Technische Universität Hamburg-Harburg
Vision Systems

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

Implementierung und Evaluation einer robusten Echtzeitkantendetektion auf dem humanoiden NAO-Robotiksystem

Bachelorarbeit

Pascal Loth

19. Oktober 2015

Eidesstattliche Erklärung

Ich, Pascal Loth, geboren am 14.09.1991 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

Nicht nur in der Industrie, wenn das autonom fahrende Auto Straßenschilder und Fahrbahnmarkierungen zu erkennen hat, sondern auch in der Medizin zur 3D-Rekonstruktion innerer Organe, ist es unabdingbar geworden, Informationen aus Kamerabildern zu extrahieren.
Die Schwierigkeit liegt dabei in den sich ständig wechselnden Umwelteinflüssen. Aufgaben, die für den Menschen mithilfe seiner kognitiven Fähigkeiten leicht erscheinen, sind technisch nur schwer zu realisieren. Sich ändernde Lichtverhältnisse beeinflussen Kamerasysteme sehr stark. Eine teils beschattete einfarbige Fläche erscheint für den Menschen homogen, doch lässt der Helligkeitsunterschied einen Roboter eine separierte Fläche wahrnehmen. Nötig werden Algorithmen, die die Informationsverarbeitung des Menschen nachahmen.
Jahrelang bestand ein Großteil der Forschung für künstliche Intelligenz in der Programmierung von Schachcomputern, eine für ein Computersystem ziemlich ideale Aufgabe, da sich aktuelle und zukünftige Spielzüge mathematisch bewerten lassen, um in jeglicher Situation bestmöglich zu reagieren. Als im Jahre 1997 der IBM Schachcomputer Deep Blue den amtierenden Weltmeister Garri Kasparow in einem Schachturnier schlug, fand auch der erste RoboCup statt.
Der RoboCup ist ein jährlich stattfindender internationaler Wettbewerb, an dem Wissenschaftler und Studenten der ganzen Welt teilnehmen, um die Forschung an künstlicher Intelligenz und Robotik voran zu bringen. Er ist zusammengesetzt aus unterschiedlichen Ligen mit jeweils eigenen Entwicklungsschwerpunkten. Neben Rettungsrobotern für Katastrophensituationen in der *Rescue League* und autonomen Robotern für das eigene Heim zur Mensch-Maschinen-Interaktion, kommt in der *Standard Platform League (SPL)* ein humanoider Roboter namens NAO zum Einsatz.
In der SPL wird basierend auf gleicher Hardwaregrundlage im Roboterfußball gegeneinander angetreten. Eine Herausforderung, die sich auch das im Jahr 2013 gegründete Team *Hamburg Ultra Legendary Kickers (HULKs)* [HUL15] angenommen hat.

Zielsetzung Die Restriktion, keine modifizierte oder zusätzliche Hardware zu verwenden, fordert ein Zurechtkommen allein mit den dem NAO gegebenen Mitteln. Die digitale Bildverarbeitung ist die einzige Möglichkeit für den Roboter seine Umwelt umfassend wahrzunehmen, welche ihr also einen entsprechend großen Stellenwert verleiht. Durch sich jährlich verschärfende Regeln in der *Standard Platform League* verschwinden eindeutige Erkennungsmerkmale der Spielumgebung. Neben dem Austausch der markanten gelben durch weiße Tore, wird in der nächsten Saison der bisher benutzte rote Spielball durch einen nicht farbcodierten ersetzt.
Detektionsalgorithmen, die also allein auf der Erkennung von Farben basieren, sind in der Zukunft somit keine Option. Formen von Objekten sind zusätzliche Merkmale, die das maschinelle Sehen unterstützen.
Das Fußballspiel erfordert zugleich Verfahren, die in Echtzeit Kamerabilder verarbeiten können, was durch begrenzte Rechenleistung nur mit drastischer Reduktion der Informationsmenge zu bewältigen ist. Ein bewährter Schritt ist die Kantenerkennung, um inhaltlich ähnliche Regionen zusammenzufassen.
Diese Bachelorarbeit nimmt sich die Evaluation eines Kantenerkennungsverfahrens und dessen echtzeitfähige Implementierung auf dem NAO-Robotiksystem zum Thema

Kapitel 2

Grundlagen

Der NAO ist ein 58cm großer humanoider Roboter der französischen Firma *Aldebaran Robotics*, den sie im Jahr 2006 zum ersten Mal vorstellte.
Entwickelt wurde er als interaktiver täglicher Begleiter [Rob15d]. Er fand aber auch auf der ganzen Welt Anwendung in Schulen und Universitäten, denn die mitgelieferte Software ermöglicht bereits Kindern ihre ersten Schritte in der Programmierung zu bewältigen.
Anwendung fand der NAO auch in der *Standard Platform League* des *RoboCups*, in der bis dato noch der vierbeinige hundähnliche *Sony Aibo* Roboter verwendet wurde. 2007 fanden die ersten Wettkämpfe auf Basis des NAOs statt.

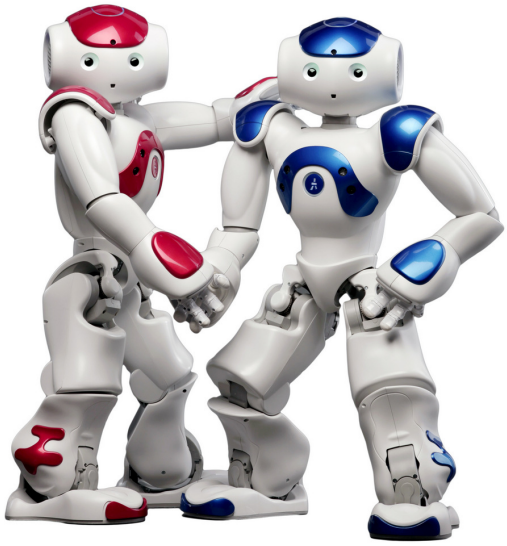


Abbildung 2.1: Zwei NAO-Roboter der Firma *Aldebaran Robotics* [Rob15a]

Über die Zeit entwickelte *Aldebaran Robotics* ihr Robotersystem weiter und veröffent-

lichte stetig verbesserte Versionen. Das aktuelle Exemplar wiegt etwa 4,3kg, besitzt 25 Freiheitsgrade und ist mit einem Intel Atom Z530 Prozessor (1,6GHz) und 1GB Arbeitsspeicher ausgestattet. Auf diesem läuft ein auf der Gentoo Distribution basierendes Linux Betriebssystem. Der NAO verfügt über zwei Kameras, auf die in Abschnitt 2.1 genauer eingegangen wird. Im Kopf sind Infrarotmodule, Lautsprecher und 4 gerichtete Mikrofone eingebaut, die Geräuschquellenlokalisierung möglich machen. Hinzu kommen im restlichen Körper verbaute Drucksensoren an den Händen und Füßen, Ultraschall- und Inertialsensoren. Über Ethernet und Wireless LAN lässt sich mit dem System kommunizieren.

2.1 Kameras

In dem NAO-Roboter sind zwei CMOS Bildsensoren des Typs MT9M114 verbaut, die nicht wie zu erwarten in den Augen sitzen, sondern in der Stirn und in der unteren Gesichtshälfte platziert wurden. Diese Anordnung macht es möglich, dass der NAO das Geschehen in der Ferne und direkt vor seinen Füßen zur gleichen Zeit betrachten kann. In Abbildung 2.2 ist die nur leichte Überschneidung der Sichtfelder verdeutlicht. Die gewählte Anordnung ermöglicht ein Gesamtsichtfeld von 72,6°, aber unterbindet die Aufnahme dreidimensionaler Bilder.

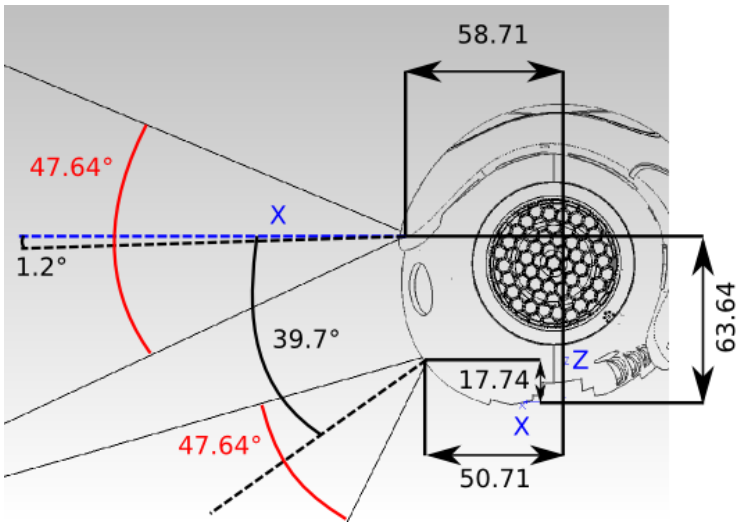


Abbildung 2.2: Öffnungswinkel und Positionen der Kameras im Kopf [Rob15b]

Die im NAO verbauten CMOS Bildsensoren sind aktive Pixelsensoren (APS). Die gemessene Spannung, des auf den Sensor treffenden Lichtes, ist von der Intensität abhängig, welche daraufhin von einem Analog-Digital-Umsetzer diskretisiert wird.
Ein über dem Bildsensor liegender Farbfilter sorgt für das Passieren auftreffender Wellenlängen auf zugehörige Detektorflächen. Häufig wird die Bayer-Matrix als Filter verwendet. Diese sorgt für eine doppelt so häufige Abtastung der Grüntöne gegenüber der

beiden anderen Grundfarben, denn das menschliche Auge ist besonders empfindlich für Grün und damit einschließlich auch für die darin enthaltenen Grautöne, welche für das Helligkeitsempfinden zuständig sind [Poy12a]. Diese erhöhte Abtastung, wie in Abbildung 2.3 dargestellt ist, und anschließender Mittelwertbildung sorgt für eine bessere Kontrast- und Schärfewahrnehmung.
Die RGB Werte jedes Pixels werden daraufhin noch innerhalb der Kamera in das YCbCr-Farbmodell, auf welches in Abschnitt 2.2.1 genauer eingegangen wird, umgesetzt. Eine Umrechnung ist wie folgt definiert:

$$\begin{bmatrix} Y \\ CB \\ Cr \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \cdot \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ -0,168736 & -0,331264 & 0,5 \\ 0,5 & -0,418688 & -0,081312 \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{2.1}$$

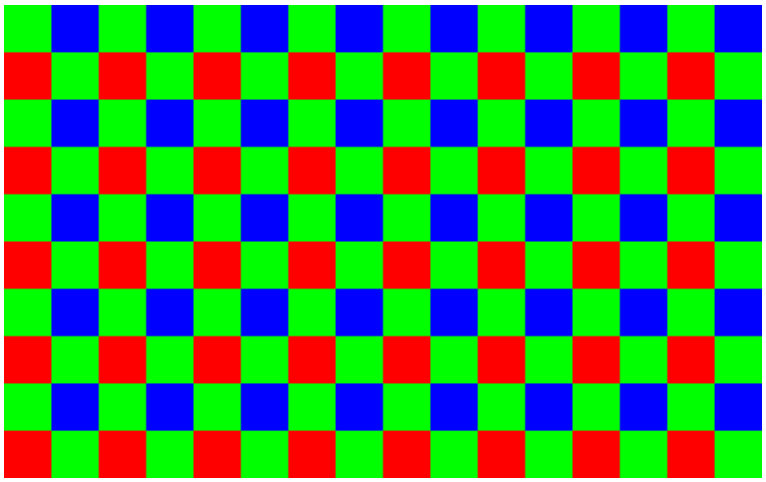


Abbildung 2.3: Bayer-Matrix

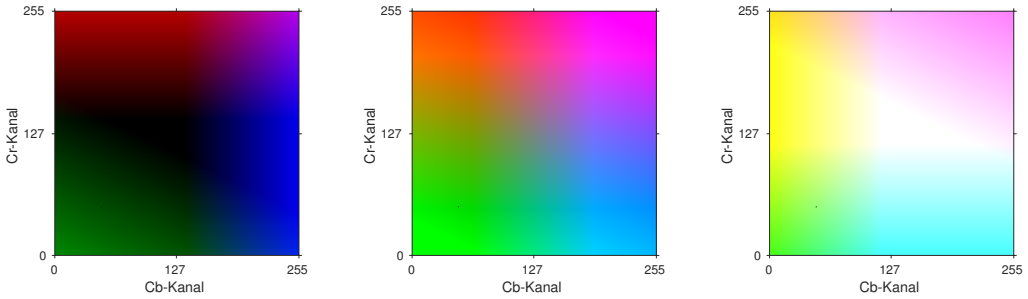
Ein großer Nachteil der Bilder, die von CMOS Sensoren stammen, ist der *Electronic-rolling-shutter"*-Effekt (ERS). Es kommt dabei meist zu Verzerrungen, wenn sich die Kamera oder der Bildinhalt bei der Aufnahme stark bewegt, da die Bildzeilen nacheinander aufgenommen werden [NFM07, S. 402].

2.2 Bilddarstellung und Verarbeitung

2.2.1 Farbraum

Der NAO speichert die Rohdaten seiner Kamera im *YUV422* Format ab [Rob15c]. Y'UV besteht aus zwei Komponenten. Zum einen ist im Y-Kanal die Helligkeitsinformation bzw. Luminanz (luma) eines Pixels gespeichert, zum anderen kam in der Zeit

des Farbfernsehens die UV-Komponente hinzu. Sie beschreibt die Farbigkeit oder Chrominanz (chroma) eines Pixels, um genauer zu sein die Abweichung von einem neutralen Grauton. U ist dabei der Verlauf von Gelb nach Blau und V von Türkis nach Rot.
Der Ausdruck *YUV422* steht allerdings in der heutigen Computerindustrie für YCbCr 4:2:2, welches eine skalierte und normierte digitalisierte Version des Y'UV Farbmodells ist [Poy12c]. Abbildung 2.4 zeigt die durch CbCr aufgespannten Farbebenen bei drei exemplarischen Y-Werten.



(a) Y= 0 (b) Y= 127 (c) Y= 255

Abbildung 2.4: Y'CbCr Farbmodell bei exemplarischen Y-Werten

Beide Farbmodelle bieten wesentliche Vorteile gegenüber einem additiven Farbraum wie RGB, in dem sich jede Farbe durch Kombination der drei Grundfarben darstellen lässt. Denn in der Natur sorgen Schattierungen nur für eine grundlegende Änderung der Helligkeitswerte und nicht für eine wesentliche Veränderung der Chrominanz. Die Farben haben somit für die Wahrnehmung eine nicht so hohe Bedeutung wie die Helligkeitsinformation.
Dies resultiert in der Möglichkeit, eine Unterabtastung der UV Komponente vorzunehmen, um eine starke Informationsreduktion zu erreichen [Poy12a]. Eine solche Komprimierung ist für das menschliche Auge nahezu unsichtbar. Auch die Bilder, die die Kameras des NAO-Roboters liefern, sind bereits unterabgetastet.
Der Zusatz 4:2:2 steht für die Schema des stattgefundenen Sub-Samplings. Eine Abtastung J:a:b beschreibt wie häufig in einem Jx2 Gitter die Cb und Cr Kanäle abgetastet werden. Im häufigsten Fall handelt es sich um ein Fenster mit einer Breite von 4 Pixeln (J=4). Das a steht für das Verhältnis der Abtasthäufigkeit zu J in der ersten Reihe. Respektive b für das der Zweiten [Poy12b, S. 124]. In Abbildung 2.5 sind häufig verwendete Abtastungen exemplarisch dargestellt.
Es ist deutlich zu erkennen, dass der Helligkeitskanal unberührt bleibt. In R'G'B' 4:4:4 und Y'CbCr 4:4:4 finden keine Unterabtastungen statt. In 4:2:2 und 4:2:0 wird die erste Reihe im Cb- und Cr-Kanal nur halb so oft abgetastet, wie der Y-Kanal. Die zweite Reihe der Chrominanz wird sogar ganz in 4:2:0 unterbunden. Hingegen existiert in 4:1:1 nur noch 25% der Farbinformation.

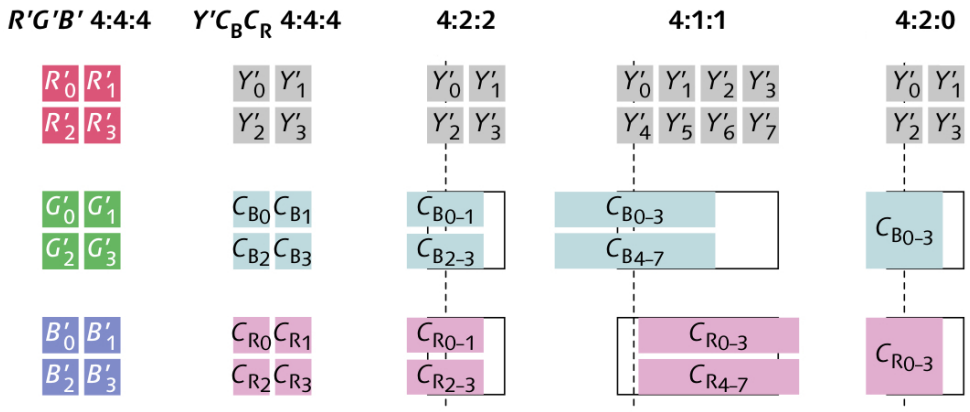


Abbildung 2.5: Gebräuchliche Farbunterabtastungen in Anlehnung an [Poy12b, S. 124]

2.2.2 Faltung

Eine häufig angewandte Methode in der digitalen Bildverarbeitung ist die Faltung. Sie beschreibt eine Überlagerung zweier Funktionen, um ein gewichtetes Bild zu erzeugen. Um ein Bild mit einer Gewichtungsfunktion zu falten, ist von dieser zuerst eine Matrix zu bestimmen. Diese Matrix wird auch Kern genannt und besteht aus äquidistanten Abtastungen der Gewichtungsfunktion, weswegen auch von einer diskreten Faltung zu sprechen ist. Ein Bild ist durch vorhandene Pixel bereits implizit diskret.
Das Zentrum der Filtermatrix wird üblicherweise *hot spot* genannt und liegt zumeist bei $H(0,0)$. Die Matrix besitzt also ein eigenes Koordinatensystem.
Der *hot spot* wird über jeden Pixel des Originalbildes I gelegt und wie in Abbildung 2.6 dargestellt, werden die Filterkoeffizienten $H(i,j)$ mit den darunter liegenden Pixeln multipliziert und anschließend aufsummiert [Bur06, S. 92].
Die Faltung eines Bild mit einem 3×3 Kern ist mathematisch wie folgt definiert:

$$I'(u,v) = \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u+i, v+j) \cdot H(i,j) \tag{2.2}$$

Genau betrachtet sind die Randbereiche des Bildes problematisch, da es keine Nachbarpixel gibt, die mit in die Berechnung einfließen können. Es gibt verschiedene Möglichkeiten, dies zu umgehen.
Unter der Annahme, dass sich nicht existierende Pixel außerhalb des Originalbildes nur wenig ändern würden, kann auf den nächstliegenden vorhandenen Bildpunkt zugegriffen werden. Dargestellt ist diese Vorgehensweise in Abbildung 2.7b.
Andererseits ließe sich das Bild wie in Abbildung 2.7c zyklisch fortsetzen. Doch diese Methode ist nur in wenigen Anwendungsfällen von Nutzen. Zumeist wird die Faltung auf Bildern zur Glättung ausgeführt.

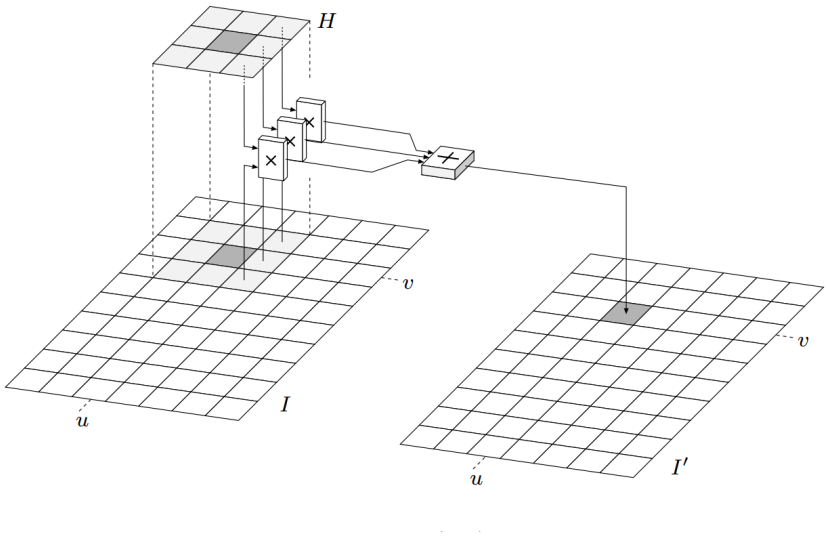
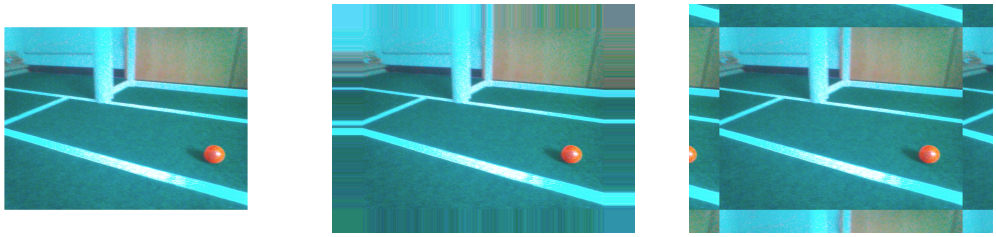


Abbildung 2.6: Darstellung der Faltungsoperation [Bur06, S. 92])



(a) Originalbild (b) Erweitert (c) Wiederholung

Abbildung 2.7: Bildrandproblematik bei Faltung

Dazu sei folgender Kern gegeben:

$$H = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.3}$$

Der Kern H sorgt für eine gleich gewichtete Einbeziehung aller umliegenden Bildpunkte von I. Die Aufsummierung aller Pixelwerte mit anschließender Division durch den Normierungsfaktor N, in diesem Fall 9, resultiert in einem Durchschnittspixel, der an entsprechende Stelle in I' gespeichert wird.
Der Grad der Glättung geht mit der Größe des Kerns einher. Dadurch steigt nicht implizit der Nutzen, da eine Vergrößerung immer in einem deutlich höherem Berechnungsaufwand resultiert.

2.3 Kantendetektion mit Canny-Algorithmus

Mit dem Canny-Algorithmus lassen sich flächige Bereiche auf einem digitalem Bild voneinander trennen, sofern sie sich anhand von Helligkeitswerten hinreichend unterscheiden lassen. Ursprung hat das Verfahren in dem 1986 publizierten Paper *A Computational Approach to Edge Detection* von John Canny, der dem Algorithmus auch seinen Namen verleiht [Can86].
Das Verfahren ist zur Verwendung auf einem Graustufenbild entworfen worden, welches bei der Verwendung des Y'UV Farbmodells nicht mehr berechnet werden muss, da es durch den Y-Kanal schon gegeben ist. Der U- und V-Kanal bleiben somit unberührt.
Der Algorithmus wurde entworfen ein ungestörtes Kantenbild, mit minimaler Distanz zwischen der detektierten und der echten Kante, zu liefern. Dabei sollen auf eine Kante nicht mehrere Antworten folgen [NA12].
Mithilfe verschiedener Faltungsoperationen wird versucht gegen Bildstörungen anzugehen. Kanten werden anhand partieller Ableitungen an Stellen großer Helligkeitsschwankungen erkannt und anschließend auf eine Breite von einem Pixel reduziert. Letztlich angewandte Schwellwerte bei einer Kantenverfolgung verhindern unbedeutende Resultate. Folgende Abschnitte beschreiben die im Paper behandelten Verarbeitungsschritte im Detail.

2.3.1 Gaußscher Weichzeichner

Digitale Bilder sind häufig durch ein Rauschen gestört, welches sich durch Informationsverlust äußert. Solche Bildstörungen können viele Ursachen haben, auf die in dieser Arbeit nicht weiter eingegangen werden.
Es gilt vorhandenes Rauschen zu unterdrücken, damit ein zufällig hell auftretender Pixel auf einer dunklen Fläche nicht als Kante detektiert wird. Dazu wird das Bild geglättet. Es ist zu bedenken, dass dadurch auch immer real existierende Kanten weich gezeichnet werden. In Abschnitt 2.2.2 wurde bereits kurz ein einfaches Filter zur Glättung vorgestellt.
Der Canny-Algorithmus sieht es vor, einen Kern ähnlich der zweidimensionalen Gauß-Glocke, dargestellt in Abbildung 2.8, zu verwenden, welche umliegende Pixel nicht im gleichen Maße einbezieht [Nis+11, S. 178]. Die Gewichtung ist abhängig von dem Abstand zum Mittelpunkt der Funktion.

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \mathrm{e}^{-\frac{x^2}{2\sigma^2}} \tag{2.4}$$

Gleichung 2.4 zeigt die Impulsantwort für den eindimensionalen Fall, welche durch Multiplikation mit einer weiteren Normalverteilung in orthogonale Richtung auf zwei

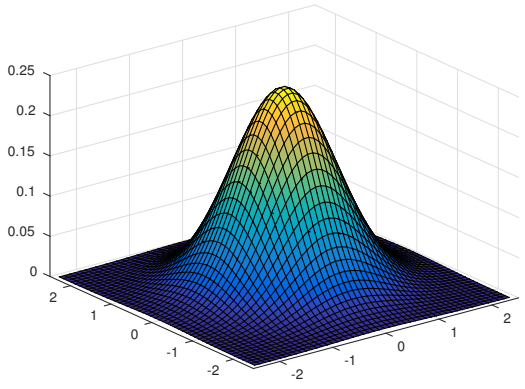


Abbildung 2.8: Gauß-Glocke ($\sigma = 0.8$; $\mu = 0$)

Dimensionen erweitert wird. Die Funktion der Gauß-Glocke ist in Gleichung 2.5 dargestellt.
Zunächst ist die Größe des Filters festzulegen, welche mit der Stärke der Glättung einhergeht. Bedingt durch die Anzahl der Matrixelemente ist die Dimension beschränkt auf $(2k+1) \times (2k+1)$ mit $k \in \mathbb{N}^*$.

$$g(x) \cdot g(y) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \mathrm{e}^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma} \cdot \mathrm{e}^{-\frac{y^2}{2\sigma^2}} = \frac{1}{2\pi\sigma^2} \cdot \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} = g(x,y) \tag{2.5}$$

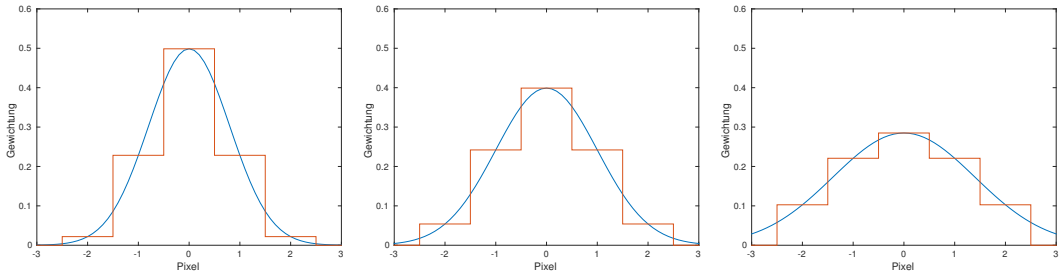
Mit einer leichten Anpassung von $g(x,y)$ liefert Gleichung 2.6 die Elemente des Kerns [Jia05, S. 42]. Zudem ist eine Standardabweichung σ zu wählen, die den Kern bestmöglich die Gauß-Glocke repräsentieren lässt. Die Ausdehnung der Verteilung ist, wie in Abbildung 2.9 dargestellt, abhängig von der Standardabweichung.

$$H_{i,j} = \frac{1}{2\pi\sigma^2} \cdot \mathrm{e}^{-\frac{(i-k)^2+(j-k)^2}{2\sigma^2}} \tag{2.6}$$

Ein häufig verwendeter Gauß-Filterkern ist in Gleichung 2.7 dargestellt. Wie auch das Integral über die Normalverteilung, muss die Summe der Matrixelemente 1 ergeben.

$$H_{5\times5} = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{2.7}$$

Abbildung 2.10 zeigt $H_{5\times5}$ angewendet auf einen Kamerabildausschnitt des NAO-Roboters.



(a) $\sigma = 0.8$ (b) $\sigma = 1.0$ (c) $\sigma = 1.4$

Abbildung 2.9: Ausdehnung der Normalverteilung

(a) Originalbild (b) Geglättetes Bild

Abbildung 2.10: Anwendung einer 5×5 Gauß-Filtermatrix

2.3.2 Kantendetektion

Der Canny-Algorithmus erkennt Kanten, die eine sprunghafte Helligkeitsänderung aufzeigen, über die partiellen Ableitungen in horizontaler und vertikaler Richtung. Mit dieser Information lässt sich zusätzlich die Kantenrichtung berechnen. Der euklidische Betrag beider partiellen Ableitungen ergibt die jeweiligen absoluten Kantenstärken an jedem Pixel.

2.3.2.1 Partielle Ableitung

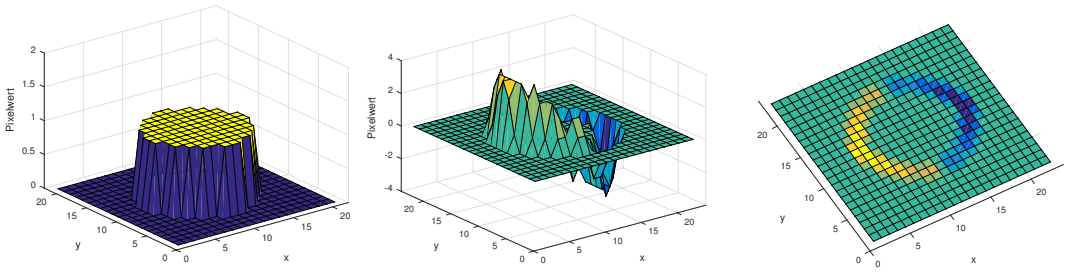
Die partielle Ableitung wird über eine Finite-Differenzen-Methode der Form $f(x+b) - f(x+a)$ angenähert. Zum Einsatz kommt dafür der Sobel-Feldman-Operator, der die Ableitung mithilfe einer Faltungsoperation berechnet [Gri13, S. 126]. Zur Bestimmung der Ableitung in horizontaler Richtung ist der S_x Operator, welcher in Gleichung 2.8 beschrieben ist, und in vertikaler Richtung der um 90° gedrehte Operator

S_y, siehe Gleichung 2.9, zu verwenden. Die Faltungen mit den jeweiligen Operatoren erzeugen zwei neue Bilder G_x und G_y.

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.8}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.9}$$

Zudem führt der Sobel-Operator auch eine Glättung des Bildes aus, die ähnlich der Glättung des Gaußschen Weichzeichners ist.
Abbildung 2.11a zeigt ein manuell erzeugtes Binärbild, welches einen Ball darstellen könnte. Wird dies mit dem Sobel-Operator in x-Richtung gefaltet, entsteht ein Gradientenbild 2.11b, in dem deutlich zu erkennen ist, dass auch nur ein sprunghafter Unterschied entlang der x-Achse eine Antwort erzeugt. Die Faltung mit S_y würde ein ähnliches und um 90° gedrehtes Bild erzeugen.



(a) Binärbild eines Balles (b) G_x (c) Draufsicht auf G_x

Abbildung 2.11: Faltung mit dem Sobel-Operator

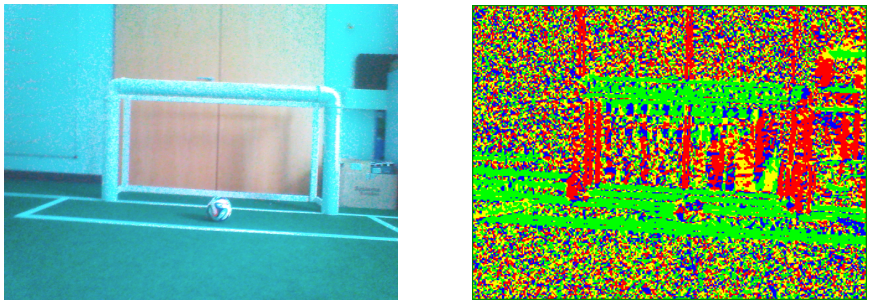
2.3.2.2 Kantenrichtung

Für spätere Schritte des Canny-Algorithmus ist die grobe Richtungsinformation einer Kante notwendig. Aus den beiden Bildern G_x und G_y der partiellen Ableitungen lässt sich über die Umkehrfunktion des Tangens die Gradientenrichtung bestimmen [Nis+11, S. 171]. Ein Gradient zeigt in die Richtung der größten Steigung, somit steht er orthogonal zu der Kante. Doch ist das Abbildungsintervall des Arkustangens auf $[-\frac{\pi}{2}, \frac{\pi}{2}]$ beschränkt.
Abhilfe schafft der atan2, welcher wie in Gleichung 2.10 definiert ist und durch eine

leichte Anpassung auf alle vier Quadranten des kartesischen Koordinatensystems abbildet $[-\pi, \pi]$ [Mat15].

$$\text{atan2}(y,x) := \begin{cases} \arctan\frac{y}{x} & \text{für } x > 0 \\ \arctan\frac{y}{x} + \pi & \text{für } x < 0, y \geq 0 \\ \arctan\frac{y}{x} - \pi & \text{für } x < 0, y < 0 \\ +\frac{\pi}{2} & \text{für } x = 0, y > 0 \\ -\frac{\pi}{2} & \text{für } x = 0, y < 0 \\ 0 & \text{für } x = 0, y = 0 \end{cases} \tag{2.10}$$

Mit $\text{atan2}(G_y, G_x) = \theta$ lässt sich an jeder Position aus den entsprechenden Pixelwerten der partiellen Ableitungen eine Richtung bestimmen. Da aber jeder Pixel nur acht Nachbarn hat, wird θ auf $0°, 45°, 90°$ oder $135°$ gerundet.
In Abbildung 2.12 ist ein Kamerabild mit den dazugehörigen gerundeten Gradientenrichtungen dargestellt.
Horizontale Kanten wurden Grün und vertikale Rot eingefärbt. Hingegen sind Kanten im 35°-Winkel Gelb und bei einem 135°-Winkel Blau.



(a) Originalbild (b) Gradientenrichtungen

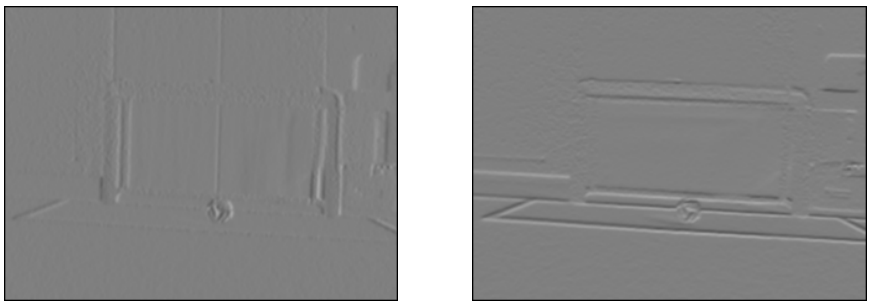
Abbildung 2.12: Kamerabild mit den dazugehörigen Gradientenrichtungen

2.3.2.3 Absolute Kantenstärke

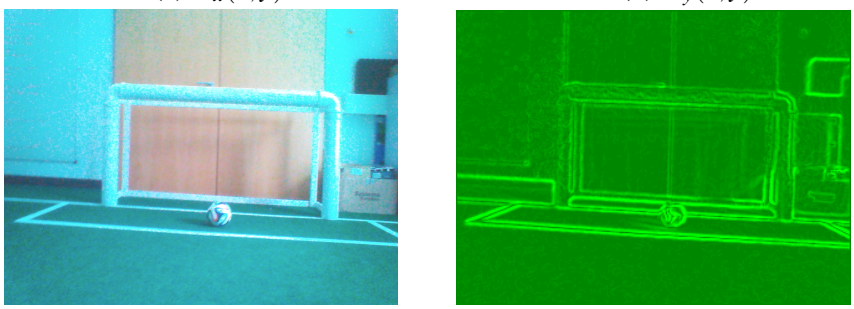
Der euklidische Betrag der vertikalen und horizontalen partiellen Ableitung liefert die absolute Kantenstärke.

$$G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{2.11}$$

Abbildung 2.13 zeigt die partiellen Ableitungen und das durch Gleichung 2.11 entstehende Kantenbild.
Durch die Verarbeitung im Y'UV-Farbmodell und nicht vorhandene Informationen im U- und V-Kanal ist das Bild der absoluten Kantenstärke grün.



(a) $G_x(x,y)$ (b) $G_y(x,y)$



(c) Originalbild (d) $G(x,y)$

Abbildung 2.13: Partielle Ableitungen und resultierende absolute Kantenstärke

2.3.3 Non-maximum suppression

Die Problematik des bisherigen Kantenbildes 2.13d ist ersichtlich. Eine Kante resultiert nicht in einer deutlich erkennbaren präzisen Linie, die Bereiche voneinander trennt. Dies kann unter anderem erst durch die *Non-maximum suppression (NMS)* erreicht werden. Dabei wird versucht die Kantenlinien auf eine maximale Breite von einem Pixel zu reduzieren.
Die NMS untersucht hierfür jeden Pixel in $G(x,y)$ und entscheidet anhand eines Vergleichs mit gewählten Nachbarpixeln, welche zu unterdrücken sind [Nis+11, S. 178]. Wie in Abschnitt 2.3.2.2 bereits erwähnt, steht die Gradientenrichtung orthogonal zu der Kantenrichtung, somit lassen sich die beiden Nachbarpixel an jedem Bildpunkt links und rechts einer Kante untersuchen. Ist einer der beiden Werte größer, wird der aktuelle Pixel zu Null gesetzt.
Dies resultiert in einem Kantenbild 2.14, welches klare Trennungen zwischen Bereichen aufzeigt.

2.3.4 Hysterese

Das Ergebnis der *Non-maximum supression* lässt sich weiter verbessern, da noch viel Unbedeutendes in dem Kantenbild zu sehen ist, welches durch Bildrauschen entstanden ist. Diese Kanten sind nicht besonders stark gekennzeichnet und würden sich durch

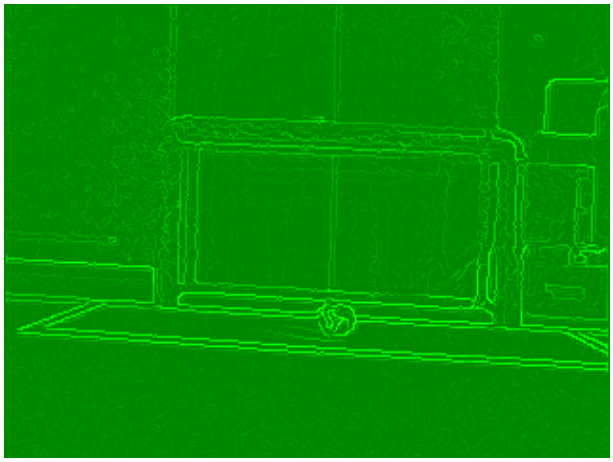


Abbildung 2.14: Non-maximum supression

Anwendung eines Schwellenwertes filtern lassen, doch möchte man verhindern, dass relevante Kanten aufbrechen können.
Deswegen kommt ein Hysterese genanntes Verfahren zum Einsatz, bei dem zwei Schwellwerte $T_1 < T_2$ zu definieren sind. Daraufhin wird das Bild nach einem Helligkeitswert durchsucht, der größer oder gleich T_2 ist. Bei diesen wird von einer starken Kante ausgegangen, die in beide Richtungen zu verfolgen ist, solange jeder darauf liegende Wert auch noch größer als T_1 ist [Nis+11, S. 178].
Nur schwache Kanten, die nicht mit einer Kantenstärke von mindestens T_2 verbunden sind, verschwinden aus dem Kantenbild. Das endgültige Ergebnis des Canny-Algorithmus ist in Abbildung 2.15 zu sehen.

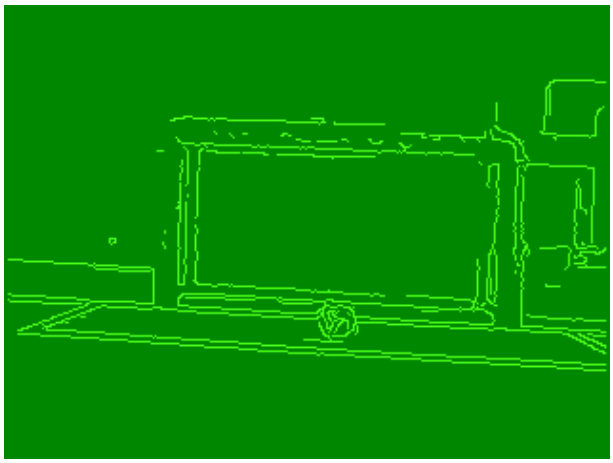


Abbildung 2.15: Resultat des Canny-Algorithmus

2.4 Alternative Kantendetektion

Thomas Reinhardt ist Mitglied des *Standard Platform League*-Teams der Hochschule für Technik, Wirtschaft und Kultur Leipzig (HTWK) und hat im Jahr 2011 in seiner Masterarbeit "Kalibrierungsfreie Bilderverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußballëinen eigenen Kantenerkennungsalgorithmus vorgestellt [Rei11].
Auch dieser arbeitet wie der Canny-Algorithmus auf dem Helligkeitskanal eines Bildes und betrachtet die Ableitung der Intensität. Ein wesentlicher Unterschied ist das von vornherein zeilen- und spaltenweise Auslassen von Information (Scanlines) zur Geschwindigkeitsoptimierung. Er trifft die Annahme, dass relevante Kanten eine minimale Ausdehnung von 2 Pixeln haben, somit werden keine relevanten Informationen missachtet.
Die Ableitung 2.12 der Intensitätsänderung $df(x)$ bezogen auf die Bilddistanz dx wird für den diskreten Fall mit dem symmetrischen Gradienten approximiert 2.13.

$$f'(x) := \frac{df(x)}{dx} \tag{2.12}$$

$$g(x) := \frac{1}{2}(f(x+1) - f(x-1)) \tag{2.13}$$

Auf die in Abbildung 2.16 abgebildete exemplarische Scanline angewendet, liefert der symmetrische Gradient die in Abbildung 2.17 dargestellte Kurve.

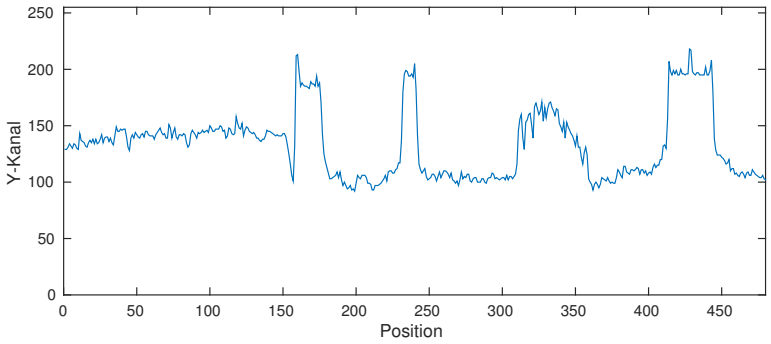


Abbildung 2.16: Eine von der NAO-Kamera stammende 480 Pixel lange Scanline

Die lokalen Maxima und Minima des symmetrischen Gradienten geben Aufschluss über die genaue Position einer Kante. Der in 2.17 eingezeichnete Schwellenwert t_{edge} erlaubt eine Mindesthöhe festzulegen, ab der ein Intensitätssprung als Kante detektiert wird.
Der Algorithmus errechnet zuerst zwischen zwei Helligkeitswerten den symmetrischen Gradienten aus. Liegt dieser außerhalb des Intervalls $[-t_{edge}, t_{edge}]$ oder ist er größer als ein vorläufiger maximaler Gradient, ersetzt er diesen. Die dazugehörige Position

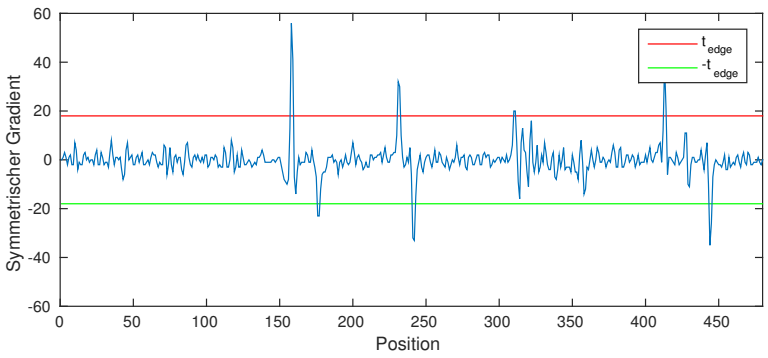


Abbildung 2.17: Symmetrischer Gradient einer Scanline

wird als Peak hinterlegt. Sobald der nächste Gradient sich wieder in dem Intervall $[-t_{edge}, t_{edge}]$ befindet, wird der zuletzt abgespeicherte Peak als Kante übernommen.
Der symmetrische Gradient wird zwischen zwei Pixeln berechnet, somit liegt dieser auch dazwischen. Der Algorithmus nimmt eine Korrektur der Peakposition x_{peak} vor und platziert damit die Kante an korrekter Position.
Wird dieser Algorithmus auf das bereits bekannte Kamerabild angewendet produziert es folgendes Ergebnis, welches in Abbildung 2.18 dargestellt ist. Der symmetrische Gradient wurde, wie auch im Falle der Masterarbeit, nur auf vertikale Scanlines angewendet.

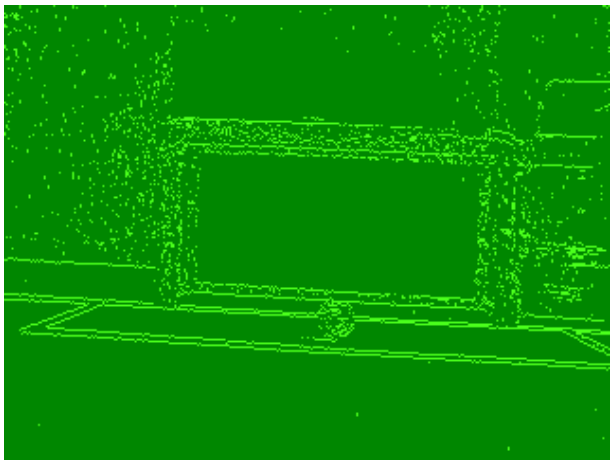


Abbildung 2.18: Kantenbild der HTWK

Der folgende Pseudocode stammt aus der besagten Masterarbeit [Rei11, S. 59].

Algorithm 1 Kantendetektion der HTWK

```
1:  function FINDEDGES(scanline, t_edge)
2:      edges := {}
3:      t_edge := t_edge * 2
4:      g_max := -t_edge
5:      g_min := t_edge
6:      x_peak := 0
7:      f_last := scanline_0
8:      for x := 2; x < sizeof(scanline); x := x + 2 do
9:          f_x := scanline_x
10:         g := f_x - f_last
11:         if g > g_max then
12:             if g_min < -t_edge then
13:                 edges := edges ∪ x_peak
14:             end if
15:             g_max := g
16:             g_min := t_edge
17:             x_peak := x - 1
18:         end if
19:         if g < g_min then
20:             if g_max > t_edge then
21:                 edges := edges ∪ x_peak
22:             end if
23:             g_min := g
24:             g_max := -t_edge
25:             x_peak := x - 1
26:         end if
27:         f_last := f_x
28:     end for
29:     return edges
30: end function
```

Kapitel 3

Laufzeitoptimierungen

Bei dem Canny-Algorithmus muss besonders stark auf die Implementierungsweise geachtet werden, damit er in Echtzeitanwendungen verwendet werden kann. Da mehrmals über das ganze Bild iteriert wird und zu jedem Pixel erneut Umliegende betrachtet werden, sorgt dies für viele Lese- und Schreibzugriffe.
Im Folgenden wird auf unterschiedliche Optimierungen eingegangen. Ihre exakten Laufzeiten werden in Abschnitt 4.2 evaluiert.

3.1 Scanlines

Wenn der Anwendungsfall bekannt ist, lassen sich Laufzeiten sehr gut optimieren, da sich Abschätzungen darüber treffen lassen, auf welche Informationen verzichtet werden kann. Ein gutes Beispiel dafür sind Scanlines, also das wiederholende Auslassen von horizontalen und vertikalen Bildreihen. Bei Verwendung jeder zweiten Zeile und Spalte wird die Anzahl der Speicherzugriffe, welche sehr aufwendig sind, auf 25% gesenkt. Es ist aber darauf zu achten, dass keine essentiellen Informationen verloren gehen, wenn zu erkennende Objekte sehr klein aufgenommen wurden. In dem Anwendungsfall Roboterfußball der *Standard Platform League* könnte die Erkennung der Feldlinien, welche nur $6,5cm$ breit sind [Com15, S. 1-4], in der Ferne problematisch werden.
Laut den Untersuchungen von Thomas Reinhardt treten nur äußert selten Linien mit einer Liniensegmentbreite von nur einem Pixel auf [Rei11, S. 50-53], welche bei der Verwendung von Scanlines verschwinden könnten.
Somit empfiehlt es sich, wie auch die Evaluierung der Laufzeiten in Abschnitt 4.2 zeigen wird, die Scanline-Technik zu implementieren.

3.2 Faltung

Der Canny-Algorithmus führt nacheinander insgesamt drei Faltungsoperation durch. Im Folgenden wird auf unterschiedliche Optimierung eingegangen, die die Berechnung ei-

ner Faltung bezüglich ihrer Ausführungszeit verbessern werden.

Bildrand In Kapitel 2.2.2 wurde bereits die Problematik des Bildrandes bei der Faltungsoperation behandelt. Die ständige Abfrage, ob es sich um einen Pixel am Bildrand handelt, hat einen großen Einfluss auf die Laufzeit. Somit bietet sich die Einführung eines Offsets an, die eine solche Überprüfung hinfällig macht und zusätzlich dafür sorgt, dass ein leicht kleineres Bild bearbeitet wird.
Die Berechnung beginnt folglich nicht bei $I(0,0)$, sondern bei $I(1,1)$, wenn ein 3×3 Filterkern verwendet wird.
Das Ergebnisbild wird in der Breite und Höhe um jeweils 2 Pixel kleiner und resultiert bei einer ursprünglichen Bildgröße von 320×240 in einem Informationsverlust von knapp $1,5\%$ am Bildrand.
Dieses Verfahren lässt sich für jede aufeinander folgende Faltungsoperation wiederholen, dafür wird aber auch das Ergebnisbild immer etwas kleiner.

Pixelzugriff Es ist besonders wichtig zu wissen, wie die Bilddaten im Speicher vorliegen, denn nur so lässt sich der Zugriff auf diese optimieren.
Angenommen die Pixel liegen in einem eindimensionalen Array hintereinander, dann empfiehlt sich nicht der Zugriff über eine Funktion, die zu *x*- und *y*-Werten in Bildkoordinaten die dazugehörige Arrayposition errechnet. Denn im Canny-Algorithmus sind so viele Bildzugriffe notwendig, dass das ständige Errechnen der Arrayposition, welches eine Multiplikation und eine Addition beinhaltet, zu lange dauert. Es ist ratsam mehrere Pointer in dem Array zu verteilen und diese allein über Additionen zu positionieren.

Bitweise Verschiebung Wie bereits angedeutet ist die Multiplikation für den Prozessor sehr aufwendig, nicht hingegen bitweise Operationen, die direkt auf die im Speicher liegenden Binärzahlen angewendet werden können. Das Verschieben um ein Bit nach links resultiert in einer Multiplikation mit 2, respektive das Verschieben um ein Bit nach rechts in einer Division durch 2.
Allgemein ausgedrückt ist die Verschiebung um n eine Multiplikation bzw. Divison mit 2^n. Tabelle 3.1 zeigt die Funktionsweise der bitweisen Verschiebung.

Tabelle 3.1: Prinzip der bitweisen Verschiebung

Dezimalzahl	**Bitweise Verschiebung**	**Dezimalzahl**
1	$0001 << 1 = 0010$	2
1	$0001 << 2 = 0100$	4
8	$1000 >> 3 = 0001$	1

Damit dieses Verfahren auf jede Multiplikation in der Implementierung des Canny-Algorithmus angewendet werden kann, impliziert es die Nutzung des

3×3 Gauß-Filterkerns, der in Gleichung 3.1 definiert ist.

$$H_{3\times3} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{3.1}$$

Jedes Matrixelement des Filterkerns und der Normierungsfaktor lassen sich als Zweierpotenzen darstellen, was für die bitweise Verschiebung elementare Grundlage ist.

3.3 Euklidischer Betrag

In dem Abschnitt 2.3.2.3 ist die Berechnung der absoluten Kantenstärke aus den partiellen Ableitungen beschrieben. Das Quadrieren und anschließende Ziehen der Wurzel ist sehr aufwendig, besonders weil diese Operationen für jeden einzelnen Pixel durchgeführt werden müssen, somit wird der euklidische Betrag häufig durch Gleichung 3.2 approximiert, die allein auf der Addition der Beträge beruht.

$$G(x,y) = |G_x(x,y)| + |G_y(x,y)| \tag{3.2}$$

3.4 Separierbarkeit

Eine nützliche Eigenschaft einiger Faltungskerne ist die Separierbarkeit. Darunter versteht man die Auftrennung der Filtermatrix in eine Multiplikation aus einem Zeilen- und Spaltenvektor [BB09, S. 136]. Hier am Beispiel des Sobel-Operators in x-Richtung in Gleichung 3.3. Es ist zu beachten, dass sich nicht jeder Kern auftrennen lässt.

$$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} = \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} \cdot \begin{bmatrix}1 & 0 & -1\end{bmatrix} \tag{3.3}$$

Die algebraischen Eigenschaften der Faltungsoperation erlauben, anstelle der Faltung mit dem gesamten Kern, zwei aufeinander folgende Faltungen durchzuführen. Zuerst wird das Bild A mit einem der beiden Vektoren gefaltet und das entstehende Zwischenergebnis daraufhin mit dem anderen Vektor.

$$A' = \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} * \begin{bmatrix}1 & 0 & -1\end{bmatrix} * A \tag{3.4}$$

Es wird dadurch eine deutliche Einsparung von Rechenzeit erreicht.
Wird ein $N \times N$ Filter auf ein Bild angewendet, sind für einen Pixel N^2 Lesezugriffe im

Bild notwendig. Daraufhin werden N^2 Multiplikation der Werte mit den Filtermatrixelementen durchgeführt. Die anschließende Summation zur Errechnung des *hot spots* erfolgt in $N^2 - 1$ Schritten.
Bei einem separierten Kernel sind hingegen nur $2N$ Lesezugriffe und somit auch nur $2N$ Multiplikationen notwendig. Die Aufsummierung kann in $2(N-1)$ Additionen erfolgen.
Folgende Tabelle 3.2 zeigt die Einsparungen am Beispiel eines 320×240 Pixel Bild mit einem 3×3 Filter.

Tabelle 3.2: Einsparung von Rechenzeit durch Separation

	Lesezugriffe	**Multiplikationen**	**Additionen**
3×3 Filter	691200	691200	614400
Separation	460800	460800	307200
Einsparung	33%	33%	50%

Bei größeren Filtern steigt die Einsparung der Rechenzeit natürlich deutlich wegen der quadratischen Abhängigkeit.

3.5 Approximation des atan2

Die Berechnung des atan2 stellt sich in der Laufzeitmessungen als sehr aufwendig dar, besonders wenn für jeden Pixel in einem Bild die Gradientenrichtung berechnet werden muss, wie es bei dem Canny-Algorithmus der Fall ist.
Idealerweise sind nur die gerundeten Winkel $0°, 45°, 90°$ und $135°$ notwendig, da ein Pixel nur 8 Nachbarn in diesen entsprechenden Richtungen hat.
Clement Boesch beschreibt ein Verfahren, dass sehr ähnlich der Methodik in *Efficient Approximations for the Four-Quadrant Arctangent Function* [RI06] ist.
Zur Laufzeitverbesserung wird nicht $\text{atan2}(G_y, G_x)$ ausgewertet, sondern G_y/G_x gegenüber zweier Tangens bestimmter Referenzwinkel vergleichen, um eine grobe Einschätzung der Gradientenrichtung zu erhalten [Boe15]. Die Tangens der Referenzwinkel sind wie folgt definiert 3.5.

$$\begin{aligned} \tan\left(\frac{\pi}{8}\right) &= \sqrt{2}-1 \\ \tan\left(\frac{3\pi}{8}\right) &= \sqrt{2}+1 \end{aligned} \tag{3.5}$$

Diese Referenzwinkel liegen auf den Grenzen, mit denen entschieden wird, auf welche der vier möglichen Winkel zu runden ist. Ist G_x negativ, so wechselt es, wie auch G_y, sein Vorzeichen. Dies sorgt für Winkel, welche sich ausschließlich im ersten und vierten Quadranten befinden.

Ist G_y/G_x größer als $-(\sqrt{2}-1)$, aber kleiner $\sqrt{2}-1$ so handelt es sich um einen Gradientenwinkel um die 0°. Ergibt die Division einen Wert zwischen $\sqrt{2}-1$ und $\sqrt{2}+1$, so liegen etwa 45° vor. Respektive $-45°$ oder auch 135°, wenn das Ergebnis der Division zwischen $-(\sqrt{2}-1)$ und $-(\sqrt{2}+1)$ liegt. Außerhalb dieser Intervalle kann es sich nur um 90° handeln.
Grafik 3.1 veranschaulicht die gewählten Referenzwinkel.

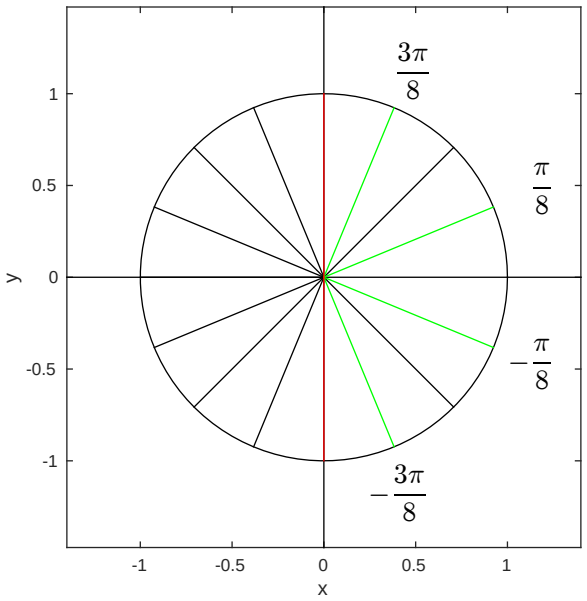


Abbildung 3.1: Referenzwinkel zur atan2 Approximation

Durch diese Optimierung lässt sich bereits viel gewinnen, doch stört die Notwendigkeit von Gleitkommazahlen. Integeroperationen sind deutlich günstiger.
Anstelle eines Vergleichs des Quotienten G_y/G_x mit Vielfachen von $\sqrt{2}-1$, lässt sich auch G_y mit $G_x \cdot \sqrt{2}-1$ vergleichen. Dies spart die Division und macht es möglich beide Seiten um 16 Bits nach links zu verschieben, was einer Multiplikation mit 65536 gleich kommt. Nun ist möglich auf einem 32-Bit Integer zu speichern, ohne die Genauigkeit der Gleitkommazahl zu verlieren.
Der 32-Bit Integer bietet sich an, da G_x bzw. G_y nur im Intervall $[-1020, 1020]$ liegen können ($[(-1)\cdot 255 + (-2)\cdot 255 + (-1)\cdot 255; 1\cdot 255 + 2\cdot 255 + 1\cdot 255]$). Siehe auch Gleichung 2.8 und 2.9. Der Wertebereich eines unsigned 32-Bit Integers liegt bei $[0, 4294967296]$ und das Ergebnis der Multiplikation kann nicht außerhalb liegen ($1020 << 16 = 66846720$).

Kapitel 4

Evaluation

Die Schwierigkeit der Evaluierung von Kantenerkennungsalgorithmen ist die Festlegung einer Metrik. Es ist nicht sinnvoll Abstände auf dem Bild zwischen detektierten Kanten zu deren *Ground Truth*-Daten zu messen, denn der Algorithmus lässt sich entsprechend empfindlich einstellen, so dass das Kantenbild übersät ist und zu den zugrundeliegenden existierenden relevanten Kanten keine Abstände herrschen. Das Kantenbild ist unter diesen Bedingungen keineswegs brauchbar.
Es ist, abhängig von dem Anwendungsfall, also festzulegen welche Kanten allein zu detektieren sind. Doch sind die in den Abschnitten 2.3 und 2.4 vorgestellten Algorithmen nicht auf einen bestimmten Anwendungsfall hin entwickelt worden, somit werden in den entstehenden Kantenbildern auch immer irrelevante Informationen enthalten sein, die bei der Weiterverarbeitung unterschiedlich stören.
Zur Evaluierung wurden fünfzehn Bildsequenzen mit dem NAO-Roboter aufgenommen und ihm diese darauffolgend anstelle seiner Kamerabilder geliefert, damit das Einstellen der Parameter und deren Auswertung immer auf gleicher Datengrundlage stattfinden kann. Die Sequenzen wurden unter verschiedenen Lichtbedingungen aufgenommen, um die Robustheit der Algorithmen gegen jegliche Störeinflüsse zu testen.
Die Auswertung mithilfe von *Ground Truth*-Daten kann nicht im Rahmen dieser Arbeit stattfinden, da eine händische Zugrundelegung relevanter Kanten auf über tausend Bildern einen gewissen zeitlichen Umfang mit sich bringt. Deswegen wird sich die Evaluierung auf eine rein visuelle Auswertung stützen.
Die Parameter beider Kantendetektionsalgorithmen wurden anhand Sequenzen, bestehend aus 1496 Einzelbildern, angepasst, so dass für das Fußballspiel wichtige Elemente sichtbar werden. Zur Selbstlokalisierung, Schießen des Balles und zum Zusammenspiel unter den NAOs ist es wichtig die Feldlinien, das Tor, den Ball und auch andere NAO-Roboter zu erkennen. In den Kantenbildern sollten dabei möglichst wenig irrelevante Kanten zum Beispiel durch Rauschen entstehen. Zudem wurde, mit Bedacht auf die nächste RoboCup Weltmeisterschaft, auf die Erkennung der Kanten des neuen Spielballes geachtet.

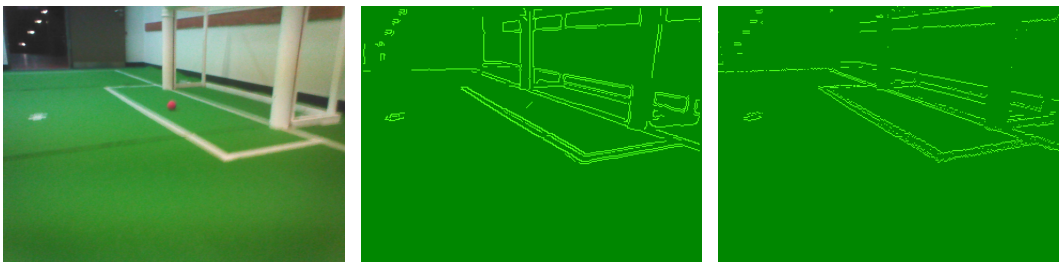
4.1 Vergleich der Kantenbilder

Im Folgenden werden die Kantenbilder des Canny-Algorithmus und die der HTWK in verschiedenen Situationen getestet und deren Vor- und Nachteile erläutert. Alle Bilder entstanden beim Laufen und weisen somit realistische Unschärfe auf. Der Canny-Algorithmus zeigt die besten Ergebnisse, wenn die Parameter der Hysterese auf $T_1 = 10$ und $T_2 = 120$ gestellt werden. Die Kantendetektion der HTWK liefert ihr bestes Kantenbild unter $t_{edge} = 8$. Die gewählten Parameter beider Algorithmen stehen in Abhängigkeit der vorherrschenden Kameraeinstellungen. Sollten sich diese Ändern, müssen auch die Parameter der Kantendetektion neu angepasst werden. In folgender Tabelle 4.1 sind die verwendeten Kameraeinstellungen dargestellt.

Tabelle 4.1: Verwendete Kameraeinstellungen

Belichtungszeit	14ms
Farbe	-22
Gamma	322
Kontrast	62
Sättigung	132
Schärfe	0
Verstärkung	76
Weißabgleich	5093

Das Bild 4.1a wurde bei Lichtverhältnissen aufgenommen, die ähnlich zu denen vergangener RoboCup-Veranstaltungen sind. Das Licht kommt zudem allein von Deckenlampen.

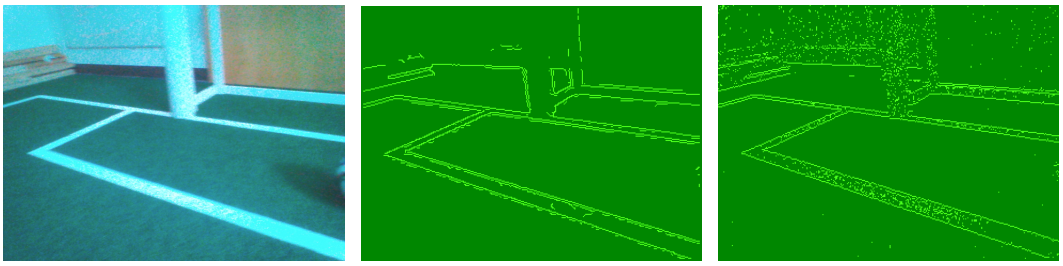


(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.1: Vergleich der Kantenbilder in regelkonformer Umgebung

Bei Betrachtung des Kantenbildes der HTWK fällt auf, dass vertikale Linien fehlen. Dies resultiert aus dem symmetrischen Gradienten, der nur auf vertikale Scanlines angewendet wird. Ein Helligkeitsunterschied entlang einer vertikalen Kante kann also nicht detektiert werden. Dazu wäre eine Ableitung in horizontale Richtungen notwendig, wie es bei dem Canny-Algorithmus der Fall ist.

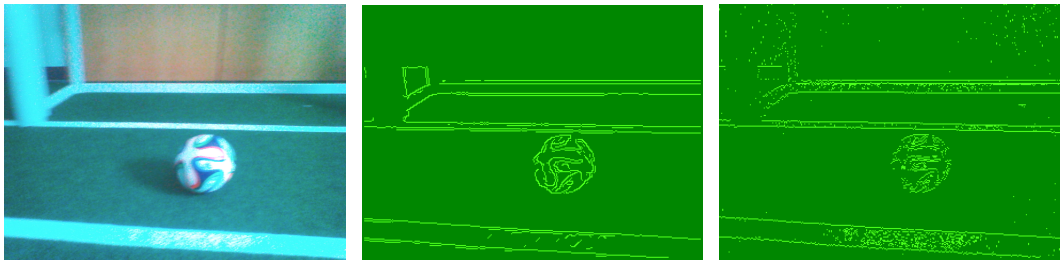
Dafür scheinen die Kanten in 4.1b leichter aufzubrechen, trotz Anwendung der Hysterese. Deutlich ist dies am oberen Feldrand und am Strafraum zu erkennen. In Abbildung 4.1c werden hingegen diese Kanten deutlich erkannt, was an einer empfindlichen Einstellung des Parameters t_{edge} liegt. Dies bringt aber Nachteile mit sich, wenn von



(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.2: Vergleich der Kantenbilder bei Sonnenlicht

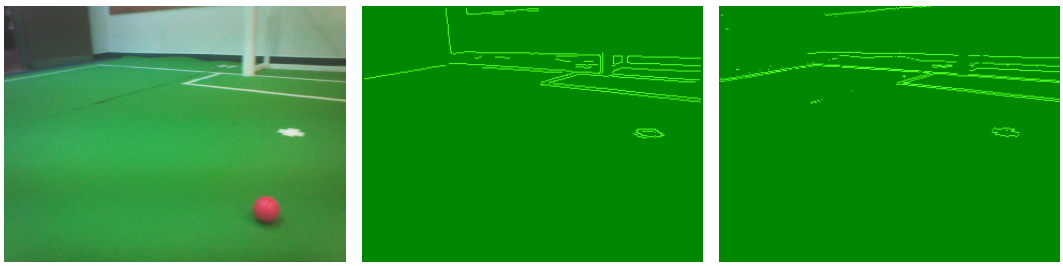
künstlichem Licht zu Sonnenlicht gewechselt wird. Im Jahr 2016 wird der RoboCup in Leipzig stattfinden. Die dortige Messehalle besitzt ein Glasdach, durch das Sonnenlicht strahlen wird. Die Lichtverhältnisse werden sich also über den Tag stark ändern.
Das Bild 4.2a wurde ohne künstliche Beleuchtung aufgenommen. Es strahlt nur Licht von der Seite durch ein Fenster. Bei gleicher Einstellung der Parameter zum vorherigen Bild, ist hier im Kantenbild der HTWK starkes Rauschen zu vernehmen. Im oberen Bildbereich stört dies zwar nicht, da meist keine Betrachtung über dem Horizont stattfindet, aber das Rauschen ist auch auf der Strafraumlinie zu erkennen. Diese irrelevanten Kanten verschwinden bei dem Canny-Algorithmus spätestens durch die Hysterese, wenn nicht bereits die Glättung des Bildes diese entfernt hat.
Der Algorithmus scheint also gegen wechselnde Lichtbedingungen deutlich stabiler zu sein. Schaut man sich aber das Bild 4.2b genauer an, werden auch diese unsauber, was sich in doppelten Antworten einer einzigen Kanten äußert.
Eine Anpassung der *Non-maximum suppression* würde diese nah beieinander liegende parallelen Kanten unterbinden und den Canny-Algorithmus noch robuster gestalten. Dazu ist der Operationsbereich der NMS zu vergrößern, anstatt nur umliegende Pixel zu betrachten.
Ein großer Nachteil ist die nicht vorhandene Umrandung des Torpfostens im Kantenbild 4.2c. Aber auch der Canny-Algorithmus hat Probleme, den weißen Torpfosten vor einer gleichfarbigen Wand zu erkennen.
Das Kantenbild 4.3b zeigt, an welchen Merkmalen der wohl zukünftige Spielball zu erkennen ist. Die Parameter lassen sich nicht dahingehend einstellen, wie es bei dem roten Ball möglich war, dass er sich sauber vom Hintergrund trennt. Es ist bei einem solchen Muster also nicht mehr möglich nach kreisrunden Formen zu suchen. Hinzu kommt, dass das Muster vom Betrachtungswinkel abhängig ist.
Ein Vorteil des neuen Balles ist die starke Präsenz im Y-Kanal. Für den roten Ball musste der Kantendetektionsalgorithmus sehr empfindlich eingestellt werden, wie Abbildung



(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.3: Vergleich der Kantenbilder des neuen Spielballes

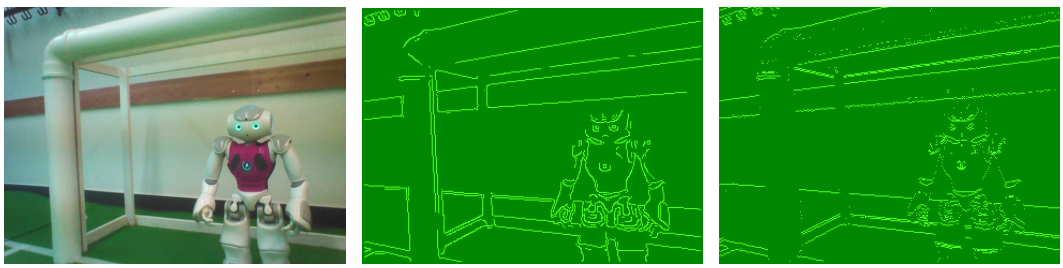
4.4 eindrucksvoll verdeutlicht.
Aufgrund der starken Abhebung von dem Hintergrund ist zu erwarten, dass beide Algorithmen den alten Ball, welcher in Abbildung 4.4a zu sehen ist, detektieren können. Der rote Ball weist aber nur im Cr-Kanal starke Pixelwertdifferenzen auf.



(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.4: Vergleich der Kantenbilder des roten Spielballes

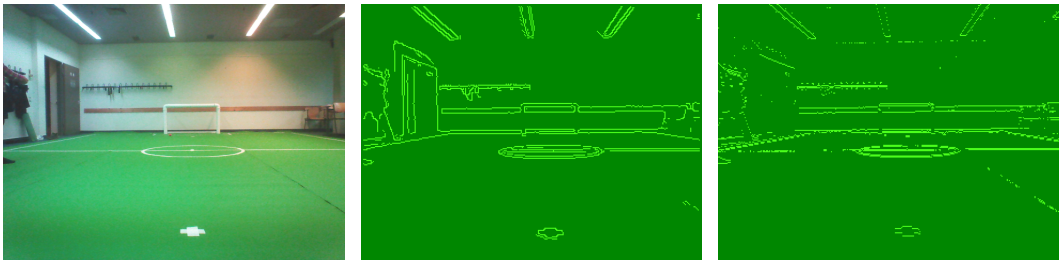
Für den Menschen ist die Erkennung und Zuordnung allein anhand des Kantenbildes keine Schwierigkeit. Doch wie auch die Bilder 4.5b und 4.5c zeigen, kann kaum festgemacht werden, woran ein Robotersystem andere NAOs erkennen kann.
Die Hüftgelenke sind verwechselnd ähnlich zu zwei Bälle in größerer Entfernung.



(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.5: Vergleich der Kantenbilder eines weiteren NAO-Roboters

Schaut der NAO über das ganze Spielfeld, wäre durch den Mittelkreis und die hindurch führende Linie eine hervorragende Selbstlokalisierung möglich. Beide Kantenbilder zei-



(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.6: Vergleich der Kantenbilder des ganzen Spielfelds

gen in der Hinsicht ähnliche Ergebnisse und der Feldrand ist jeweils klar zu erkennen. Nur die äußeren Feldlinien sind zu fein, um sie zu detektieren.
Die Kantenbilder beider Algorithmen sind auf eine solche Entfernung kaum zu unterscheiden, bis auf die kleinen irrelevanten Kanten und den Spalt im Teppich in Bild 4.6c. Der Canny-Algorithmus produziert ein homogeneres Ergebnis durch die Vorprozessierung des Gaußschen Weichzeichners und Verfolgung der starken Kanten.

(a) Originalbild (b) Canny-Algorithmus (c) HTWK

Abbildung 4.7: Vergleich der Kantenbilder in einer *Worst-Case-Situation*

Das Bild 4.7a wurde, nach dem aktuellen Regelwerk, in unrealistischen Lichtbedingungen aufgenommen. Zudem ist es durch die Bewegung des Roboters sehr unscharf. Für eine Erkennung des Feldrandes ist es im oberen rechten Bereich bereits zu dunkel. Bei sehr genauer Betrachtung ist sogar der *Electronic-rollingshutter*-Effekt zu erkennen. Es sind zwei sehr schwach gezeichnete, eng beieinander liegende Strafräume zu sehen, die nahezu keine Antwort in den partiellen Ableitungen erzeugen und somit auch nicht als Kante auftreten.
Hervorzuheben ist die nahezu einwandfreie Erkennung der Torpfosten beim Canny-Algorithmus.

4.2 Laufzeiten

Im Folgenden wird besonders auf die Laufzeiten des Canny-Algorithmus eingegangen, da eine echtzeitfähige Implementierung einigen Aufwand erfordert.
Die Laufzeiten wurden auf dem NAO-Roboter gemessen, während der parallelen Ausführung aller auch sonst vorhandenen Module, die zum Fußballspielen notwendig sind. Dies resultiert in realistischen Ausführungszeiten. Zukünftige Implementierungen sinken in der Laufzeit, da das Bild nicht über dem Horizont verarbeitet wird, da dort keine wichtigen Informationen zu erwarten sind. Die Laufzeiten würden sich also durch Betrachtung des verkleinerten Bildausschnittes verkürzen. Zum Vergleich beider Algorithmen bietet sich diese Optimierung nicht an.
Tabelle 4.2 zeigt die Laufzeitveränderung einiger in Kapitel 3 erwähnten Optimierungen, anhand einer naiven Implementierung. Wird der Algorithmus dahingehend optimiert, dass die Betrachtung des Bildrandes durch Einführung eines Offsets (O) entfällt, lassen sich wenige Millisekunden sparen.
Werden darauf Scanlines (S) implementiert, die nur noch einen Viertel des Bildes betrachten, weil jede zweite Zeile und Spalte ausgelassen wird, sinkt auch die Ausführungszeit auf fast ein Viertel im Vergleich zur vorherigen Optimierung.
Der Zugriff auf das Bild über x und y Koordinaten erfordert die Berechnung der Pixelposition im Array. Dies wird durch Verwendung von Pointern (P) hinfällig und resultiert in einer Laufzeithalbierung.
Einen nur kleinen Einfluss hat die bitweise Verschiebung (BV) zur Realisierung von Multiplikation und Division, da bereits von vornherein der Compiler den Code teilweise optimiert hat.
In Abbildung 4.8 sind Laufzeiten zu exemplarischen 250 Bildern dargestellt.

Tabelle 4.2: Laufzeitoptimierungen des Gaußschen Weichzeichners

	Laufzeit $[ms]$		
Gauß-Filterkern $H_{3\times3}$	*min*	*avg*	*max*
Naive Implementierung	24.3800	26.4022	31.8910
O	21.8700	23.3595	25.9510
O + S	5.1690	6.2190	10.3310
O + S + P	2.2310	2.7150	5.3180
O + S + P + BV	1.8920	2.3234	3.6580

Die Tabelle 4.3 zeigt, die bei Echtzeitanwendungen unvertretbare, Laufzeit einer naiven Implementierung der Berechnung der Kantenstärke und ihrer Richtung für ein ganzes Bild. Eine leichte Optimierung bringt die Verwendung der Approximation des euklidischen Algorithmus (AEB) aus Abschnitt 3.3. Zu beachten ist die Abweichung im Durchschnitt von 21,28% zum exakten Wert. Dieser Wert entstand in 181496 Messungen.

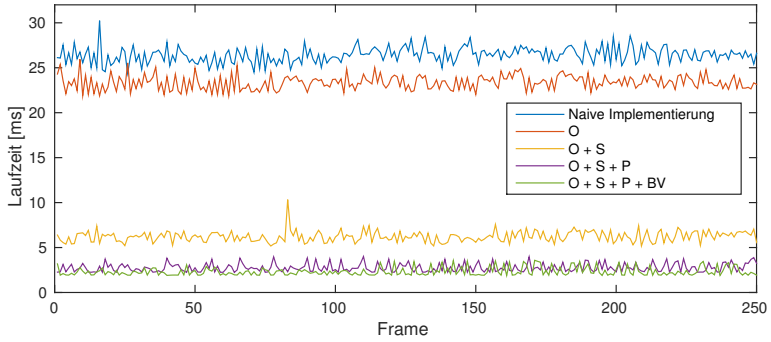


Abbildung 4.8: Laufzeitverlauf der Optimierungen des Gaußschen Weichzeichners

Wird zusätzlich die Approximation des atan2 (AA2), des Abschnittes 3.5 verwendet, kann eine deutliche Reduzierung der Ausführungszeit erzielt werden.

Tabelle 4.3: Laufzeitoptimierungen durch Approximationen der Kantenstärke und Richtung

Kantendetektion	**Laufzeit** $[ms]$		
	min	*avg*	*max*
Naive Implementierung	42.2140	45.4118	49.4470
AEB	35.9830	39.1823	43.2570
AEB + AA2	4.9420	6.1019	8.6200

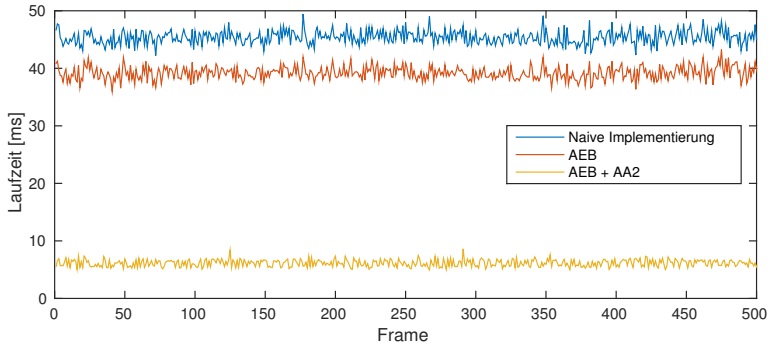


Abbildung 4.9: Laufzeitverlauf der Kantendetektionsoptimierungen

Außerdem wurde in Abschnitt 3.4 auf die Separation der Kerne eingegangen. Rechnerisch werden dadurch 33% an Lesezugriffen und Multiplikationen, sowie 50% der Additionen gespart. Die Erwartung ist also eine Reduktion der Laufzeit, jedoch zeigte eine Implementierung das Gegenteil. Dies könnte daran liegen, dass zweimal nacheinander über das Bild iteriert werden musste und es nicht zur optimalen Ausnutzung des

Caches kommen konnte. Es sind in diesem Bereich tiefer gehende Untersuchungen vonnöten.
Die Laufzeiten der einzelnen Schritte des Canny-Algorithmus sind in Tabelle 4.4 festgehalten. Es ist deutlich zu sehen, dass die eigentliche Kantendetektion, also die Berechnung der partiellen Ableitungen, ihre Summation und Bestimmung der Kantenrichtungen, am längsten dauert. Lässt sich die Faltung weiter optimieren, profitiert davon die Sobel-Operation sehr stark. Zudem wird auch die Laufzeit des Gaußschen Weichzeichners abnehmen.

Tabelle 4.4: Laufzeitvergleich der Schritte des Canny-Algorithmus

	Laufzeit $[ms]$		
	min	*avg*	*max*
Gaußscher Weichzeichner	1.8910	2.3536	3.7460
Sobel	4.9420	6.1019	8.6200
Non-maxiumum suppression	1.7160	2.2227	4.1720
Edge Tracking	0.8710	1.4232	3.0270

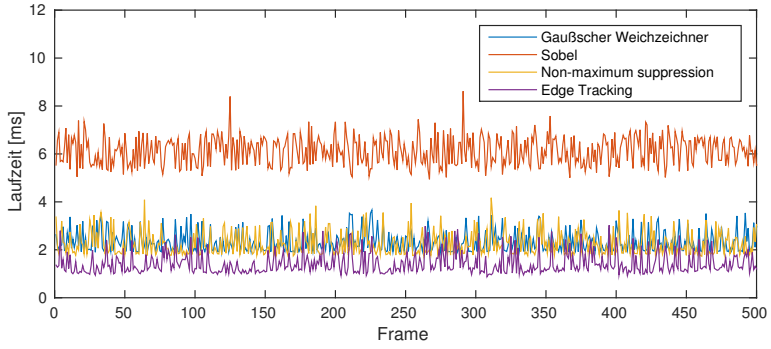


Abbildung 4.10: Laufzeitverlauf der Schritte des Canny-Algorithmus

Im Vergleich zu der Kantendetektion der HTWK schneidet der Canny-Algorithmus nicht besonders schlecht ab, wenn bedacht wird, dass nicht nur horizontale, sondern auch vertikale Kanten detektiert werden. Der Algorithmus der HTWK ließe sich zusätzlich auf horizontale Scanlines erweitern, wodurch sich auch die Laufzeit verdoppelt. Zu erwarten wäre ein noch ähnlicheres Kantenbild.
Trotzdem ist festzuhalten, dass der HTWK-Algorithmus deutlich schneller, bei einem überwiegend identischen Kantenbild, ist.

Tabelle 4.5: Laufzeitvergleich beider Kantendetektionen

	Laufzeit $[ms]$		
	min	*avg*	*max*
Canny-Algorithmus	10.2850	12.1045	15.0280
Kantendetektion HTWK	4.1370	5.2895	9.4280

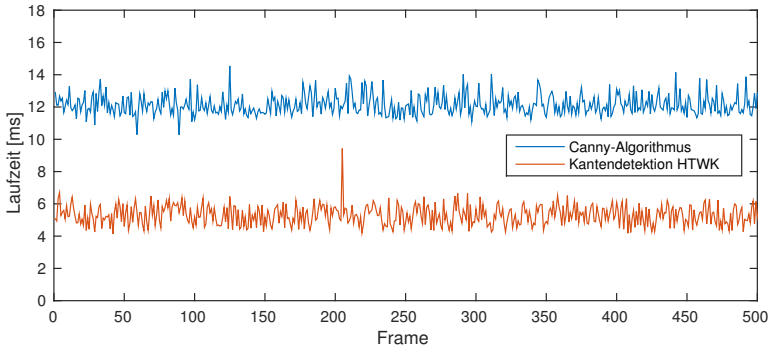


Abbildung 4.11: Laufzeitverlauf beider Kantendetektionen

4.3 Zusammenfassung der Ergebnisse und Ausblick

Beide Kantendetektionsalgorithmen liefern brauchbare Ergebnisse, die zur Selbstlokalisierung verwendet werden können, da Feldlinien und Feldrand zumeist erkannt werden. Allerdings haben beide Algorithmen ihre Schwierigkeiten mit dem Tor. Die Kantendetektion der HTWK kann es wegen der Scanlineausrichtung nicht erkennen und der Canny-Algorithmus versagt oft, wenn das Tor vor gleichfarbigen Hintergrund steht, was bei RoboCup Veranstaltungen, durch weiße Banden, nicht selten der Fall ist.
Beide Verfahren erzeugen von dem neuen Spielball, wie auch von anderen NAOs, ähnliche Kantenbilder. Die Zukunft wird zeigen, inwiefern die Ergebnisse brauchbar zur weiteren Verarbeitung sind, denn es ist eine Ballerkennung notwendig, die mit dem neuen Ball zurecht kommt.
Die vorherige Weichzeichnung stellt sich als besonders wichtig heraus, da die Kantenbilder der HTWK teilweise von starkem Rauschen betroffen sind. Der Gaußsche Weichzeichner hat allerdings den Nachteil, auch real existierende Kanten zu glätten. Die Implementierung eines nichtlinearen bilateralen Filters sorgt für das Erhalten real existierender Kanten bei gleichzeitiger Rauschweichzeichnung [RM98]. Dies könnte auch eine Lösung für die Torproblematik sein, da dann die leichte Kante zwischen weißem Hintergrund und Torpfosten erkannt werden könnte.
Es hat sich gezeigt, dass die Optimierungen des Canny-Algorithmus notwendig sind, um ihn ansatzweise echtzeitfähig zu gestalten.
Die HULKs verarbeiten 30 Bilder pro Sekunde. Dies impliziert, dass die visuelle Auswertungen und die Berechnung der Bewegungen und Spielabläufe innerhalb von $33ms$

zu vollziehen ist, da dann bereits das nächste Kamerabild geladen wird. Grob geschätzt bleiben $10 - 15ms$, indem die komplette Auswertung des Bildes stattzufinden hat. Ein Algorithmus der allein für die Kantenerkennung im Durchschnitt bereits $12ms$ benötigt, sorgt für eine Reduzierung der angestrebten 30 Bilder pro Sekunde. Die Laufzeit der HTWK Kantendetektion ist hingegen mit $6ms$ akzeptabel.
Beide Algorithmen bieten allerdings Platz zur weiteren Optimierung und damit auch die Möglichkeit zur Verwendung in Echtzeitanwendungen.
Da beim Canny-Algorithmus drei Faltungen durchgeführt werden und dies auch die Schritte mit der höchsten Laufzeit im gesamten Verfahren sind, sollte sich die Optimierung in erster Linie darauf konzentrieren.
Eine Möglichkeit ist die Verwendung von *Streaming SIMD Extensions (SSE)* [Kus14, S. 273]. Dies ist eine von Intel entwickelte Erweiterung des x86-Befehlssatzes und bringt 128 Bit Register mit sich, in denen 4 Pixel gespeichert werden können. SSE macht es möglich eine Operation in einer einzigen Instruktion auf diese 4 Pixel anzuwenden, was in einer deutlichen Geschwindigkeitssteigerung resultiert.

Literatur

[BB09] Wilhelm Burger und MarkJames Burge. “Edges and Contours”. English. In: *Principles of Digital Image Processing*. Undergraduate Topics in Computer Science. Springer London, 2009, S. 1–26. ISBN: 978-1-84800-190-9. DOI: 10.1007/978-1-84800-191-6_6. URL: http://dx.doi.org/10.1007/978-1-84800-191-6_6.

[Boe15] Clement Boesch. *Fun and canny optim for a Canny Edge Detector*. English. 2015. URL: http://blog.pkh.me/p/14-fun-and-canny-optim-for-a-canny-edge-detector.html#content.

[Bur06] “Filter”. German. In: *Digitale Bildverarbeitung*. Hrsg. von Wilhelm Burger. X.media.press. Springer Berlin Heidelberg, 2006, S. 89–116. ISBN: 978-3-540-30940-6. DOI: 10.1007/3-540-30941-1_6. URL: http://dx.doi.org/10.1007/3-540-30941-1_6.

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

[Com15] RoboCup Technical Committee. *RoboCup Standard Platform League (NAO) Rule Book*. 2015. URL: http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2015.pdf.

[Gri13] Prof. Dr.-Ing. R.-R. Grigat. *Digital Image Processing. First-order derivative edge detection with noise reduction*. English. Technische Universität Hamburg-Harburg Vision Systems, 2013.

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

[Jia05] “Detektion lokaler Merkmale”. German. In: *Digitale Bildverarbeitung*. Hrsg. von Prof. Xiaoyi Jiang. Computer Vision und Pattern Recognition Group, 2005, S. 27–62. URL: http://www.uni-muenster.de.

[Kus14] Daniel Kusswurm. “X86-SSE Programming - Packed Integers”. English. In: *Modern X86 Assembly Language Programming*. Apress, 2014, S. 273–302. ISBN: 978-1-4842-0064-3. DOI: 10.1007/978-1-4842-0064-3.

[Mat15] MathWorks. *Four-quadrant inverse tangent*. 2015. URL: https://de.mathworks.com/help/matlab/ref/atan2.html.

[NA12] Mark S. Nixon und Alberto S. Aguado. “Chapter 4 - Low-level feature extraction (including edge detection)”. In: *Feature Extraction & Image Processing for Computer Vision (Third edition)*. Hrsg. von Mark S. NixonAlberto S. Aguado. Third edition. Oxford: Academic Press, 2012, S. 137 –216. ISBN: 978-0-12-396549-3. DOI: http://dx.doi.org/10.1016/B978-0-12-396549-3.00004-5. URL: http://www.sciencedirect.com/science/article/pii/B9780123965493000045.

[NFM07] Steven P. Nicklin, Robin D. Fisher und Richard H. Middleton. “Rolling Shutter Image Compensation”. English. In: *RoboCup 2006: Robot Soccer World Cup X*. Hrsg. von Gerhard Lakemeyer u. a. Bd. 4434. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, S. 402–409. ISBN: 978-3-540-74023-0. DOI: 10.1007/978-3-540-74024-7_39. URL: http://dx.doi.org/10.1007/978-3-540-74024-7_39.

[Nis+11] Alfred Nischwitz u. a. “Kanten und Linien”. German. In: *Computergrafik und Bildverarbeitung*. Vieweg+Teubner Verlag, 2011, S. 159–216. ISBN: 978-3-8348-1712-9. DOI: 10.1007/978-3-8348-8300-1_7. URL: http://dx.doi.org/10.1007/978-3-8348-8300-1_7.

[Poy12a] “10 - Constant luminance”. In: *Digital Video and {HD} (Second Edition)*. Hrsg. von Charles Poynton. Second Edition. The Morgan Kaufmann Series in Computer Graphics. Boston: Morgan Kaufmann, 2012, S. 107 –114. ISBN: 978-0-12-391926-7. DOI: http://dx.doi.org/10.1016/B978-0-12-391926-7.50010-2. URL: http://www.sciencedirect.com/science/article/pii/B9780123919267500102.

[Poy12b] “12 - Introduction to luma and chroma”. In: *Digital Video and {HD} (Second Edition)*. Hrsg. von Charles Poynton. Second Edition. The Morgan Kaufmann Series in Computer Graphics. Boston: Morgan Kaufmann, 2012, S. 121 –128. ISBN: 978-0-12-391926-7. DOI: http://dx.doi.org/10.1016/B978-0-12-391926-7.50012-6. URL: http://www.sciencedirect.com/science/article/pii/B9780123919267500126.

[Poy12c] “A - {YUV} and luminance considered harmful”. In: *Digital Video and {HD} (Second Edition)*. Hrsg. von Charles Poynton. Second Edition. The Morgan Kaufmann Series in Computer Graphics. Boston: Morgan Kaufmann, 2012, S. 567 –572. ISBN: 978-0-12-391926-7. DOI: http://dx.doi.org/10.1016/B978-0-12-391926-7.50066-7. URL: http://www.sciencedirect.com/science/article/pii/B9780123919267500667.

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

[RI06] Sreeraman Rajan und Sichun Wang und Robert Inkol. “Efficient Approximations for the Four-Quadrant Arctangent Function”. In: *IEEE CCECE/CCGEI, Ottawa* (2006).

[RM98] C. Tomasi und R. Manduchi. “Bilateral Filtering for Gray and Color Images”. In: *IEEE International Conference on Computer Vision, Bombay, India* (1998).

[Rob15a] Aldebaran Robotics. *Gallery*. 2015. URL: https://www.aldebaran.com/en/press/gallery/nao.

[Rob15b] Aldebaran Robotics. *Head 4.0*. 2015. URL: http://doc.aldebaran.com/1-14/family/robots/video_robot.html#robot-video.

[Rob15c] Aldebaran Robotics. *NAO Datasheet*. 2015. URL: https://www.aldebaran.com/sites/aldebaran/files/nao_datasheet.pdf.

[Rob15d] Aldebaran Robotics. *Who is NAO*. 2015. URL: https://www.aldebaran.com/en/humanoid-robot/nao-robot.

Anhang A

Inhalte der DVD

Ordner	**Inhalt**
/Bachelorarbeit	Kompilierte PDF und die zur Erstellung benötigten Quelldateien
/Paper	Alle referenzierten Paper dieser Arbeit
/Sequenzen	Die Bilder der verwendeten 15 Sequenzen inkl. aller Ergebnisse und Zwischenschritte in insgesamt 13464 Einzelbildern
/Software	Die Source- und Headerdatei des implementierten Canny-Algorithmus in C++