von Nutzen, noch einige grundlegende Anmerkungen zu den Bilddaten zu machen. Die Algorithmen arbeiten direkt auf den YCbCr-Rohbildern und prozessieren pro Programmzyklus wahlweise die kompletten oder auch eine Untermenge der bereitgestellten Daten (Subsampling). Ein Bild besteht hierbei aus einem zweidimensionalen Punktfeld und ein Punkt auf diesem Gitter bzw. in dieser Matrix wird als Pixel bezeichnet. Für jeden Pixel kann die Position über Angabe von Koordinaten im bildeigenen Ortsraum beschrieben werden (siehe Abbildung 2.9).

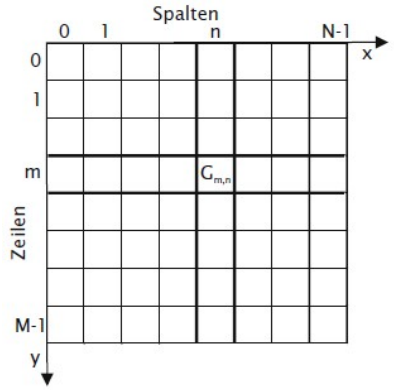


Abbildung 2.9: Anordnung der Pixel im Koordinatensystem der Bildmatrix [Jäh05].

Wie für Matrizen typisch, ließe sich eine Pixelposition per Zeile m und Spalte n angeben. Häufig ist jedoch in der digitalen Bildverarbeitung ein X-Y-Koordinatensystem in Gebrauch, bei welchem der Nullpunkt des Koordinatensystem an Matrixstelle $G_{0,0}$ liegt, und die Y-Achse, analog zur Matrixnotation, positiv von oben nach unten verläuft (anstatt wie gebräuchlich von unten nach oben) [Jäh05, S. 31-32]. Jeder Pixel enthält des Weiteren wie zuvor schon beschrieben die Intensitätsinformationen der drei Farbkanäle, angegeben mit jeweils 8 Bit. Dies ermöglicht das Darstellen von ca. 16,7 Millionen Farben, was eine natürlich erscheinende Abbildung der Bildobjekte ermöglicht.

Bei Verwendung der Pixelinformationen in Algorithmen ist es oft erstrebenswert, keine Sprünge durch die Bildmatrix durchzuführen, sondern die Pixel in gerasterter Weise durchzugehen. Da die Bildinformationen nämlich sequenziell im Speicher liegen, verursachen häufige Sprünge im Bild Verluste in der Laufzeit eines Algorithmus. Somit ist bei Bearbeitungsschritten in der digitalen Bildverarbeitung immer auch ein Augenmerk darauf zu legen, in welcher Art und Weise die Bildpunkte ausgewertet werden.

Kapitel 3

Ballerkennung

Seit der Nutzung der NAOs im SPL-Roboterfußball des RoboCup bestand die Aufgabe der Ballerkennung darin, die Position des roten Balls auf dem Spielfeld zu ermitteln. Viele teilnehmende Teams haben diese Aufgabe letztlich sehr gut lösen können, da sich die Farbe des einheitlich roten Balls in der Regel sehr gut vom Grün des Spielfeldes, von den Robotern sowie größtenteils auch von der Umgebung absetzt. Da der rote Spielball allerdings durch einen realistischeren, verschiedenfarbigen Ball mit Muster ersetzt werden soll, kann die Farbe zukünftig wahrscheinlich nur noch schwierig als Kriterium zum Finden der Ballposition verwendet werden (Abbildung 3.1 verdeutlicht den großen Unterschied zwischen den Bällen).

Diese Aussage lässt sich bei Betrachtung einiger Abbildungen untermauern, in welchen Aufnahmen von verschiedenen Spielbällen - aufgeteilt in die Y-, Cb- und Cr-Kanäle - gezeigt werden (siehe Abbildung 3.2).
Es ist gut zu erkennen, dass sich der rote Spielball im Cr-Kanal deutlich vom Hintergrund abhebt. Auch bei Betrachtung mehrerer Bilder liegt der maximale Cr-Wert meist

(a) (b)

Abbildung 3.1: Bild (a) zeigt den aktuellen Standard-SPL-Ball und (b) zeigt einen möglichen zukünftigen Spielball.

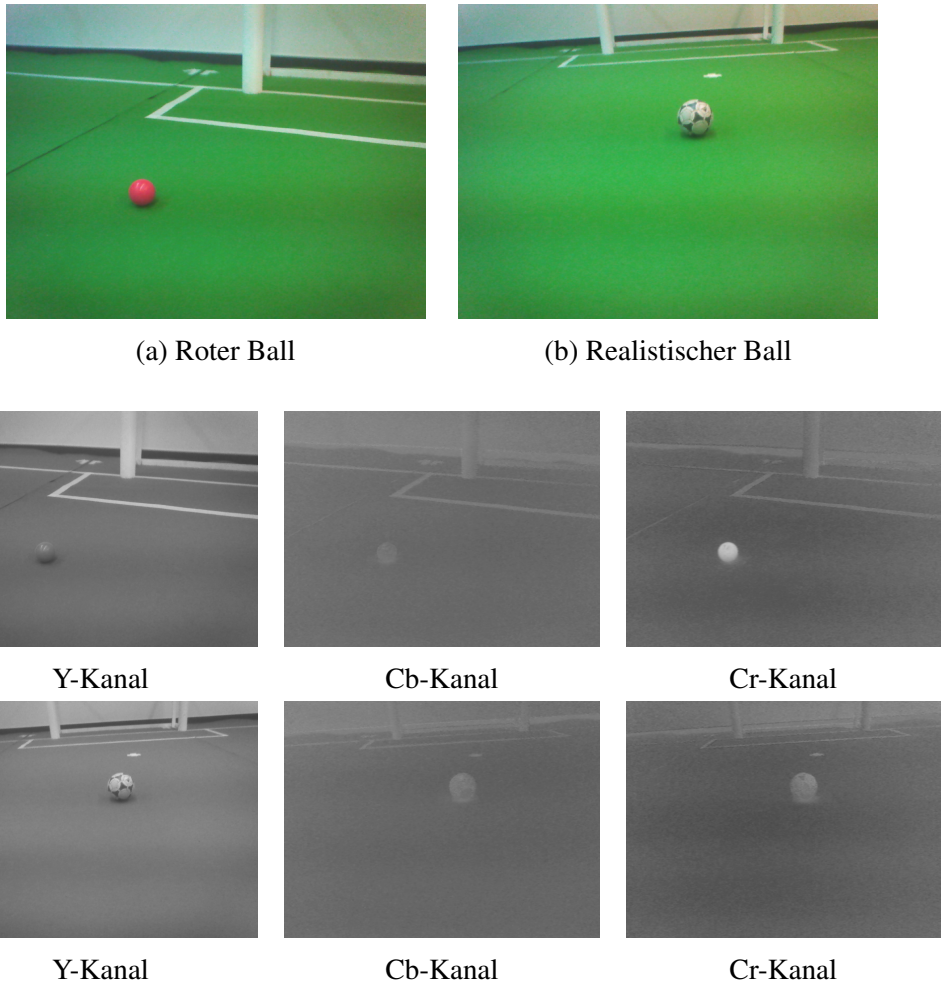


Abbildung 3.2: Die obere Bildreihe gehört zum roten Spielball (a) und die untere zum realistischen Ball (b). Die Bilder wurden mit der Kamera des NAO-Roboters aufgenommen.

an der Position des Balls vor. Aus diesem Grund kann der rote Ball gut primär über seine Farbe lokalisiert werden. Der realistische Ball hingegen hebt sich in keinem Kanal besonders vom Hintergrund ab, am Wenigsten sogar in den beiden Chroma-Kanälen. In diesem Fall bietet der Y-Kanal die größte Unterscheidungsmöglichkeit, weshalb dieser zum Finden des Balls verwendet werden soll. Nachfolgend soll einerseits anhand eines Beispiels eine farbbasierte Ballerkennung erklärt werden und andererseits der Ansatz der formbasierten Erkennung eine nähergehende Beschreibung erfahren.

3.1 Farbbasierte Verfahrensweise

Zur Detektierung eines roten Spielballs existiert bereits eine Vielzahl von im Rahmen des RoboCup entwickelter Lösungen. Durch Austausch mit anderen Teilnehmer und Betrachtung einiger Team-Berichte ist es aber schnell ersichtlich, dass fast immer die Farbe des Balls als primäres Erkennungsmerkmal dient, während Sekundärfaktoren wie Form und Größe nur zur Überprüfung von gefundenen Positionen verwendet wird [Röf+13][Ash+14]. An dieser Stelle soll beispielhaft ein anschaulicher Balldetektionsalgorithmus des NAO-Team HTWK der Universität von Leipzig beschrieben werden, welcher zuverlässig gute Ergebnisse liefert. Die nachfolgenden Erklärungen dieser Sektion sind eine Zusammenfassung eines Kapitels der Arbeit von [Rei11, S. 81-92] und erfahren dort eine detailliertere Beschreibung.

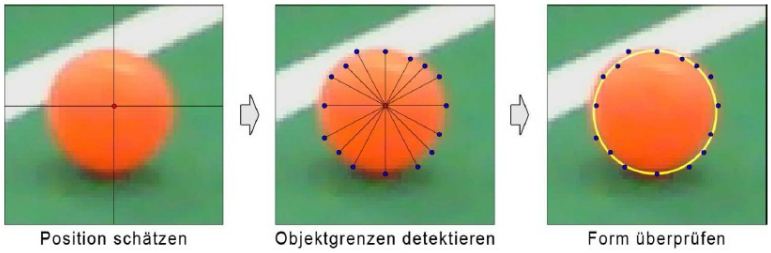


Abbildung 3.3: Ablauf der Detektion des roten Spielballs [Rei11, S. 81-92].

Zunächst wird das 640 x 480 Pixel große Eingangsbild mit einem starren 4x4-Pixel-Raster unterabgetastet. Dies verringert die Datenmenge erheblich und trägt somit zu einer schnellen Algorithmuslaufzeit bei, während trotzdem noch gewährleistet ist, dass immer mindestens einer der unterabgetasteten Bildpunkte auf dem Ball liegt. Zuvor für das Bild gewonnene Informationen über das Spielfeld werden dann genutzt, um anschließend nur noch den Bereich des Bildes zu untersuchen, welcher sich unterhalb des Spielfeldrandes befindet. In diesem Bereich wird nun ein Histogramm des Cr-Kanals angefertigt und die Position des maximalen Histogrammwerts als möglicher Kandidat für den Spielball geschätzt. Um nun zu überprüfen, ob der ermittelte Bildpunkt tatsächlich die Ballposition markiert, wird als nächstes die Form der gefundenen roten Bildregion überprüft. Hierfür wird von der geschätzten Position aus strahlenförmig eine Kantendetektion ausgeführt, welche basierend auf Schwellwerten die Stelle des ersten großen Helligkeits- bzw. Farbunterschieds als Kante speichert (siehe Abbildung 3.3). Als Letztes wird überprüft, ob die gefundenen Kantenpunkte kreisförmig angeordnet sind, um die detektierte rote Region als Ball akzeptieren oder ausschließen zu können. Ein Kreismodell zusammen mit der Bestimmung der Summe der quadratischen Abstände aller Kantenpunkte zu diesem Kreis wird genutzt, um eine Plausibilität der Daten zu ermitteln bzw. um ein Maß dafür zu erhalten, wie "kreisförmig"die gefundene Region

ist. Liegt dieses Maß unter einem vorher festgelegten Schwellenwert, wird die gefundene Region als Ballposition angenommen, ansonsten wird sie verworfen und kein Ball wurde im Bild erkannt. Der Algorithmus arbeitet außerdem Bild-für-Bild. Das heißt, für alle zur Verfügung gestellten Bilddaten startet der Verarbeitungsprozess von Neuem, ohne vorheriges Wissen vergangener Bilder über zuvor detektierte Ballpositionen zu nutzen. Das beschriebene Verfahren enthält zur Steigerung der Erkennungsrate und Verminderung der falsch erkannten Spielbälle weitere Schritte wie zB. das Aussortieren von Ausreißern unter den Kantenpunkten oder die Überprüfung der Ballgröße. Jedoch reicht die bisherige Erklärung dieser Methode bereits aus, um Unterschiede zum in dieser Arbeit verwendeten Ansatz erkennen zu können. Das eben beschriebene Verfahren erreicht für den roten Ball eine sehr gute Erkennungsrate von 99,2 %, bei einer Falsch-Erkennungsrate von lediglich 0,17 % und einer Laufzeit von durchschnittlich 3,2 ms [Rei11].

Es ist also zu verstehen, dass die Farbe des roten Balls das wichtigste Kriterium zum Finden seiner Position ist, während die Form nur innerhalb eines von mehreren Schwellenwert-Toleranzen bestimmten Rahmen überprüft wird. Beides zusammen führt zu einer sehr guten Erkennungsrate. Hierbei ist jedoch zu beachten, dass der Algorithmus hoch spezialisiert für den roten SPL-Ball entworfen wurde und somit nicht vielseitig einsetzbar ist.

3.2 Formbasierte Vorgehensweise

Nun soll also nicht mehr nur der rote Spielball detektiert werden, sondern ein realistischer Ball. Die meisten Fußbälle sind weiß und treten, wie zuvor beschrieben, nicht besonders in einem Farbkanal hervor (siehe Abbildung 3.2, untere Reihe). Es ist also naheliegend, ein anderes wichtiges Kriterium, welches den Ball von allen anderen Elementen der Spielumgebung unterscheidet, zu verwenden: seine kreisrunde Form. Eine der am weitesten verbreiteten Methoden zur Erkennung von Kreisen auf Bildern ist die Circle-Hough-Transformation (CHT) [DH75]. Die CHT zielt darauf ab, Kreise mit einem gegebenen Radius R in einem Bild zu finden. Hierfür wird zunächst ein binäres Kantenbild erstellt. Anschließend wird für jeden Kantenpunkt ein Kreis mit dem Radius R in einen Ausgaberaum geschrieben. Wenn der Radius des zu findenden Kreises unbekannt ist, werden für jeden Kantenpunkt mehrere Kreise mit allen zu erwägenden Radien im Ausgaberaum erzeugt. Dort ergeben sich, nachdem alle Kantenpunkte verarbeitet wurden, genau an den Stellen Maxima, an welchen sich im Bild tatsächlich Kreise wiederfinden. Nachteile der Methode sind großer Speicherbedarf, viel benötigte Rechenkapazität und Instabilität gegenüber Bildrauschen. Daher ist die CHT eher ungeeignet für Echtzeitanwendungen auf mobilen Robotern [SPV05].

Scaramuzza et al. beschreiben in ihrem Paper „Ball Detection and Predictive Ball Following Based on a Stereoscopic Vision System“ unter anderem einen Bild-für-Bild-

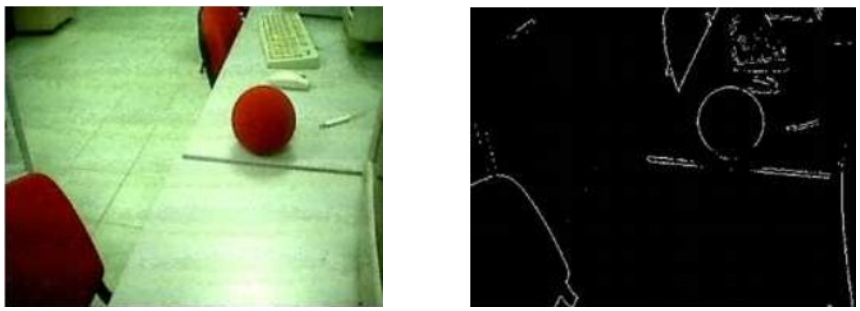
Balldetektionsalgorithmus, welcher für den Einsatz auf mobilen Roboterplattformen entwickelt wurde. Dort wird eine Vorgehensweise erklärt, bei welcher zunächst ein Ball in einem Bild detektiert und dessen Position anschließend über mehrere Bilder hinweg verfolgt wird. Für diese Bachelorarbeit sollte nur die *pixel2pixel* genannte Balldetektion auf ihre Nutzbarkeit auf der NAO-Robotikplattform hin untersucht werden. Dieser Algorithmus wurde nicht nur aus dem Grund gewählt, dass er speziell für Roboter vorgesehen ist, sondern auch weil er schnell implementiert und auf seine Leistungsfähigkeit hin untersucht werden kann.

3.3 *pixel2pixel*-Algorithmus

Der pixel2pixel-Algorithmus wird, wie auch die Circle-Hough-Transformation, auf ein binäres Kantenbild angewendet. Vorteile gegenüber der CHT sind aber, laut Scaramuzza et al., eine gesteigerte Effizienz, erhöhte Genauigkeit sowie der Ausschluss von störenden Kanten in einem Bild [SPV05].

3.3.1 Kantenbild

Ein Kantenbild entsteht als Ergebnis einer Kantendetektion, die auf ein Eingangsbild angewendet wurde. Es besteht aus einer Pixelmatrix in Größe des Originalbildes, in welcher Kantenpunkte durch eine 1 und andere Bildpunkte durch eine 0 repräsentiert werden. Es handelt sich also um ein Binärbild. Das heißt, dass in solch einem Bild alle Farbinformationen nicht mehr vorhanden sind und stattdessen nur noch Formen erkannt werden können. Für die hier präsentierte Methode muss gewährleistet sein, dass alle Kanten im Kantenbild nur einen Pixel breit sind. Um ein Kantenbild zu erzeu-



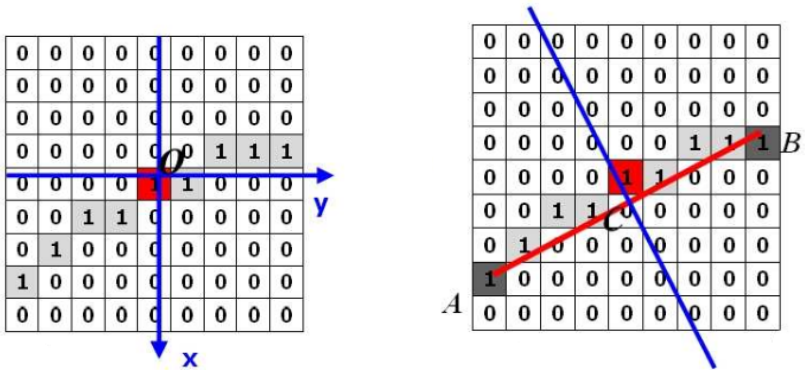
(a) (b)

Abbildung 3.4: Ein Kamerabild (a) als Originalaufnahme und (b) als binäres Kantenbild [SPV05].

gen, gibt es verschiedene Methoden, wie zB. der Canny-Kantendetektionsalgorithmus (kurz: Canny) [Can86] oder auch der Sobel-Operator [Jäh05, S. 365-366]. Beide erzeugen auf unterschiedliche Arten Kantenbilder und finden verbreitete Anwendung. Auf die Theorie der Kantendetektion soll hier jedoch nicht weiter eingegangen werden. Es gibt für Kantendetektionsalgorithmen bereits zahlreiche Implementierungen, auf welche zugegriffen werden kann. Für diese Arbeit wurde der Canny-Algorithmus gewählt, da dieser zuverlässige Ergebnisse liefert und die erzeugten Kanten nur einen Pixel breit sind. Zudem wird momentan eine effiziente Implementierung des Canny-Kantendetektionsalgorithmus in das HULKs-Framework integriert [Lot15].

3.3.2 Funktionsweise

Liegt nun ein Kantenbild wie in Abbildung 3.4b vor, kann der pixel2pixel-Algorithmus auf dieses angewendet werden. Zur Untersuchung einer Kanten wird ein Untersuchungsfenster der Größe $(2k+1)\cdot(2k+1)$ mittig auf einen Kantenpixel gelegt. Das für das Fenster gewählte Koordinatensystem kann Abbildung 3.5a entnommen werden.



(a) (b)

Abbildung 3.5: Untersuchungsfenster (a) und gemoetrische Berechnungen (b) [SPV05].

Jedes Untersuchungsfenster, welches potenzielle Teile eines Kreises als Kantenstruktur enthält, soll zur Bestimmung des Mittelpunktes dieses Kreiskandidaten beitragen. Da hierfür nur Bogenkanten ausgewertet werden sollen, muss zuerst festgestellt werden, ob in dem Untersuchungsfenster tatsächlich solch eine Bogenkante vorliegt. Es soll zunächst überprüft werden, ob die im Untersuchungsfenster enthaltenen Pixel eine verbundene Kante ergeben. Allgemeines Merkmal für eine fortlaufenden Kante in einem Canny-Kantenbild ist die Nachbarschaft der beteiligten Kantenpixel. Da Methoden zur Kantenverfolgung rechenintensiv sind, entschieden sich Scaramuzza et al. für eine Vereinfachung: Aus der Tatsache, dass jede im Binärbild enthaltene Kante nur einen Pixel

breit sein kann, ergibt sich, dass in einem störungsfreien Untersuchungsfenster mit genau einem fortlaufenden Kantenstück exakt $(2k+1)$ Kantenpixel vorliegen. Ist dies der Fall, wird davon ausgegangen, dass ein verbundenes Kantenstück vorliegt, andererseits wird die Bearbeitung des aktuellen Fensters beendet.

Als nächstes soll überprüft werden, ob die im Fenster enthaltene Kante bogenförmig oder gerade verläuft. Es werden zwei Kantenendpunkte $A := (x_A; y_A)$ und $B := (x_B; y_B)$ durch das Bestimmen zweier Koordinatenpositionen, die jeweils den größten Abstand zum Koordinatenursprung des Fensters besitzen. Zudem müssen die Kantenendpunkt jeweils mindestens ein verschiedenes Vorzeichen in den Paaren (x_A, x_B) und (y_A, y_B) besitzen (siehe Abbildung 3.5b). Die unterschiedlichen Vorzeichen seien notwendig, um Störungen durch vereinzelt auftretende Kantenpixel und durch verwinkelte Kantenstücke zu vermeiden [SPV05]. Die Distanz eines Punktes zum Koordinatenursprung wird mittels der Formel 3.1 berechnet.

$$d(x,y) = |x| + |y| \tag{3.1}$$

Anschließend wird eine Verbindungslinie $\overline{AB}$ zwischen den Kantenendpunkten berechnet. Wenn diese den Koordinatenursprung passiert, bedeutet dies, dass die Punkte A, B, und $(0;0)$ auf einer Geraden liegen (siehe Abbildung 3.6).

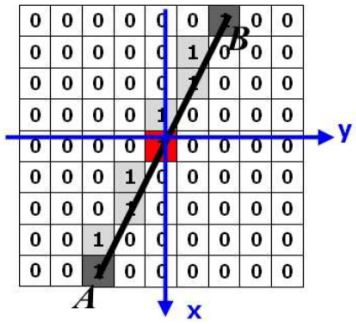


Abbildung 3.6: Überprüfung, ob ein gerades Kantenstück vorliegt [SPV05].

Zur Überprüfung, ob eine Linie $\overline{AB}$ einen Punkt $(0;0)$ durchquert, kann die Formel 3.2 verwendet werden.

$$y_B(x_B - x_A) - x_B(y_B - y_A) = 0 \tag{3.2}$$

Da in diesem Fall aber in einem diskreten Koordinatenraum gerechnet wird, verwendeten Scaramuuza et al. eine hierfür angepasste Gleichung [SPV05]:

$$\left|y_B(x_B - x_A) - x_B(y_B - y_A)\right| \leq \min\left(|x_B - x_A|, |y_B - y_A|\right) \tag{3.3}$$

Ist Gleichung 3.3 erfüllt, liegt ein gerades Kantenstück vor und die Bearbeitung des aktuellen Untersuchungsfensters wird beendet. Andernfalls wird von einer bogenförmig verlaufenden Kante ausgegangen.

In einem Fenster, in dem eine Bogenkante identifiziert wurde, wird nun mittels Geometrie die Richtung der Bogenform der Kante berechnet, um die Position des hypothetisch zugehörigen Kreises zu ermitteln. Hierfür wird der Mittelpunkt C der Linie $\overline{AB}$ mithilfe von Formel 3.4 berechnet, sowie die Senkrechte zu $\overline{AB}$ durch diesen Punkt C ermittelt.

$$C = \left(\frac{x_A + x_B}{2}; \frac{y_A + y_B}{2}\right) \tag{3.4}$$

Abbildung 3.5b zeigt beispielhaft ein Untersuchungsfenster, in welchem für die enthaltene Kante die berechnete Linie $\overline{AB}$ in rot und die Senkrechte in blau eingezeichnet sind. Die Position des Punkts C wird genutzt, um vom Koordinatenursprung aus eine Linie durch C entlang der Senkrechten in den Ausgaberaum zu schreiben. Ist dies abgeschlossen, wird das Verfahren mit dem dem nächsten Kantenpunkt wiederholt.

(a) Original (b) Ausgaberaum (c) Überlagertes Kantenbild

Abbildung 3.7: Resultate des pixel2pixel-Algorithmus [SPV05].

Sind alle Kantenpunkte auf diese Weise verarbeitet worden, existiert im Ausgaberaum eine Linie für jedes als Bogenkante identifiziertes Kantenstück. Dort, wo sich die meisten Linien überschneiden, liegt ein Maximum vor. Von diesem wird angenommen, dass es im Mittelpunkt eines sich im Bild befindlichen Kreises befindet. Da sich die zu einem Kreis zugehörigen, einen Pixel breiten Linien jedoch nicht alle in exakt einem Bildpunkt schneiden würden, wird vor Bestimmung des Maximums noch ein Boxfilter zum Weichzeichnen auf den Ausgaberaum angewendet. Dies ist ein Filter, bei welchem jeder Pixel im Operatorfenster des Filters den Durchschnittswert aller im Operatorfenster enthalte-

nen Pixel erhält [Aur11]. Abbildung 3.7 zeigt eine beispielhafte Visaulisierung von der Ausgabe des Algorithmus.

Zur Funktionsweise des Algorithmus seien zuletzt noch zwei Dinge hervorzuheben: Er besitzt zum Einen den einstellbaren Parameter k zur Änderung der Größe des Untersuchungsfensters. Dieser Parameter bestimmt damit indirekt die Größe der Kreise, die der Algorithmus detektieren kann. Wird er zu klein gewählt, können keine sinnvollen Ergebnisse erzielt werden, da stets zu wenig Pixel betrachtet werden. Wird k hingegen ein zu großer Wert für k verwendet, liefert der Algorithmus ebenfalls keine guten Ergebnisse, da er so entworfen wurde, dass nur Teile von Bogenkanten in einem Untersuchungsfenster liegen dürfen. Befinden sich Halbkreise oder auch ganz Kreise in einem Untersuchungsfenster, funktionieren die Verarbeitungsschritte des Algorithmus nicht mehr. Dies hängt selbstverständlich auch immer von der Größe ab, mit der sich ein Kreis im Kantenbild wiederfindet. Der Parameter ist voreingestellt und während der Laufzeit konstant; dies könnte für Einschränkungen in der Detektionsfähigkeit für Bälle verschiedener Größen und in verschiedenen Distanzen sorgen. Zum Anderen liefert der Algorithmus auch dann immer eine Ballposition, wenn kein Ball vorhanden ist. Da sich durch Objekte im Sichtfeld des Roboters immer Konturen im Kamerabild feststellen lassen, ergeben sich bei Abwesenheit eines Balls zufällige Maxima im Ausgaberaum und somit zufällig detektierte Positionen.

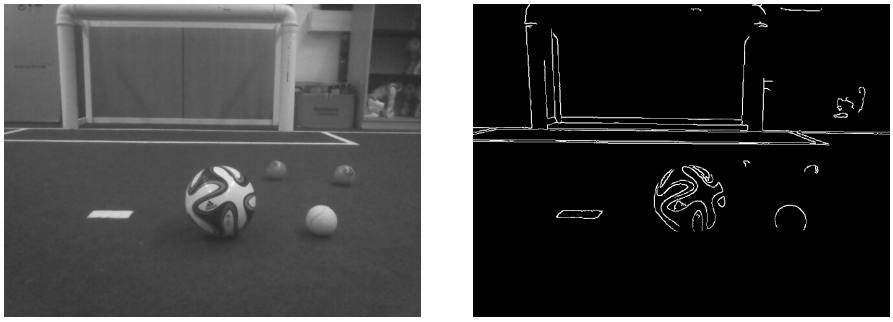
3.3.3 Anpassungen und Implementierung

Der Algorithmus wurde in C++ direkt auf der NAO-Robotikplattform implementiert. Eine Offline-Implementierung auf einem Computersystem wäre auch möglich gewesen, würde aber die laufzeitkritische Komponente der Entwicklung nicht korrekt widerspiegeln. Das Testen direkt auf dem NAO ermöglichte es, schon während früher Tests Laufzeitänderungen zu beobachten. Um den Algorithmus komplett realisieren zu können, reichte die vom HULKs-Framework auf dem Roboter gebotene Funktionalität allerdings noch nicht aus. Zum Erstellen des Kantenbildes wurde daher die Open-Source-Grafikbibliothek OpenCV verwendet [Its15]. Sie bietet für den Bereich der digitalen Bildverarbeiten eine Vielzahl bereits implementierter Algorithmen. Das Kantenbild ließ sich mithilfe der OpenCV-Funktion *Canny* leicht erzeugen. Ein Kompromiss musste hier allerdings gemacht werden: Da das interne Bild-Datenformat von dem abweicht, welches OpenCV verwendet, muss eine Konvertierung der Kamera-Bilddaten in das *Mat*-Format von OpenCV stattfinden. Dies bedeutet zusätzlich benötigte Rechenzeit pro Programmzyklus. Sobald der Canny-Algorithmus in das HULKs-Framework integriert wird, ließe sich diese Konvertierung vermeiden.

Ein weiteres Detail der Implementierung ist die Art und Weise, wie die Kantenpunkte behandelt werden, welche am Rand des Bildes liegen. Würden diese Kantenpunkte ein Untersuchungsfenster hervorrufen, welches über den Bildrand hinausragt, so wird die Untersuchung dieser Punkte abgebrochen. Das weitergehende Untersuchen

der Randpunkte würde zusätzliche Überprüfungen notwendig machen, um zu vermeiden, dass nicht auf unerlaubte Speicherbereiche außerhalb der Bilddaten zugegriffen wird. Des Weiteren sind die vom NAO aufgenommenen Kamerabilder am Rand stärker verrauscht und dunkler als in der Mitte, weswegen die Daten an den Rändern ohnehin fehlerbehaftet sind. Außerdem liefert der pixel2pixel-Algorithmus schon wegen seiner Funktionsweise her schlechtere Ergebnisse, wenn eine unvollständige Kreisform (wie zB. ein Ball am Rande des Sichtfeldes) vorliegt, da sich die Ausgabe-Linien entweder gar nicht oder nur schwach innerhalb des Ausgaberaums schneiden. Zuletzt sei hierzu gesagt, dass Scaramuzza et al. in ihrem Paper nicht beschreiben, wie mit Randpunkten verfahren werden soll.

Nach erfolgter Implementierung des Algorithmus, so wie er in der Arbeit der Forscher beschrieben wurde, war es schon früh ersichtlich, dass das bloße Erzeugen eines Kantenbildes auf Grundlage der Originaldaten keine zufriedenstellenden Ergebnisse liefert. Abbildung 3.8 zeigt den Y-Kanal eines vom NAO aufgenommenen Kamerabildes und das dazugehörige Kantenbild.



(a) (b)

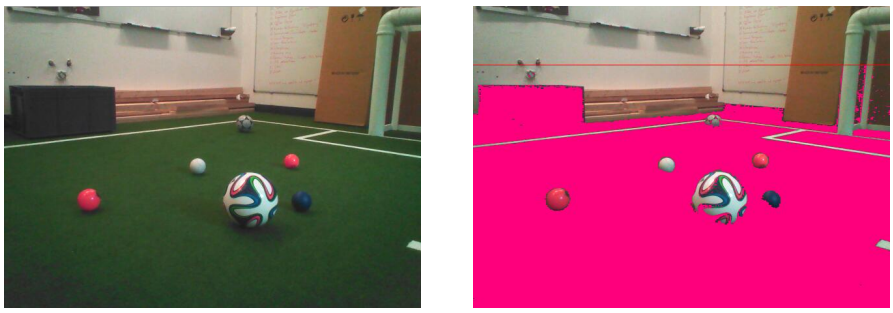
Abbildung 3.8: Eine Aufnahme der NAO-Kamera, dargestellt als Y-Kanal (a) und als das zugehörige Kantenbild (b).

In Abbildung 3.8a sind ein realistischer und ein weißer Ball im Vordergrund, sowie zwei rote Bälle im Hintergrund abgebildet, während in Abbildung 3.8b das mit optimierten Parametern erzeugte Ergebnis der Kantendetektion zu sehen ist. Es ist zu erkennen, dass lediglich der weiße Ball eine kreisförmige Kante im Kantenbild hervorruft. Die anderen Bälle besitzen im Y-Kanal wenig Kontrast, wodurch die Außenkonturen dieser Bälle schlecht als Kanten detektiert werden. Der realistische Ball besitzt zudem viele markante Stellen, die das Erkennen einer Außenkontur zusätzlich erschweren. Das einfache Erzeugen eines Kantenbildes bringt also kein Bild hervor, auf dem die Formen der Bälle zu erkennen sind. In der Arbeit zu dem Algorithmus von Scaramuzza et al. befindet sich keine Erwähnung darüber, welcher Kantendetektionsalgorithmus verwendet wurde. Es ist also möglich, dass ein anderer Algorithmus als Canny für das Erzeugen eines Kantenbildes zum Einsatz kam. Weiterhin könnten die Versuche der Forscher stets mit

einem einem roten Testball bei hellem Hintergrund und guten Kontrasten stattgefunden haben. Außerdem könnte für das Erstellen des Kantenbilds der Rot-Kanal (bei einem RGB-Bild) bzw. der Cr-Kanal (bei einem YCbCr-Bild) verwendet worden sein. Dies dürfte zu weitaus zuverlässigeren Kantenbildern führen. Im dem Einleitungstext dieses Kapitels wurde jedoch bereits erklärt, dass für diese Arbeit lediglich der Y-Kanal verwendet werden soll, welcher für verschiedenfarbige Bälle aber nicht immer zuverlässige Kanten hervorbringt.

Zur Lösung des Problems wurde unabhängig des Papers von Scaramuzza et al. eine neue Methode entwickelt, welche zum Zwecke der Kantendetektion zunächst ein binäres Bild erstellt, bei welchem die Bälle als Vordergrundobjekte und das Spielfeld als Hintergrund voneinander getrennt werden. Hierfür wird eine zuvor bereits im HULKs-Framework implementierte Funktion verwendet, welche per Histogramm-Auswertung die Spielfeldfarbe auf einem Kamerabild ermitteln kann.

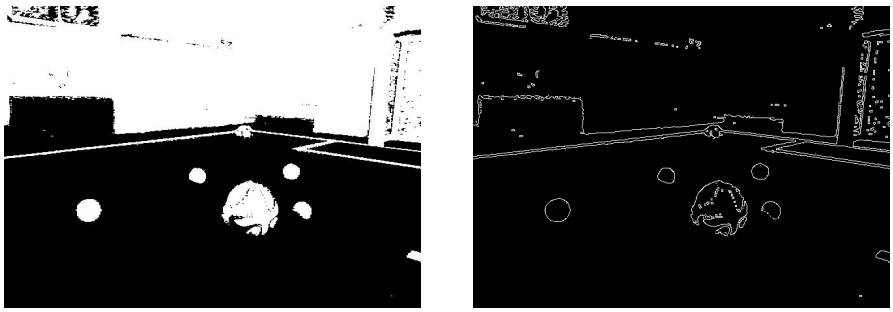
Abbildung 3.9 zeigt das Ergebnis der Feldfarberkennung. Diese wurde mit hohen Toleranzwerten für die Farbe eingestellt, was dazu führt, dass zum Teil auch nicht-Spiel-Elemente wie die Kiste am Spielfeldrand mit eingeschlossen werden. Außerdem ist festzustellen, dass Teile des blauen Balls in Abbildung 3.9b nicht mehr sichtbar sind. Jedoch sollen sich die verschiedenen Bälle möglichst gut für die Erzeugung des Binärbilds vom Feld abheben, daher musste ein Kompromiss für diese Einstellung der Toleranzen gewählt werden.



(a) (b)

Abbildung 3.9: Bild (a) zeigt die Originalaufnahme der Bälle und (b) die detektierte Feldfarbe, eingefärbt.

Alle Teile des Bildes, die nicht zu der Feldfarbe gehören, werden in einem Binärbild als Vordergrund und alle anderen Teile als Hintergrund klassifiziert. Hiermit sollte erreicht werden, dass die Bälle auf dem Feld unabhängig von Farbe und Muster über ihre Form identifiziert werden können. So erzeugt diese Methode ein Bild, auf dem die Bälle gut als weiße Flächen zu erkennen sind. Nach Anwendung der Kantendetektion auf dieses Binärbild ergibt sich ein Kantenbild mit besser erkennbaren Ballkonturen (siehe Abbildung 3.10).



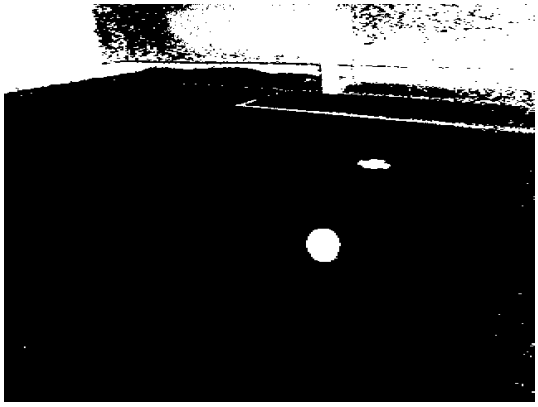
(a) (b)

Abbildung 3.10: Das binäre Vordergrund- / Hintergrund-Bild (a) und das Ergebnis der Kantendetektion (b).

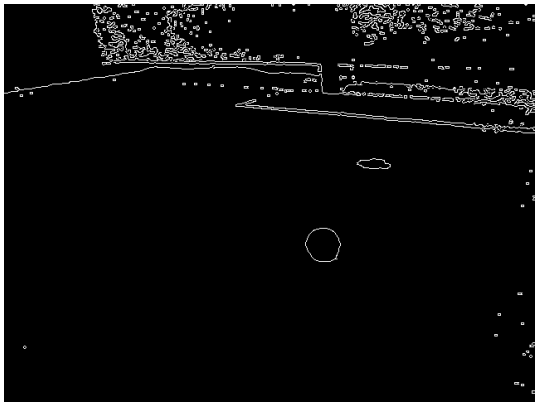
Es fällt auf, dass nun fast alles, was sich nicht auf dem Spielfeld befindet, als „Vordergrund“ klassifiert wurde. Die Betrachtung des Kantenbildes zeigt allerdings, dass dies sogar vorteilhaft sein kann, da das Auftreten störender Kanten außerhalb des Spielfeldes zum Teil minimiert wird. Außerdem ist festzustellen, dass der im Vordergrund liegende realistische Ball auf dem Kantenbild in Abbildung 3.10b im Gegensatz zu den anderen Bällen keine besonders runde Außenkontur, sowie störende Kanten innerhalb seiner Fläche besitzt. Grund hierfür ist, dass sich ein gemusterter Ball unter ausschließlicher Verwendung der Feldfarbdetektion immer noch nicht ausreichend gut vom Spielfeld hervorheben lässt. Es liegt die Vermutung nahe, dass sich ein realistischer Spielball mittels der hier implementierten Methode nur schlecht detektieren lässt. Trotzdem enthält das aus dem Binärbild hervorgebrachte Kantenbild besser erkennbare kreisförmige Kanten als eines, welches nur auf Grundlage des Y-Kanals erzeugt wurde. Aus diesem Grunde wird im Folgenden nur noch die neu entwickelte Lösung mit der vorangehenden Binärbild-Erzeugung verwendet. Zuletzt wird in Abbildung 3.11 noch gezeigt, wie die Bildausgaben der Implementierung auf dem NAO im Beispiel aussieht.

Die anschließend folgende Evaluation soll, zusammen mit der Diskussion der Ergebnisse, genauen Aufschluss darüber liefern, wie gut die implementierte Lösung tatsächlich funktioniert.

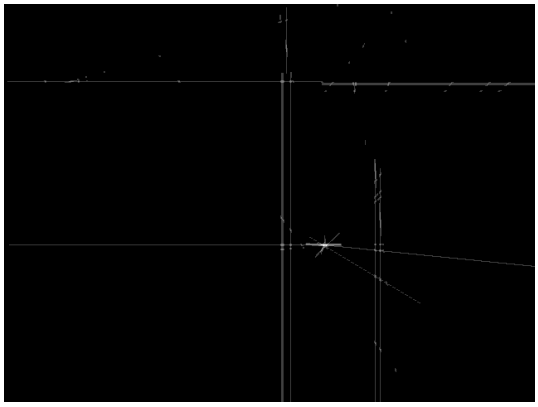
(a) Kamerabild



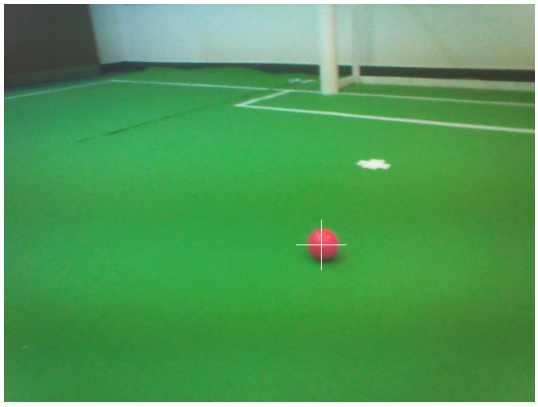
(b) Binärbild



(c) Kantenbild



(d) Algorithmus-Ausgaberaum



(e) Detektierte Ballposition

Abbildung 3.11: Die verschiedenen Bearbeitungsschritte und Ausgaben des Algorithmus. Die erfolgreich detektierte Ballposition wurde in (e) markiert.

Kapitel 4

Evaluation und Diskussion

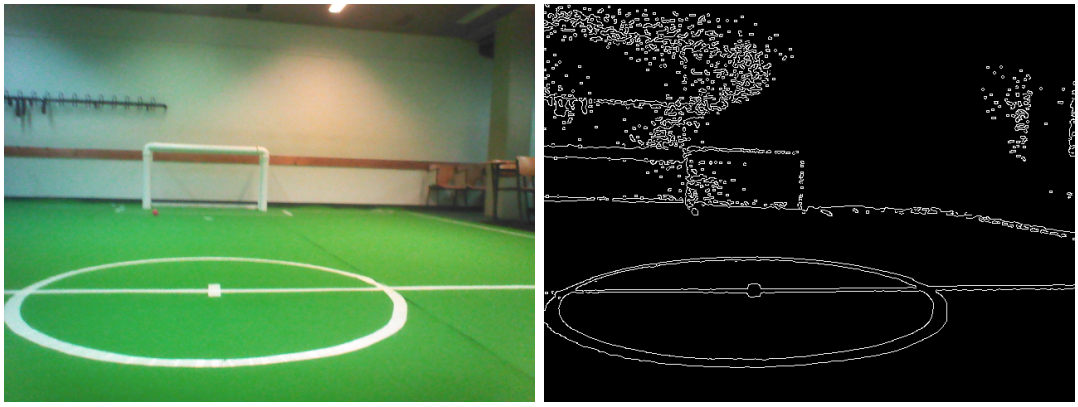
In diesem Kapitel soll untersucht werden, wie hoch die Erkennungsrate der implementierten Balldetektion ist. Zunächst wird beschrieben, wie die Ergebnismessungen durchgeführt wurden und auf welche Aspekte dabei Wert gelegt wurde. Die ermittelten Ergebnisse sollen zudem erklärt und anschließend diskutiert werden. Zuletzt werden unter Betrachtung der Ergebnisse einige Vorteile und Probleme, sowie mögliche Änderungen des Algorithmus besprochen.

4.1 Versuchsdurchführung

Zum Ermitteln der Erfolgsquote der Balldetektion wurde ein dem SPL-Standard entsprechendes Spielfeld aufgebaut. Auf diesem sollte der Algorithmus mit jeweils verschiedenen Parametern für k sowie mit verschiedenartigen Bällen getestet werden. Außerdem wurden auch unterschiedliche Lichtbedingungen getestet, um zu untersuchen, wie stark die Erkennungsleistung mit guter Beleuchtung der Umgebung zusammenhängt.

Ein NAO wurde wiederholt an Positionen des Feldes gestellt, von welchen aus ein Ball in ca. 3-4 Meter gesehen werden konnte. Weiter entfernte Distanzen stellten sich für diesen Algorithmus als problematisch heraus, selbst mit klein gewähltem Parameter k. Abbildung 4.1 zeigt den Grund hierfür. Der Ball ist in dem Kantenbilder bei solch einer Distanz nicht mehr vom Störrauschen im Hintergrund des Bildes zu unterscheiden.

Um weiterhin einen realistischen Einsatz zu simulieren, wurde dem NAO nach jeder Positionierung ein Bewegungsbefehl gegeben, welcher ihn aus seiner Startposition auf den Ball zulaufen ließ. Dieser Vorgang wurde vier mal mit verschiedenen Bällen, drei mal mit verschiedenen Beleuchtungssituationen und einmal mit weiterem NAO als Störfaktor im Bild wiederholt. Hierbei wurde eine großflächige und gleichmäßige Beleuchtung des Spielfeldes als gut bezeichnet, während eine ungleichmäßige und weniger Lichtstarke Beleuchtung als schlecht eingestuft wurde. Während der ganzen Versuchs-



(a) (b)

Abbildung 4.1: Der rote Ball liegt vor dem linken Torpfosten, in weiter Entfernung vom NAO.

dauer wurden dabei Kamerabilder gespeichert, welche von der oberen Kamera im Kopf des NAO aufgenommen wurden.

Die so entstandene, mehrere tausend Bilder lange Bildfolge, wurde anschließend in sieben kleine Sequenzen der verschiedenen Situationen, mit einer Länge von 50 bis 25 Bildern, eingeteilt (siehe Abbildung 4.3). 50 Bilder wurden für die Sequenz mit rotem Ball und guter Beleuchtung gewählt, da diese das bisher offizielle SPL-Spiel am Besten repräsentiert und hier genauere Daten ermittelt werden sollten. Für die restlichen Sequenzen wurden 25 Bilder verwendet, um den Arbeitsaufwand für das Erstellen der Ground-Truth-Daten in einem realisierbaren Rahmen zu halten. Außerdem ist bereits mit 25 Bildern erkennbar, wie zuverlässig eine Balldetektion in Standardsituationen funktioniert. Alle Sequenzen enthalten hierbei nur solche Bilder, auf denen der Ball auch enthalten ist. Wie in Sektion 3.3.2 bereits erklärt, ergibt jedes Bild, das keinen Ball enthält, ohne Ausnahme eine Falscherkennung (wie dies vermieden werden kann, soll zu einem späteren Zeitpunkt erklärt werden). Aus diesem Grund ist nur die Untersuchung jener Fälle sinnvoll, in welchen der Ball tatsächlich im Sichtfeld des Roboters liegt.

Zu jedem Bild der sieben Test-Sequenzen wurde mit dem in MATLAB enthaltenen *Training Image Labeler* jeweils eine rechteckige Region um den Ball herum als Ground-Truth-Daten, zum Zwecke des späteren Vergleichs, markiert (siehe Abbildung 4.2). Alle Bilder der sieben Test-Sequenzen wurden anschließend auf dem NAO anstelle des Kamerabildes zur Verfügung gestellt, wodurch diese vom implementierten Algorithmus direkt auf dem Roboter bearbeitet wurden. Der Algorithmus wurde jeweils für die Parameter $k = 2$, $k = 4$, $k = 6$ und $k = 8$ ausgeführt und die detektierten Ballpositionen pro Bild und Parametereinstellung gespeichert. Der Wertebereich für k wurde zwischen 2 und 8 gewählt, da dies einer Untersuchungsfenstergröße von 5 bis 17 Pixeln entspricht

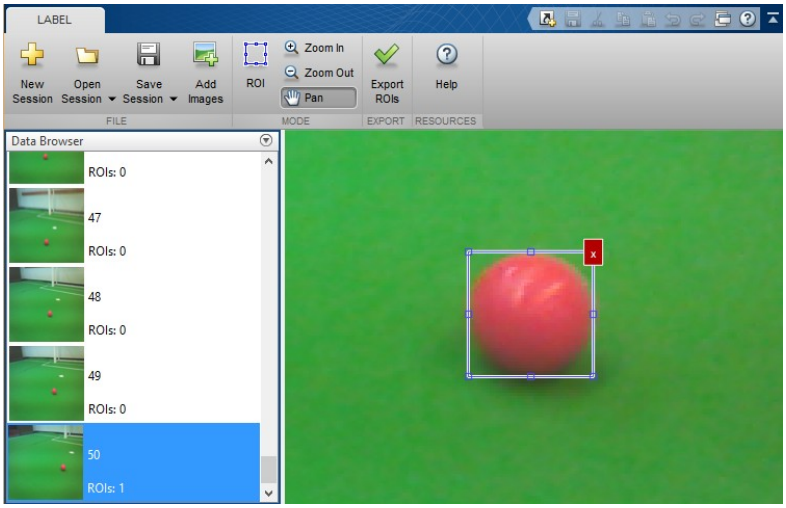


Abbildung 4.2: Der *MATLAB Training Image Labeler* und eine markierte Ground-Truth-Region.

und der Ballradius bei den gegebenen Distanzen in einem bereich von 6 bis 27 Pixeln lag. Eine kurze Evaluation größerer Werte für k zeigte außerdem schnell, dass der Algorithmus keine brauchbaren Ergebnisse mehr erzeugte.

4.2 Ergebnisse

Zur Bewertung der Leistungsfähigkeit des Algorithmus bei sichtbarem Ball wurden zwei Werte berechnet: Die positive Detektionsrate und die negative Detektionsrate. Hierfür sind pro Bild zwei Fälle zu unterscheiden: Entweder wurde eine Ballposition gefunden, welche im spezifizierten Ground-Truth-Bereich liegt, oder es wurde eine falsche Position außerhalb dieses Bereichs detektiert. Zum Ermitteln der Erkennungsrate wurde für jede Bildsequenz pro Paramatereinstellung für k gezählt, wie viele positive Detektionen vorliegen. Diese Anzahl $N_{positiv}$ wurde gemäß Formel 4.1 in ein Verhältnis zur Gesamtzahl N_{Gesamt} der Bilder einer Sequenz gesetzt, um die prozentuale positive Detektionsrate PD zu erhalten.

$$PD = \frac{N_{positiv}}{N_{Gesamt}} \cdot 100 \tag{4.1}$$

Hieraus kann nun einfach die prozentuale negative Detektionsrate ND ermittelt werden:

$$ND = 100 - PD. \tag{4.2}$$

(a) Gute Beleuchtung und roter Ball

(b) Gute Beleuchtung und weißer Ball

(c) Gute Beleuchtung und realistischer Ball

(d) Gute Beleuchtung und blauer Ball

(e) Gute Beleuchtung, NAO und roter Ball

(f) Schlechte Beleuchtung und roter Ball

(g) Schlechte Beleuchtung und roter Ball

Abbildung 4.3: Beispielbilder der aufgenommenen Sequenzen.

Da in den Testsequenzen der Ball allerdings immer im Bild ist, und eine Detektion, die nicht korrekt ausfällt, somit zwangsweise automatisch eine falsche Detektion ist, soll im Folgenden nur noch die positive Detektionsrate (im Anschluss Erkennungsrate genannt) als Maß für die Qualität der Ergebnisse angegeben werden. Es folgen die ermittelten Versuchsergebnisse für jede dieser Sequenzen.

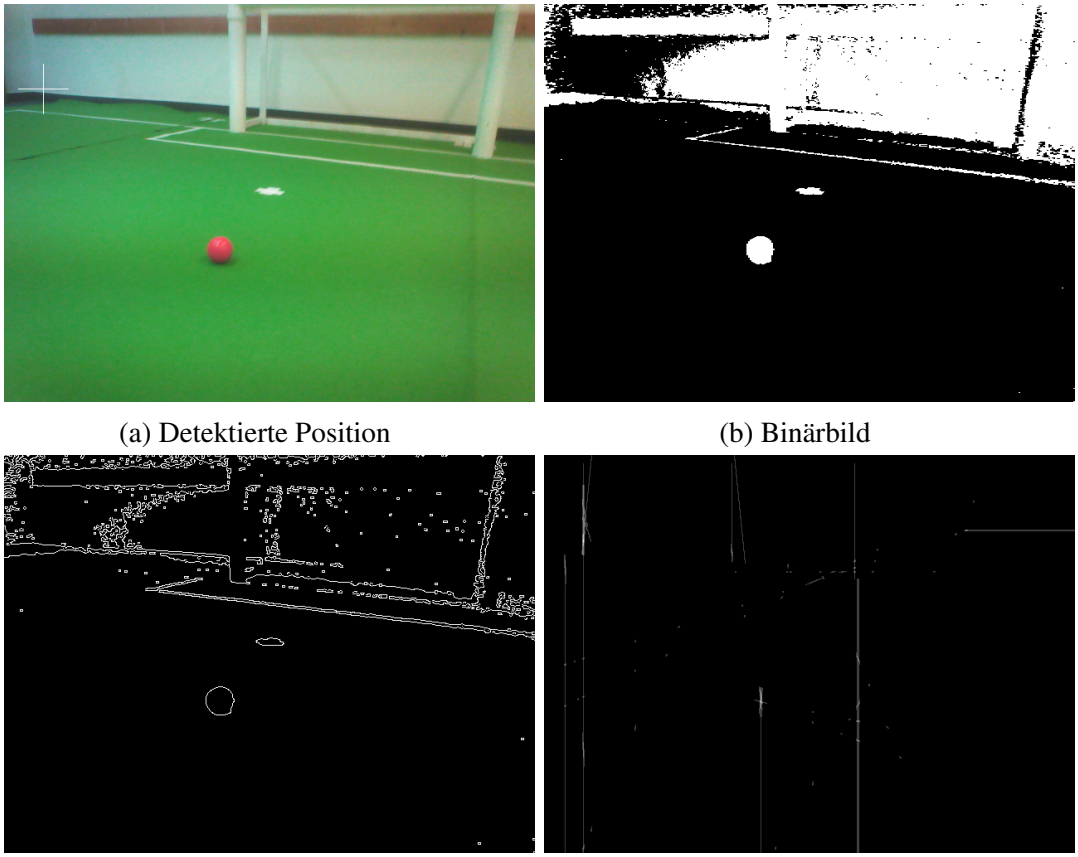
4.2.1 Balldetektion bei guter Beleuchtung

Roter Ball		**Weißer Ball**		**Realistischer Ball**	
k	Erkennungsrate	k	Erkennungsrate	k	Erkennungsrate
2	0%	2	0%	2	0%
4	48%	4	16%	4	4%
6	52%	6	4%	6	8%
8	26%	8	0%	8	4%

Blauer Ball		**Roter Ball und NAO**	
k	PE	k	Erkennungsrate
2	0%	2	0%
4	0%	4	16%
6	0%	6	40%
8	0%	8	0%

Tabelle 4.1: Die Erkennungsraten der Balldetektion bei guter Beleuchtung.

Wie in Tabelle 4.1 zu erkennen ist, liegt bei guter Beleuchtung die beste erreichte Erkennungsrate bei lediglich 52%. Da der Algorithmus umso besser funktioniert, je kreisförmiger die Ballkonturen im Kantenbild sind, bieten die rauschbehafteten Kamera-Bilder des NAO anscheinend keine gute Datengrundlage für diesen Algorithmus. Um dies zu überprüfen, werden Ausgabebilder der Zwischenschritte des Algorithmus betrachtet (siehe Abbildung 4.4).



(a) Detektierte Position (b) Binärbild

(c) Kantenbild (d) Algorithmus-Ausgaberaum

Abbildung 4.4: Die Ausgabe der Algorithmus-Zwischenschritte bei inkorrekter Detektion.

Das Binärbild 4.4b und das daraus resultierende Kantenbild 4.4c wirken zunächst so, als wäre der Ball dort als runde Form gut erkennbar. Der Ausgaberaum 4.4d zeigt aber, wo das Problem liegt: Der Linienschnittpunkt an der Position des Balls ist nicht stark genug ausgeprägt. Stattdessen liegt an einer mit Bildrauschen besetzten Stelle das Maximum. Bei genauerer Betrachtung wird ersichtlich, dass die Ballkontur im Kantenbild Unregelmäßigkeiten aufweist. Diese dürften dazu führen, dass im Algorithmus-Ausgaberaum nicht genug Linien von den Bogen-Kantenstücken des Balls in Richtung seines Mittelpunktes geschrieben wurden. Da diese Unregelmäßigkeiten immer im Kantenbild auftreten, ist die Erkennungsleistung des Algorithmus in vielen Fällen gering. Wie in Sektion 3.3.3 erklärt wurde, besitzt ein realistischer Ball im Kantenbild keine kreisrunde Kontur und wurde daher bei bester Parametereinstellung nur in 8% der Fälle detektiert. Auffällig ist noch, dass der blaue Ball in keinem Fall erkannt werden konnte. Die Ursache hierfür ist die verwendete Methode der Hintergrund- / Vordergrund-Trennung, welche die Feldfarbe als Grundlage nutzt. Die grüne Feldfarbe liegt im YCbCr-Raum nicht

weit entfernt von blauen Farben (siehe Abbildung 2.6, mittleres Diagramm). Durch die Nähe im Farbraum und aufgrund von großen Farbtoleranzen bei der Ermittlung der Pixel, die zum Spielfeld gehören, wird der blaue Ball im Binärbild als Hintergrund klassifiziert. Somit ist dieser Ball auf dem Kantenbild gar nicht mehr sichtbar. Ein weiterer Unterschied ist bei den Erkennnungsraten des roten Balls sichtbar, wenn ein NAO als Störfaktor hinter dem Ball steht.

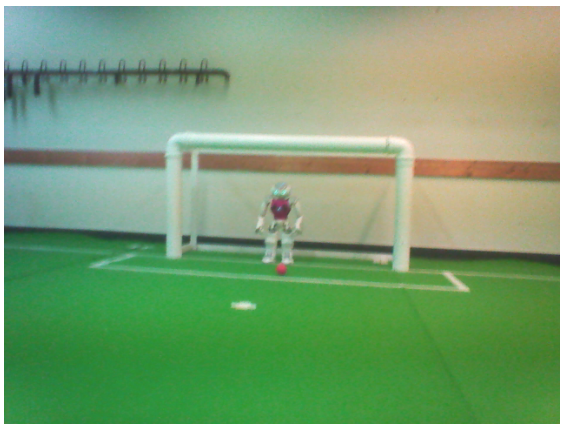


Abbildung 4.5: Roter Spielball vor einem NAO

Die Erkennungsrate dürfte bei der Sequenz mit dem NAO im Hintergrund wohl deshalb niedriger sein, weil der NAO hinter dem Ball dessen Außenkontur im Bild stört. Bei der Vordergrund- / Hintergrund-Trennung des Algorithmus werden der Ball sowie Teile des NAO als Vordergrund klassifiziert. Hierdurch das Entstehen einer klaren Kreisform des Balls im Binärbild verhindert.

Zuletzt lässt sich bei übergreifender Betrachtung der Messergebnisse erkennen, dass der Parameter k in den überprüften Situationen scheinbar mit einem Wert von 6 am Besten besetzt ist. Es bestünde die Möglichkeit, dass die Einstellung $k = 5$ noch bessere Ergebnisse liefert. Diese Überprüfung wurde angesichts der ohnehin schon geringen Erkennungsleistung nicht mehr vorgenommen, da keine sprunghaften Verbesserungen zu erwarten waren.

4.2.2 Balldetektion bei schlechter Beleuchtung

Roter Ball, Lichtsituation 1		**Roter Ball, Lichtsituation 2**	
k	Erkennungsrate	k	Erkennungsrate
2	0%	2	0%
4	4%	4	20%
6	0%	6	72%
8	4%	8	36%

Tabelle 4.2: Die Erkennungsraten der Balldetektion bei schlechter Beleuchtung.

Interessanterweise konnte der rote Spielball bei der in Abbildung 4.3g gezeigten Beleuchtungssituation in 72% der Fälle erkannt werden. Der Algorithmus liefert hier ein besseres Ergebnis als bei gleichmäßig starker Ausleuchtung des Feldes. Dies zeigt widerrum, dass die entwickelte Methode nicht unabhängig vom Licht gleichermaßen zuverlässig funktioniert. Grund hierfür dürfte die Feldfarberkennung sein, welche bei unterschiedlichen Lichtbedingungen auch unterschiedlich gut die Feldfarbe erkennen kann. Schließlich sorgen veränderte Lichtbedingungen immer auch für Änderungen der Grüntöne im Spielfeld, auf welche die Feldfarberkennung zum Teil empfindlich reagiert.

Die in Abbildung 4.3f gezeigte Situation unterbindet im Gegensatz dazu fast jegliche Erkennung des Spielballs. In den Bildern dieser Sequenz herrschte zwar eine veränderte Beleuchtung vor, jedoch sollte die Änderung nicht groß genug sein um solche Leistungseinbußen in der Balldetektion zu verursachen. Auf allen Bildern der Sequenz liegt der Ball jedoch auf einer Spielfeldlinie, welche störenden Einfluss hat. Da die Linie, so wie der Ball, nicht zu der Feldfarbe gehören, werden sie beide bei der Binärbilderzeugung als Vordergrund klassifiziert. Ähnlich wie in der Situation, bei der ein NAO hinter dem Ball steht, ergibt sich um den Ball auch hier keine kreisförmige Kontur mehr. Dies zeigt schlussendlich, dass der entwickelte Lösungsansatz störanfällig gegenüber allen Spielsituationen ist, in denen sich der Spielball nicht rundherum von dem grünen Spielfeld abhebt.

4.2.3 Laufzeit

An dieser Stelle werden die Ergebnisse von Laufzeitmessungen präsentiert, welche beim Aufnehmen der Testsequenzen gemessen wurden. Um festzustellen, ob der Algorithmus in Echtzeit einsetzbar ist, wurde die Laufzeit pro Bild über die aus 50 Bildern bestehende Sequenz 4.3a hinweg ermittelt. Zusätzlich sollte überprüft werden, ob die Änderung des Parameters k Laufzeiteinbußen bringt. Hierfür wurde für jede Paramter-

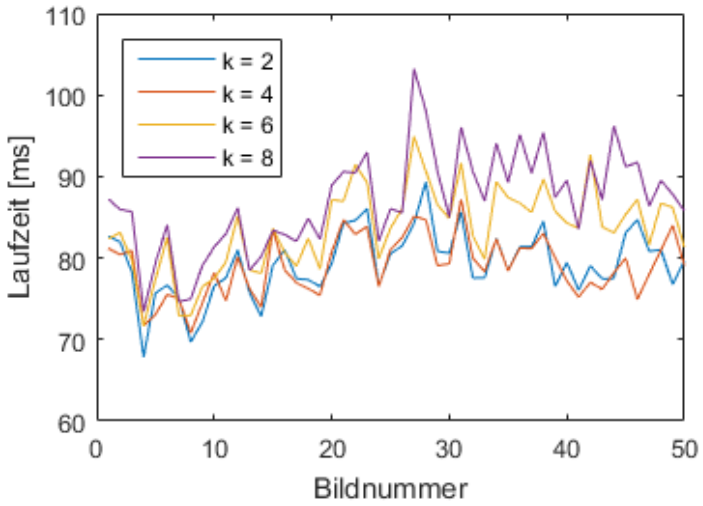


Abbildung 4.6: Parameterabhängige Laufzeitveränderungen bei Verarbeitung einer Bildsequenz

einstellung jeweils die minimale (Min), maximale (Max) und durchschnittliche (Avg) Laufzeit ermittelt.
Es ist zu erkennen, dass größere Werte für k zu längeren Laufzeiten führen. Die größere Menge an Pixeln, die bei großen Parameterwerten verarbeitet werden muss, sorgt dabei für eine erhöhte Rechenzeit. Bei Wahl des Wertes $k = 6$, welcher zu den besten Ergebnissen führt, ist im Schnitt mit einer Laufzeit von 83,6 ms zu rechnen. Auf dieser Grundlage ließe sich davon ausgehen, dass der Algorithmus pro Sekunde elf bis zwölf mal ausgeführt werden könnte. Eine genauere Betrachtung hat gezeigt, dass hierbei jeweils ca. 25-30 ms für das Erstellen des Binärbilds, des Kantenbilds sowie für den pixel2pixel-Algorithmus an sich benötigt werden. Es ist jedoch zu beachten, dass noch weitere Programmbestandteile der HULKs-Software für Bildverarbeitung, Bewegungssteuerung und künstliche Intelligenz pro Programmzyklus Prozessorzeit für sich beanspruchen. Diese benötigen insgesamt ca. 15 ms Rechenzeit, wodurch die Balldetek-

k	Min [ms]	Max [ms]	Avg [ms]
2	67,9	89,3	79,3
4	70,8	87,2	79,0
6	71,6	94,9	83,6
8	73,4	103,2	87,0

Tabelle 4.3: Laufzeiten für verschiedene Parametereinstellungen

tion noch ca. 10 mal pro Sekunde ausgeführt werden kann. Dies wäre ausreichend, um einen Ball während eines Fußballspiels in Echtzeit zu verfolgen. Wünschenswert wäre aber eine höhere Ausführungsfrequenz, weil dann auch schnell bewegte Bälle zuverlässiger erkannt werden können. Außerdem lässt sich in Anbetracht der relativ geringen Laufzeit aller anderen Module feststellen, dass die hier implementierte Lösung die Geschwindigkeit des ganzen System beträchtlich ausbremsen würde. Eine Laufzeit von nur wenigen Millisekunden, wie es beim farbbasierten Ansatz der Ballerkennung möglich ist (siehe Sektion 3.1), wäre also erstrebenswert. Da bei der Implementierung noch nicht besonderes Augenmerk auf besonders effiziente Vorgehensweisen gelegt wurde, ist es sicherlich möglich, die Laufzeit noch zu verbessern.

4.3 Diskussion der Ergebnisse

Wie bei der Betrachtung der Ergebnisse zu erkennen war, bietet der verwendete Lösungsansatz keine besonders hohe Erfolgsquote bei der Erkennung eines Spielballs. Unter speziellen Lichtbedingungen und mit dem roten Ball konnte zwar eine Erkennungsrate von 72% erreicht werden, doch auch dies reicht nicht für ein zuverlässiges Fußballspiel aus. Um mit einem NAO erfolgreich spielen zu können, muss besonders die Ballerkennungsrate möglichst hoch liegen, da er ansonsten sein Hauptziel nicht verfolgen kann. Außerdem konnte festgestellt werden, dass der gewählte Lösungsansatz nicht geeignet dafür ist, einen realistischen Spielball zu erkennen. Es lässt sich also Schlussfolgern, dass die verwendete Methode in dieser Form keinen Einsatz beim Roboterfußball in der SPL finden kann. Weiterhin ist noch zu beachten, dass der Algorithmus immer falsche Ergebnisse liefert, wenn der Ball nicht im Sichtfeld ist. Zur Verhinderung dieser Falscherkennungen sind Erweiterung notwendig, welche von Scaramuzza et al. bereits in ihrer Veröffentlichung zum pixel2pixel-Algorithmus präsentiert wurden [SPV05]. So könnte beispielsweise mithilfe von Stereo-Sicht überprüft werden, ob beide Kameras den Ball an gleicher Stelle detektieren und somit zufällige Falscherkennungen ausgeschlossen werden. Zudem könnte auch die im 3D-Raum ermittelte Bewegung des Ballposition analysiert werden, um zu erkennen ob diese sich realistisch verhält oder Sprünge im Ortsraum durchführt. Dies ist auf dem NAO, der keine Stereo-Sicht bietet, aber nicht möglich. Stattdessen könnte eine weitere im Paper vorgestellte Methode verwendet werden, bei der ein zuvor detektierter Ball mithilfe von teilweisem Bildabgleich auf nachfolgenden Bildern wiedererkannt werden kann. Da dies allerdings einen großflächigen Vergleich von Bilddaten erfordert, könnte der Ansatz zu rechenintensiv für die NAO-Robotikplattform sein.

Der pixel2pixel-Ansatz hat trotzdem noch einige positive Merkmale, die an dieser Stelle hervorgehoben werden sollen. Die Funktionsweise ist leicht verständlich und bietet somit die Möglichkeit, eventuelle Fehler und Nachteile schnell zu Finden und eventuell auszubessern. Außerdem erfordert die Implementierung keinen großen Aufwand

und kann relativ schnell vorgenommen werden. Dies bietet gerade dann Vorteile, wenn mehrere Lösungsansätze getestet und miteinander verglichen werden sollen. Dies ist bei der Entwicklungsarbeit für den NAO äußerst nützlich, da es im Voraus oft schwer abschätzbar ist, wie gut ein Lösungsansatz tatsächlich funktioniert. Dass der Algorithmus als Hauptkriterium zum Detektieren von Bällen deren Form nutzt, macht ihn außerdem allgemein für das Lösen von Problemen der Kreisdetektion nutzbar. Es wäre denkbar, diesen Algorithmus in einem Kontext einzusetzen, in welchem besser erkennbare Kreiskonturen vorliegen. Des Weiteren ermittelt die verwendete Methode lediglich annähernd den Mittelpunkt eines Balls, nicht jedoch dessen Radius. Wäre dies der Fall, so könnten falsch detektierte Ballkandidaten mithilfe einer Überprüfung ihrer Größe zusätzlich ausgeschlossen werden.

Trotz der diversen Überprüfungen zum Identifizieren von Bogenkanten in einem Kantenbild, ist bei einer genaueren Untersuchung der Funktionsweise festgestellt worden, dass nicht alle Störkanten aussortiert werden. Abbildung 4.7 zeigt beispielsweise ein Untersuchungsfenster des Algorithmus, dessen Kante nicht ausgeschlossen und somit als Bogenkante identifiziert wurde.

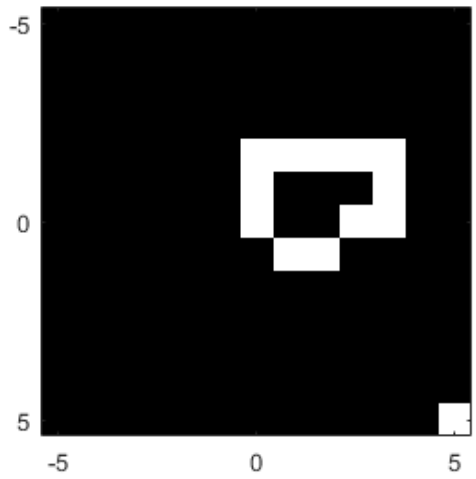


Abbildung 4.7: Störkante im Kantenbild des Untersuchungsfensters

Die Tatsache, dass auch solche Störungen den Algorithmus-Ausgaberaum beeinflussen, sorgt für verringerte Erkennungsraten, da mehr Linien im Ausgaberaum bedeuten, dass dort zufällig Maxima entstehen können. Das Herausfiltern solcher Störungen ist aber vermutlich keine Aufgabe, die schnell und ohne viel Rechenaufwand gelöst werden kann.

Es gibt weitere Möglichkeiten zum Verbessern des Algorithmus, wie zB. ein verändertes Schreiben der Linien im Algorithmus-Ausgaberaum. Im ursprünglichen Entwurf werden Linien von Bogen-Kantenstücken in Richtung des zugehörigen hypothetischen

Kreismittelpunkts komplett bis zum Bildrand hin in den Ausgaberaum gezeichnet. Da aber bekannt ist, dass der zu detektierende Ball nur eine gewisse Maximalgröße in Pixeln auf dem Bild erreichen kann, könnte auch die Maximallänge der Ausgabelinien beschränkt werden. Dies würde Falschdetektionen verringern, da sich weniger Linien-Überschneidungen an falschen Positionen ergeben und der tatsächliche Kreismittelpunkt eher als Maximum im Ausgaberaum auftritt. Zudem würde die zum Schreiben der Linien benötigte Rechenzeit verringert werden. Eine weitere Verbesserung wäre das Suchen nach dem des Spielball nur unter dem Horizont, dessen Position sich auf dem Roboter mithilfe der Gelenkwinkel- und Lagesensorinformationen ermitteln lässt. Außerdem könnte, speziell um farbige Bälle zu erkennen, einer der Bild-Farbkanäle zum Erstellen eines besseren Kantenbilds verwendet werden.

Kapitel 5

Fazit und Ausblick

Das Ziel dieser Arbeit war es, einen Algorithmus zwecks Echtzeiterkennung von Bällen durch ihre Form auf der NAO-Robotikplattform zu implementieren und die Lösung auf ihre Nutzbarkeit hin zu untersuchen. Die Versuchsergebnisse haben gezeigt, dass es zwar möglich ist, den gewählten Algorithmus in Echtzeit zu verwenden, Bälle im Roboterfußball-Kontext jedoch nicht zuverlässig erkannt werden. Die Probleme des Algorithmus könnten durch einige zuvor beschriebene Schritte zum Teil gelöst werden. Doch selbst von weiter angepassten Versionen des Algorithmus wäre es nicht zu erwarten, dass der realistische Spielball erkannt werden könnte. Die Methode, lediglich ein Kantenbild als Grundlage hierfür zu Nutzen, scheint nicht ausreichend zu sein.

In Zukunft sollten also andere Methoden verwendet werden, um verschiedenartige Bälle detektieren zu können. Bei der Erstellung dieser Arbeit entstand der Eindruck, dass ein Ball mit Muster ein zu komplexes visuelles Objekt ist, um es mit simplen Methoden detektieren zu können. Vielversprechend scheint zur Lösung dieses Problems ein Ansatz, bei welchem maschinelles Offline-Lernen zum Trainieren eines Echtzeit-Klassifizierers angewendet wird. Ein derartiges ein Verfahren wurde 2013 von einem Universitätsteam der *RoboCup Middle Size League* vorgestellt und sehr erfolgreich im Spiel verwendet [Zha+]. Solch eine Herangehensweise, bei welcher allgemeine, nicht näher spezifizierte Objekte anhand markanter visueller Merkmale erkannt werden können, scheint auch für die Erkennung von realistischen Spielbällen geeignet zu sein. Aus diesem Grunde sollten zur weiteren Entwicklung einer Balldetektion ebenfalls solche Lösungsansätze in Erwägung gezogen werden.

Literatur

[Ash+14] Jayen Ashar u. a. *RoboCup SPL 2014 Champion Team Paper*. Sydney, Australien: School of Computer Science und Engineering, University of New South Wales, 2014. URL: http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20141221-SPL2014ChampionTeamPaper.pdf.

[Aur11] Prof. Dr. Volker Aurich. *Bildverarbeitung - Skript zur Vorlesung*. Heinrich-Heine-Universität Düsseldorf, 2011. URL: http://www.acs.uni-duesseldorf.de/~aurich/Skripte/Bildverarbeitung_1.pdf.

[Bay75] Bryce E. Bayer. "Color imaging array". Pat. US3971065 A. 5. März 1975.

[Bey12] Jürgen Beyerer. *Automatische Sichtprüfung Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung*. Hrsg. von Christian Frese. Berlin, Heidelberg: Springer, 2012, S. 237–238.

[Bon15] Universität Bonn. *RoboCup in Deutschland: News*. Okt. 2015. URL: http://www.ais.uni-bonn.de/robocup.de/news.html.

[Can86] John Canny. *A Computational Approach to Edge Detection*. Techn. Ber. 1986.

[Com15] RoboCup Technical Committee. *RoboCup Standard Platform League (NAO) Rule Book*. 2015.

[CS15] Carnegie Mellon University School of Computer Science. *CMDragons: RoboCup Small-Size Robot Soccer Team*. Okt. 2015. URL: http://www.cs.cmu.edu/~robosoccer/small/.

[DH75] Richard O. Duda und Peter E. Hart. "Use of the Hough Transform to Detect Lines and Curves in Pictures". In: *Communications of ACM 15. Graphics and Image Processing* 15.1 (1975). Hrsg. von W. Newman, S. 11–15.

[Dud15] Duden. *Definition - Roboter*. Sep. 2015. URL: http://www.duden.de/node/665307/revisions/1334690/view.

[Hei15] Heise.de. *Sony schläfert Aibo ein*. Okt. 2015. URL: http://www.heise.de/newsticker/meldung/Sony-schlaefert-Aibo-ein-169676.html.

[HUL15] HULKs. *Hamburg Ultra Legendary Kickers*. Sep. 2015. URL: https://www.hulks.de.

[Its15] Itseez. *OpenCV*. Okt. 2015. URL: http://opencv.org/.

[Jäh05] Bernd Jähne. *Digital image processing*. 6., rev. and extended ed. Berlin [u.a.]: Springer, 2005.

[Kit+95] Hiroaki Kitano u. a. *RoboCup: The Robot World Cup Initiative*. 1995.

[Lot15] Pascal Loth. "Implementierung und Evaluation einer robusten Echtzeitkantendetektion auf dem humanoiden NAO-Robotiksystem". Technische Universität Hamburg-Harburg. Hamburg, 2015.

[Ore15] Viktor Orekhov. *Humanoid kid size league match at RoboCup 2011*. Okt. 2015. URL: http://www.robocup2013.org/pers/humanoid-kid-size-league-match-at-robocup-2011-photo-viktor-orekhov/.

[PBK14] Finn Poppinga, Florian Bergmann und Stefan Kaufmann. *HULKs - Team Research Report 2014*. Hamburg: Technische Universität Hamburg-Harburg, 2014.

[Poy03a] Charles Poynton. *Digital video and HDTV algorithms and interfaces*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. ISBN: 1558607927.

[Poy03b] Charles Poynton. YUV and luminance considered harmful. In: *Digital video and HDTV algorithms and interfaces*. 2003.

[Rei11] Thomas Reinhardt. "Kalibrierungsfreie Bilderverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball". Magisterarb. Hochschule für Technik, Wirtschaft und Kultur Leipzig, 2011.

[Rob15a] RoboEarth. *RoboCup German Open 2010*. Okt. 2015. URL: http://roboearth.org/robocup-german-open-2010/.

[Rob15b] Aldebaran Robotics. *C++ SDK*. Okt. 2015. URL: http://doc.aldebaran.com/2-1/dev/cpp/index.html.

[Rob15c] Aldebaran Robotics. *NAO - Construction*. Okt. 2015. URL: http://doc.aldebaran.com/2-1/family/robots/dimensions_robot.html.

[Rob15d] Aldebaran Robotics. *NAO - Video Camera*. Okt. 2015. URL: http://doc.aldebaran.com/2-1/family/robots/video_robot.html.

[Rob15e] Aldebaran Robotics. *NAO Datasheet*. Sep. 2015. URL: https://www.aldebaran.com/sites/aldebaran/files/nao_datasheet.pdf.

[Rob15f] Aldebaran Robotics. *Who is NAO?* Sep. 2015. URL: https://www.aldebaran.com/en/humanoid-robot/nao-robot.

[Röf+13] Thomas Röfer u. a. *B-Human Team Report and Code Release 2013*. Bremen, 2013. URL: http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf.

[SPV05] D. Scaramuzza, S. Pagnottelli und P. Valigi. "Ball Detection and Predictive Ball Following Based on a Stereoscopic Vision System". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE. Barcelona, Spanien, 2005, S. 1573–1578.

[UB15] Technologie-Zentrum Informatik und Informationstechnik der Universität Bremen. *Call for Applications for Participation, RoboCup 2015 Standard Platform League*. Okt. 2015. URL: http://www.informatik.uni-bremen.de/spl/bin/view/Website/Call2015.

[Zha+] Hui Zhang u. a. "A Novel Generic Ball Recognition Algorithm Based on Omnidirectional Vision for Soccer Robots". In: Bd. 10. 388. International Journal of Advanced Robotic Systems.

Anhang A

Inhalt der DVD

Ordner	Inhalt
/Bachelorarbeit	enthält dieses Dokument als PDF und die zugehörigen Quelldateien
/Literatur	enthält die referenzierten Paper
/Bildsequenzen	enthält verwendete Testsequenzen sowie Algorithmus-Ausgaben
/Quelltext	enthält Source- und Header-Dateien der Implementierung