Julius-Maximilians-Universität Würzburg

Fakultät für Mathematik und Informatik

Informationstechnik für Luft- und Raumfahrt

Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro

Bachelorarbeit

Implementierung und Evaluierung einer Objekterkennung für einen Quadrocopter

Vorgelegt von

Christian Reul

Matr.-Nr.: 1716820

Prüfer: Prof. Dr. Sergio Montenegro

Betreuender wissenschaftlicher Mitarbeiter: Dipl.-Ing. Nils Gageik

Würzburg, 08. 04. 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.
Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Würzburg, 08. 04. 2013

_____

Christian Reul

Aufgabenstellung

Die Fortschritte im Bereich Sensorik und Mikrotechnik ermöglichen heutzutage den kostengünstigen Bau kleiner unbemannter Luftfahrzeuge (UAV, unmanned airial vehicle, Drohne) wie Quadrocopter sowie deren Ausstattung mit einer Vielzahl an Sensorik. Kameras sind heute für wenige Euros zu kaufen und stellen eine sehr mächtige Sensorik dar. Während es Systeme für die ferngesteuerte Anwendung schon zu kaufen gibt, ist die Forschung im Bereich autonomer Systeme noch in den Anfängen. Insbesondere der Indoor-Betrieb ist aufgrund fehlender absoluter Positionsstützung durch GPS problematisch. Der Aufbau eines eigenen autonomen Systems wird daher am Lehrstuhl Aerospace Information Technology der Universität Würzburg erforscht und erprobt. Im Rahmen dieses Forschungsvorhabens ist ein System zu entwickeln, das unter Verwendung von Videobildern eine Objekterkennung ausführt.

Hauptaugenmerk dieser Arbeit ist die Erkennung einfacher Objekte wie z.B. das Finden eines roten Balls auf grünem Grund. Das System soll Anwendung finden in einem Quadrocopter. Ziel der Arbeit ist die Entwicklung und Implementierung eines geeigneten Algorithmus zur Objekterkennung, wobei auf bestehende Lösungen aufgebaut werden soll. Eine ausführliche Übersicht über den Stand der Technik ist dabei erforderlich, die einen Einblick in bestehende Lösungsmöglichkeiten gibt. Dabei soll der Algorithmus sowohl Farben als auch Formen unterscheiden können. Die entwickelte Software ist in das bestehende System einzubinden und soll die Bilder der bereits implementierten CMOS-Kamera verwenden. Schließlich ist die Objekterkennung ausgiebig zu evaluieren. Zur Aufgabe gehört eine ausführliche Dokumentation der Software und der zugrundeliegenden Theorie.

Aufgabenstellung (stichpunktartig):

- Stand der Technik: Aufzählung & Erklärung von Lösungen zur Objekterkennung
- Algorithmus Objekterkennung erklären & implementieren: Farben & Formen
- Implementierung in QT mit CMOS-Kamera
- Anbindung an Quadrocopter für Echtzeitflug
- Evaluierung
- Dokumentation

Zusammenfassung

Thema dieser Arbeit ist die Entwicklung eines Systems zur Objekterkennung, das in einem autonomen Quadrocopter zur Innenraumerkundung zum Einsatz kommt. Das System erkennt kugelförmige Objekte (z.B. Bälle) während des Flugs wieder, deren Charakteristika (Größe/Radius und Farbe) vorher automatisch durch einen initialen Scan bestimmt werden. Zu diesem Zweck wurde eine nach unten gerichtete Kamera in die Bodenplatte des Quadrocopters installiert. Die so aufgenommenen Bilder werden mittels eines in den Quadrocopter integrierten Pico-ITX PCs im Flug verarbeitet. Zur Evaluierung der Vorgänge auf dem on-Board Computer des Quadrocopters dient eine Bodenstation, die via W-LAN und Remote-Desktop verbunden ist.

Der Algorithmus zur Suche nach dem Zielobjekt wurde je einmal ohne und unter Verwendung der OpenCV-*Library* implementiert, wobei die wesentlichen Schritte sich gleichen: Nach dem Herausfiltern der gesuchten Farbe wird eine Kantendetektion durchgeführt. Anschließend erfolgt die Kreisdetektion mittels der Hough-Kreis-Transformation. Das so entwickelte System wurde sowohl im Quadrocopter als auch unter Laborbedingungen ausführlich evaluiert. Dabei wurde besonders der Einfluss variierender Höhen, wechselnder Lichtverhältnisse und horizontaler Bewegungen auf die Zuverlässigkeit der Detektion untersucht. Die Evaluierung zeigt somit die Machbarkeit dieser Realisierung sowie deren Grenzen.

Inhaltsverzeichnis

1. Einleitung

Aktuell wird die Verwendung autonomer Drohnen (Quadrocopter) in vielen Anwendungs-Szenarien erprobt. Diese reichen vom Einsatz in der Landwirtschaft, Archäologie und Geoinformatik zur Luftbildaufnahme [Microdrones 2013] über die Biologie zum Zählen von Pinguinen [ZeitOnline 2011] bis hin zu Rettungs- und Katastropheneinsätzen. Bei Letzterem sollen die Quadrocopter zukünftig sowohl bei der Erkundung von Gebäuden, z.B. zum Aufspüren von Giftstoffen in der Luft [DRadioWissen 2010], als auch zur Wiederherstellung von Kommunikationsnetzen [SpiegelOnline 2010] zum Einsatz kommen. Für viele dieser Szenarien ist eine Objekterkennung erforderlich, damit das System vollkommen autonom agieren kann.

Im Rahmen des AQopterI8-Projekts des Lehrstuhls Informatik VIII für Aerospace Information Technology werden an der Universität Würzburg Quadrocopter (vgl. Abb. 1) zur Innenraumerkundung entwickelt. Sie unterscheiden sich von ihren oft als „Spielzeug“ kommerziell vertriebenen Namensvettern v.a. durch ihre Fähigkeit autonom zu agieren.

Abbildung 1: Quadrocopter der Universität Würzburg [Gageik 2012]

Diese Drohnen sind dazu in der Lage, auch ohne den kontrollierenden Einfluss eines Menschen mit Fernsteuerung, sich selbstständig fortzubewegen, dabei ihre Höhe zu regulieren und Kollisionen zu vermeiden. Des Weiteren kommen verschiedenste Sensoren wie Infrarot-, Ultraschall- und Luftdruck-Sensoren zum Einsatz. Eine neu entwickelte Mapping-Software erlaubt es dem Quadrocopter zusätzlich eine Karte seiner direkten Umgebung zu erstellen. Diese Fähigkeiten sollen es später ermöglichen, Einsätze in für Menschen unzugänglichen oder zu gefährlichen Gebieten zu fliegen. So ist z.B. ein Einsatz in brennenden Häusern oder eingestürzten Minenschächten denkbar. Dort könnte ein Quadrocopter das Ausmaß des Schadens bestimmen oder nach eventuellen Überlebenden suchen [Gageik 2012]. Speziell für Letzteres ist es nötig, eine Möglichkeit der optischen Detektion in das bereits bestehende System zu integrieren. Jedoch reicht der bloße Einbau einer Kamera nicht aus. Dem Quadrocopter muss es möglich sein, selbstständig Objekte von Interesse, z.B. einen am Boden liegenden Körper, in einem zuvor

aufgenommenen Bild zu erkennen. Die Grundlage dafür soll im Verlauf dieser Arbeit geschaffen werden. Ziel ist es einen Algorithmus zu entwickeln, der mit Hilfe einer Kamera ein Objekt mit vorgegebenen Eigenschaften, Farbe und Form, innerhalb des Bildes identifiziert und lokalisiert. Zunächst wird sich dabei auf kreisförmige Objekte wie Bälle oder Kugeln beschränkt.

Von großer Wichtigkeit ist, dass die so entwickelte Objekterkennung an die speziellen Anforderungen der Einsatzumgebung, also in einem Quadrocopter, angepasst ist. So muss sie nicht nur in der Lage sein, Objekte zuverlässig aus unterschiedlichen Flughöhen zu erkennen, sondern darf auch nicht durch die Vorwärtsbewegung des Quadrocopters zu sehr eingeschränkt werden. Des Weiteren ist es nötig, dass die Objekterkennung mit den an Board des Quadrocopters zur Verfügung stehenden Ressourcen auskommt. Dies gilt besonders für die relativ geringe Rechenleistung, die zum einen zu einer kleineren Bildrate führt und zum anderen das Arbeiten mit einer nur geringen Kameraauflösung zulässt.

2. Stand der Technik

2.1. Computer-Vision

Bei Computer-Vision handelt es sich um die Transformation von Daten einer (Video-) Kamera hin zu einer neuen Repräsentation, mit deren Hilfe eine Entscheidung zu einer bestimmten Fragestellung getroffen werden kann [OpenCV 2008].

2.1.1. Ziele

Diese Transformationen werden stets durchgeführt, um ein bestimmtes Ziel zu erreichen. So könnte die Entscheidung z.B. darin bestehen, ob sich in einem Bild ein Auto befindet oder in einem Ausschnitt Tumorzellen zu erkennen sind. Eine neue Repräsentation könnte durch die Umwandlung eines Farbbildes in ein Schwarz-Weiß-Bild oder das Entfernen der Kamera-Bewegung aus einer Bilderfolge erreicht werden [OpenCV 2008].

2.1.2. Unterschiede: Menschliches Auge – Computer-Vision

Zumindest die eben erwähnte Aufgabe, ein Auto in einem Bild zu identifizieren, klingt zunächst trivial, da der Mensch sehr visuell veranlagt ist. Unsere Hardware-Sensoren, die Augen, senden ständig riesige Mengen an Signalen an das Gehirn. Dort werden die Signale in verschiedene Kanäle aufgeteilt, die jeweils unterschiedliche Arten an Information enthalten. So werden z.B., je nach Aufgabe, wichtige Ausschnitte des Ziel-Bildes näher untersucht, während unwichtigere Areale ausgeblendet werden. All dies geschieht intuitiv bzw. automatisch, ebenso wie die Regulierung des Lichteinfalls durch die Iris oder die Umwandlung des eintreffenden Lichts in Nervensignale durch die Netzhaut [OpenCV].

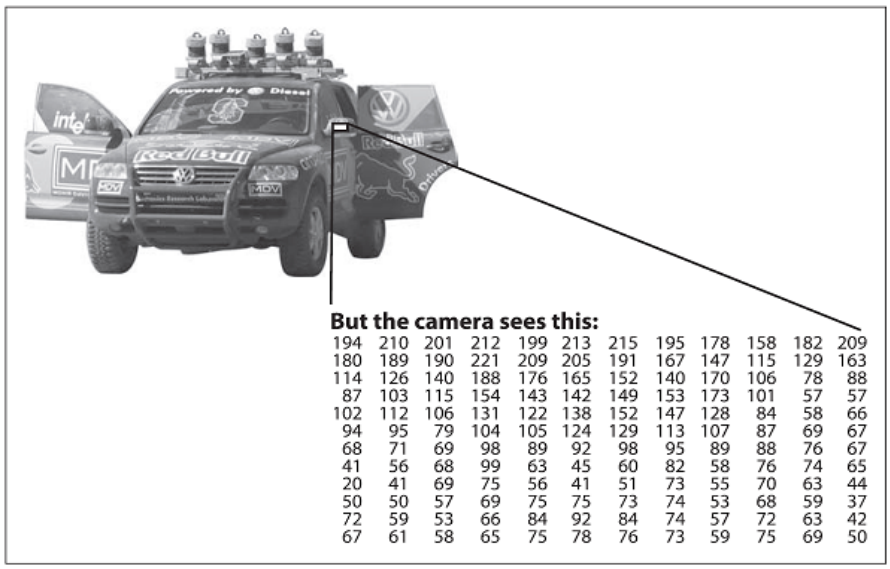

Abbildung 2: Autospiegel aus der Sicht eines Computers [OpenCV 2008]

Abb. 2 zeigt, warum Computer-Vision eine so anspruchsvolle Aufgabe ist bzw. warum ein so hoher Aufwand betrieben werden muss, um ein Kamera-Computer-System annähernd so effektiv wie das menschliche Auge zu gestalten.
Ein Mensch könnte problemlos die Reifen, Türen, Spiegel etc. identifizieren, während einem Computer zunächst nur eine schier unendliche Menge an Ganzzahlen präsentiert wird. Die Herkunft und Bedeutung dieser Darstellung wird in Kapitel 2.3 näher erläutert. Die Aufgabe der Computer-Vision ist es den Computer zu lehren, in dieser Masse von Zahlen einen Außenspiegel zu erkennen [OpenCV 2008].

2.2. Objekterkennung

„Unter *Objekterkennung* mit digitaler Bildverarbeitung versteht man, dass ein Bildobjekt anhand einer Menge charakteristischer Eigenschaften (Merkmale) vom Computer identifiziert und lokalisiert wird.“ [Linß 2010].

2.2.1. Anwendungen

Die Anwendungsbereiche sind dabei sowohl sehr zahlreich als auch vielfältig. Dazu zählen nicht nur alltägliche Anwendungen wie die Gesichtserkennung bei Digitalkameras oder das Durchleuchten der Koffer am Flughafen mit anschließendem Scannen des Inhalts auf verbotene Gegenstände, z.B. Waffen. Auch in der Medizin und Technik spielt Objekterkennung eine immer größere Rolle. So sind moderne Systeme dazu in der Lage, in hoch auflösenden Computertomographie-Bildern kleinste Anomalien zu detektieren, deren Ursache ein Tumor sein könnte. Ein weiteres großes Feld ist die Automobilbranche, in der immer häufiger kamerabasierte Fahrerassistenzsysteme zum Einsatz kommen, um z.B. die genutzte Fahrspur, andere Verkehrsteilnehmer oder Verkehrsschilder automatisch zu erkennen (vgl. Abb. 3) [Wikipedia].

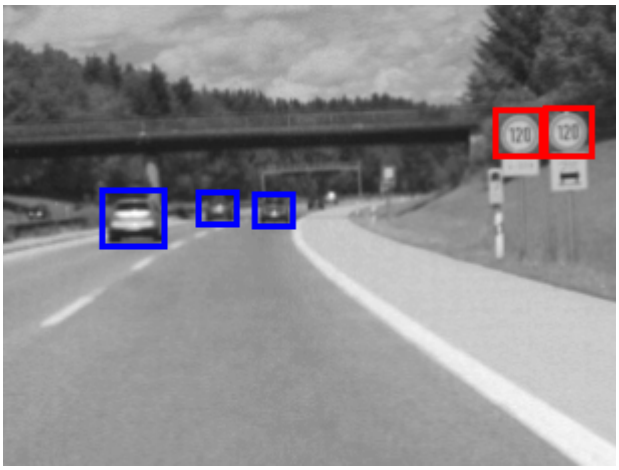

Abbildung 3: Objekterkennung in Fahrzeugen [Universität Bochum 2007]

2.2.2. Grundprinzip

Bei all diesen Anwendungen ähnelt sich der Ablauf. Am Anfang der computerbasierten Objekterkennung steht immer die Datensammlung. Es kommt ein Medium zum Einsatz, das dem Computer erlaubt, seine Umgebung bzw. den Ausschnitt seiner Umgebung, in dem die Objekterkennung erfolgen soll, abzuspeichern. Dabei kann es sich um eine handelsübliche Digitalkamera, aber auch um Radar- oder Röntgengeräte handeln. Während beim Menschen die Objekterkennung intuitiv erfolgt, benötigt ein Computer Algorithmen, um die vorliegenden Informationen zu verarbeiten und die zielführenden Informationen von den nicht zielführenden zu trennen. Anschließend werden die so gewonnenen Pixelinformationen mit bereits vorliegenden verglichen und es wird entschieden, ob die Ähnlichkeit ausreichend hoch ist, sodass ein „Treffer“, also eine Erkennung eines gesuchten Objektes, vorliegt [Lordemann 2003].

2.3. Farbräume

Im Hinblick auf die Thematik der Computer-Vision ist es vonnöten, eine gemeinsame, d.h. sowohl für den Menschen als auch für den Computer „verständliche“ Darstellung von Farben zu verwenden. Daher muss eine quantisierte Skala vorgegeben werden. Im Folgenden sollen einige der wichtigsten Modelle näher erläutert werden.

2.3.1. Das RGB-Modell

1850 entwickelte Hermann von Helmholtz die Drei-Farben-Theorie. Sie besagt, dass sich jede beliebige Farbe aus dem farbigen Licht der drei Primär-Farben Rot, Grün und Blau mischen lässt. Außerdem vermutete er, dass sich auch im menschlichen Auge drei Typen von Rezeptoren befinden, die unterschiedlich stark auf Licht verschiedener Wellenlänge reagieren. Die Existenz dieser Zäpfchen wurde mittlerweile nachgewiesen.
Basierend auf der Drei-Farben-Theorie wurde das einfache, aber wirkungsvolle RGB-Modell entwickelt. Dabei beschreibt der jeweilige Anteil der Primär-Farben die Intensität des jeweiligen Farbreizes. Während die klassische Darstellung dafür einen Wert zwischen 0 und 1 bzw. 0 bis 100 % verwendet, bietet sich gerade für die Arbeit mit Computern eine reine Integer-Skala zwischen 0 und einem Maximalwert an. Durch die verwendete Binärdarstellung in Computersystemen werden meist Wertebereiche gewählt, die sich als ganzzahlige Potenz zur Basis 2 darstellen lassen. Besonders häufige Verwendung finden die Bereiche 0 bis 7 (2^3) und 0 bis 255 (2^8). Die Anzahl der zur Kodierung der Intensität eines Farbwertes verwendeten Bits bezeichnet man als „Auflösung“. [Efford 2000]
Im Folgenden wird zur Darstellung von Farben im RGB-Modell die gängige Abkürzung „Rot-Wert“, „Grün-Wert“, „Blau-Wert“ verwendet. So wird z.B. eine Farbe, die sich aus dem Rot-Wert 50, dem Grün-Wert 100 und dem Blau-Wert 150 zusammensetzt, als „(50, 100, 150)“ angegeben.
Abb. 4 beschreibt schemenhaft den Farbraum des RGB-Modells mit einer Auflösung von 8 Bit pro Farbkanal. Es entspricht im Prinzip einem dreidimensionalen, kartesischen Koordinatensystem, bei dem der Nullpunkt in der linken, hinteren Ecke liegt. Dort ist die Intensität aller drei Primär-Farben null. Diese Abwesenheit aller

Farben entspricht einem tiefen Schwarz (0, 0, 0). Orthogonal zueinander angeordnet befinden sich die Achsen, die jeweils die Intensität einer der drei Primär-Farben angeben, d.h. Rot nach rechts, Grün nach oben und Blau nach vorne.

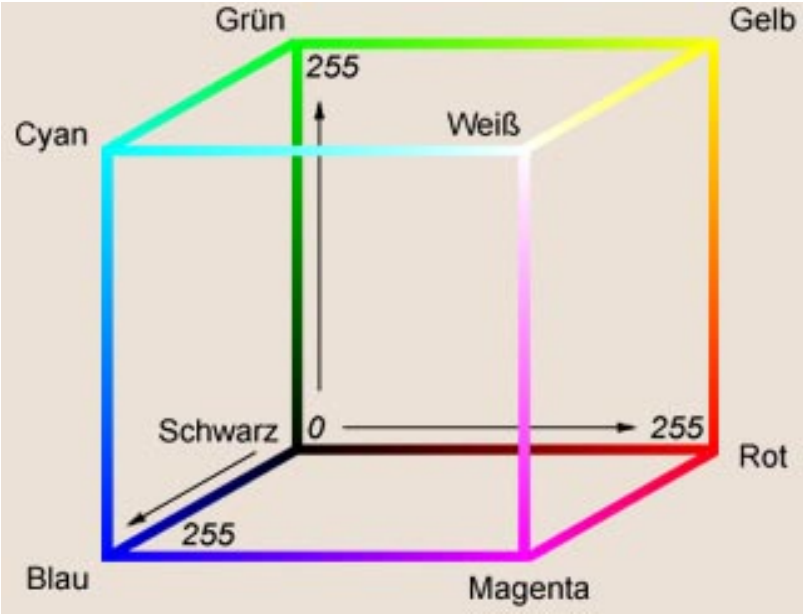

Abbildung 4: 3D-Darstellung des RGB-Modells [Universität München 2013]

Erhöht man vom Nullpunkt aus nur den Rotanteil, d.h. dieser wandert auf der Rot-Achse nach rechts, verändert sich die Gesamtfarbe immer mehr zum Roten, ausgehend von einem sehr dunklen Rot-Ton bis hin zu einem sehr kräftigen Rot (255, 0, 0). Eine Kombination zweier oder dreier Primär-Farben, d.h. die Verstärkung ihrer Intensität führt zu „neuen“ Farben. So ergibt z.B. die maximale Intensität von Rot und Grün unter der Abwesenheit von Blau (255, 255, 0) ein sattes Gelb. Treten alle drei Primär-Farben mit maximaler Intensität auf (255, 255, 255) erhält man Weiß. Das RGB-Modell existiert in verschiedensten Varianten. Das geläufigste ist das RGB888- bzw. RGB24-Format, das alle drei Farbkanäle gleichberechtigt mit einer Auflösung von 8 Bit darstellt. Da jeder Pixel sich durch 24 Bit, 8 pro Farbkanal, darstellen lässt, bezeichnet man Bilder, die dieses Modell benutzen als „24-Bit-Farbbilder“. Durch Kombination der drei Farbanteile lassen sich bei einer verwendeten Auflösung von 8 Bit pro Kanal über 16 Millionen (256^3 = 16.777.216) Einzelfarben darstellen.
Eine weitere häufig auftretende Variante ist das RGB565-Format. Hierbei werden die beiden Farbkanäle für Rot und Blau mit je 5 Bit kodiert, das entspricht einem Wert von 0 bis 31. Der Grün-Kanal jedoch wird mit 6 Bit, d.h. 0 bis 63, kodiert. Dies erklärt sich durch die erhöhte Empfindlichkeit des menschlichen Auges im Grün-Bereich. Es ist in der Lage, dort mehr Variationen wahrzunehmen als in den beiden anderen Bereichen. Der Hauptvorteil des RGB565-Formats liegt in seinem geringeren Speicherplatzbedarf. Allerdings können dementsprechend weniger Farben dargestellt werden (32*64*32 = 65536). Aufgrund der besseren Handhabbarkeit des RGB888-Formats wurde dieses in dieser Arbeit verwendet [Davies 2012] [Ebner 2007] [Forsyth 2012].

2.3.2. Graustufenbilder

Das eben vorgestellte RGB-Modell ist in der Lage, eine große Menge an Informationen darzustellen und zu speichern. Jedoch wird sich im weiteren Verlauf dieser Arbeit zeigen, dass die Aufspaltung der Information eines einzelnen Pixels in drei Teile, d.h. drei Farbkanäle, bei einigen Stufen der Weiterverarbeitung wie z.B. der Kantendetektion (s. Kapitel 2.6.) und Bildanalyse nicht praktikabel ist. Es ist nötig, eine Darstellung zu finden, die nur einen einzigen Integer-Wert verwendet, und dennoch genug Information enthält, um verschiedene Farben noch unterscheiden zu können. Hierfür verwendet man sogenannte „Graustufenbilder“.
Es gibt verschiedene Möglichkeiten, um die drei Farbkanäle R, G, B eines Pixels x zu einem Grauwert Gw zusammenzufassen. Drei der am häufigsten verwendeten und mit dem kleinsten Rechenaufwand verbundenen seien im Folgenden kurz dargestellt. Dabei entspricht z.B. der Ausdruck *x.R* dem Rotwert des Pixels *x*:

1) einfacher Mittelwert:

$$Gw(x)=\frac{x.R+x.G+x.B}{3} \tag{2.1}$$

2) Mittelwert aus Minimum und Maximum

$$Gw(x)=\frac{min(x.R, x.G, x.B)+max(x.R, x.G, x.B)}{2} \tag{2.2}$$

3) gewichteter Mittelwert

$$Gw(x)=0.3*x.R+0.59*x.G+0.11*x.B \tag{2.3}$$

Die Vorfaktoren beim gewichteten Mittelwert können variieren. Die hier aufgeführten Werte entstammen dem bekannten Grafikprogramm *GIMP*. Auffällig ist, dass der Vorfaktor des Grün-Wertes deutlich größer ist als die der beiden anderen. Dies ist wiederum auf die erhöhte Grün-Empfindlichkeit des menschlichen Auges zurückzuführen.
Bei vielen Bildern ist nur ein kleiner oder gar kein Unterschied zwischen Graustufenbildern zu erkennen, die mit verschiedenen Methoden erzeugt wurden. Oft, und so auch in dieser Arbeit, ist nur wichtig, dass sich Objekte im Vordergrund, z.B. Bälle, deutlich genug von ihrem Hintergrund abheben, um eine Analyse ihrer Form zu gewährleisten.
Die besondere Eigenschaft der Grauwerte ist, dass sie sich im oben vorgestellten RGB-Modell alle auf der Diagonale zwischen Schwarz (0, 0, 0) (im Graustufenbild Wert 0) und Weiß (255, 255, 255) (im Graustufenbild Wert 255) wiederfinden und sich somit ihre RGB-Werte nicht unterscheiden. Dies ermöglicht wiederum die Reduktion auf nur 8 Bit.
Eine Umkehr des Prozesses, also eine Rückrechnung von einem 8 Bit Graustufenbild in ein 24 Bit Farbbild, ist aufgrund des erlittenen Informationsverlusts nicht möglich.
Abb. 5 zeigt ein Farbbild dreier Bälle im RGB888-Format und die zugehörigen Graustufenbilder, berechnet nach den eben vorgestellten Methoden.

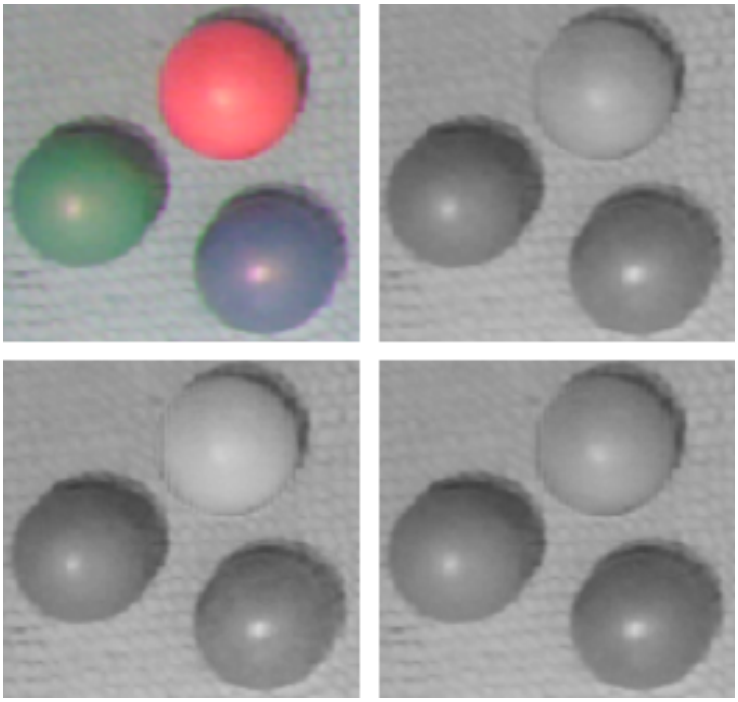

Abbildung 5: Mittels verschiedener Methoden erzeugte Graustufenbilder:
oben: Originalbild, Einfacher Mittelwert
unten: Mittelwert aus Minimum und Maximum, Gewichteter Mittelwert

	Roter Ball	Grüner Ball	Blauer Ball
Rotwert	148	42	41
Grünwert	65	89	65
Blauwert	74	74	74
Summe	287	205	180
Grauwert (einfacher Mittelwert)	96	68	60
Grauwert (Mittelwert aus Minimum und Maximum)	107	66	58
Grauwert (gewichteter Mittelwert)	91	73	59

Tabelle 1: Farbwerte und daraus resultierende Grauwerte dreier Bälle

Es lässt sich feststellen, dass trotz der deutlichen Reduktion des Informationsgehalts sich die Bälle zum Teil noch klar voneinander unterscheiden lassen. So fällt der Unterschied zwischen Rot und Grün/Blau noch recht deutlich aus. Allerdings bereitet das Unterscheiden von Grün und Blau große Schwierigkeiten. Dies lässt sich durch einen Blick auf die einzelnen Farbkanäle erklären (s. Tabelle 1).
Anzumerken ist, dass die Werte in der Tabelle den Farbwerten der am häufigsten vorkommenden Pixel innerhalb der Bälle entsprechen. Der bei allen drei Bällen identische Blau-Wert ist Zufall.
Es zeigt sich, dass der Rot-Wert des roten Balls deutlich höher liegt als der dominierende Wert der anderen beiden Bälle. Dies schlägt sich wiederum im

Graustufenbild nieder. Während der rote Ball deutlich heller erscheint, ist die Unterscheidung des ehemals grünen vom blauen Ball mit bloßem Auge kaum möglich [Burger 2009a] [Hermes 2005] [Johnson 2006].

2.3.3. Binärbilder

Die einfachste Form eines Graustufenbildes stellt das sogenannte Binärbild dar, dessen Pixel nur zwei Werte annehmen können: Schwarz (kodiert mit Wert 0) oder Weiß (Wert 1).
Zur Umwandlung eines Farb- oder Graustufenbildes in ein Binärbild gibt es keine festen Formeln oder Regeln. Vielmehr muss je nach Anwendung entschieden werden, welche Regeln bei der Umwandlung den meisten Sinn ergeben und somit das beste Ergebnis erzielen.
Eine mögliche Zielsetzung könnte sein, in dem bereits vorgestellten Bild oben links in Abb. 5 den roten Ball zu isolieren. Man würde das Bild Pixel für Pixel durchlaufen und dabei an die Stellen, die einen hohen Rot-Wert enthalten, eine 1 setzen, an alle anderen eine 0. Dabei spielt es keine Rolle, ob ein mit 0 markierter Pixel vorher blau, grün, schwarz oder weiß war. Die einzige im Binärbild gespeicherte Information ist, dass er nicht rot war. Eine mögliche Bedingung an die Pixel könnte sein, dass der Rot-Wert zwischen 120 und 180, der Grün-Wert zwischen 50 und 80 und der Blau-Wert zwischen 60 und 90 liegen muss. Erfüllt ein Pixel diese Vorgaben wird er durch 1 ersetzt, ansonsten durch 0. Somit wird jeder Pixel nur noch durch 1 Bit kodiert. Abb. 6 zeigt das Ergebnis einer solchen Umwandlung mit den eben erwähnten Parametern [Hermes 2005].

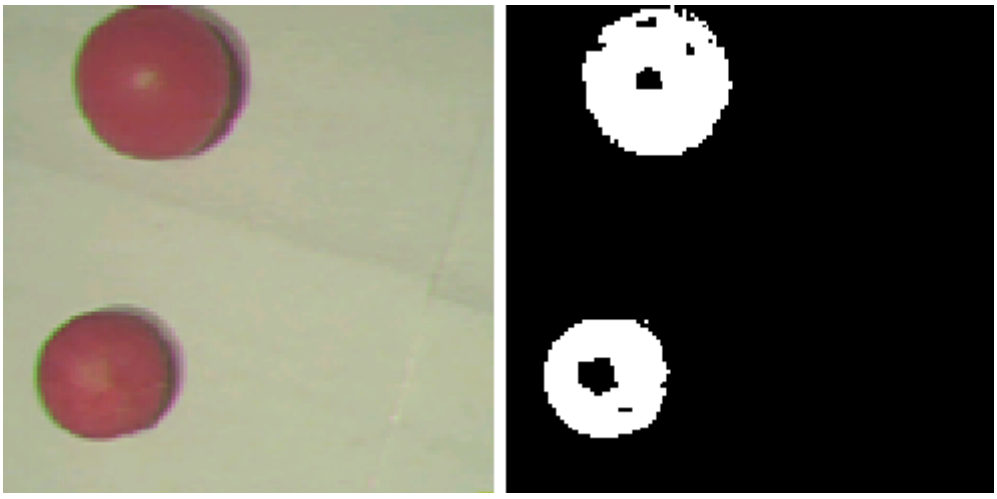

Abbildung 6: Binärbild zweier roter Bälle mit Minimum-Parametern (120, 50, 60) und Maximum-Parametern (180, 80, 90):
links: Originalbild
rechts: Binärbild

2.4. Histogramme

Histogramme sind vor allem für die Wahrscheinlichkeitsrechnung unabdingbar, da sie eine einfache, graphische Darstellung der Häufigkeitsverteilung skalierter Merkmale ermöglichen. Auch in der Bildverarbeitung spielen Histogramme eine wichtige Rolle.

2.4.1. Histogramme in Graustufenbildern

In einem typischen 8-Bit-Graustufenbild können die Pixel 256 verschiedene Intensitäten annehmen. Für jede dieser Intensitäten lässt sich die zugehörige Häufigkeit h(i) bestimmen. Dabei steht h(i) jeweils für die Anzahl der Pixel im Bild, die eine Intensität mit dem Wert i besitzen. h(0) gibt die Anzahl der Pixel mit dem Wert 0 an, h(255) die der Pixel mit dem Wert 255. Führt man dieses Abzählen für alle Intensitäten durch und trägt die Häufigkeit der einzelnen Intensitäten nebeneinander auf, ergibt sich ein Histogramm.
Dabei sei erwähnt, dass nicht zwingend jede einzelne Intensität separat dargestellt werden muss. Es ist z.B. möglich, je vier Intensitäten zu kombinieren und deren Häufigkeiten zu addieren. Dieses Vorgehen bezeichnet man als *Binning*. Der Hauptvorteil dieses Verfahrens liegt in der Reduzierung der Varianz der Verteilung einzelner Grauwerte.
Abb. 7 zeigt das aus Abb. 5 bekannte Graustufenbild der drei Bälle und das zugehörige Histogramm, erstellt mit dem FireFox-AddOn *Histogram Viewer*.

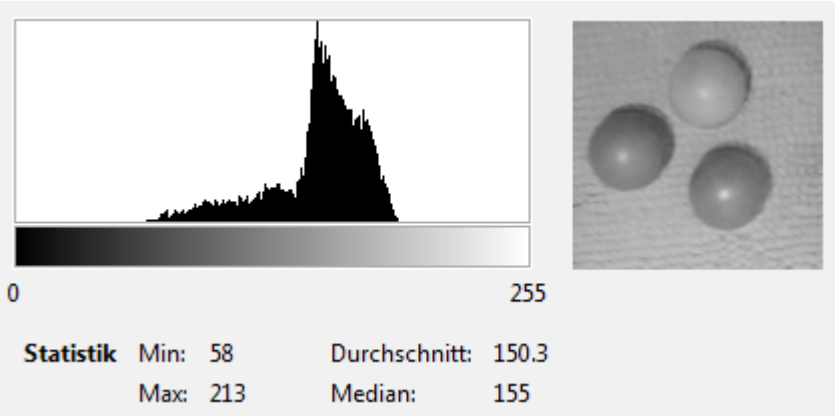

Abbildung 7: Histogramm eines Graustufenbildes

Sogleich werden die Vorteile der Histogramm-Darstellung deutlich. Auch ohne die unter „Statistik“ angegeben Minimum-, Maximum- und Durchschnittswerte der auftretenden Intensitäten kann sofort festgestellt werden, dass sich diese in einem Bereich um 150 häufen. Jedoch treten extreme Intensitäten nahe 0, sehr dunkles Schwarz bis hin zu dunklen Grau, oder 255, helles Grau bis hin zu Weiß, gar nicht auf.
Des Weiteren kann schnell die am häufigsten auftretende Intensität bestimmt werden. Dies ist mit dem bloßen Auge und obiger Darstellung nicht möglich, jedoch mit einer computerinternen Darstellung (z.B. Array) kein Problem [Burger 2009a] [Hermes 2005].

2.4.2. Histogramme in Farbbildern

Bei der Erstellung eines Farbbild-Histogramms stößt die einfache Technik des Abzählens der Häufigkeiten schnell an ihre Grenzen. Grund dafür ist die Darstellung jedes Pixels durch drei Werte, RGB. Die Lösung für dieses Problem ist die Aufteilung des Bildes in die drei Farbkanäle und das Anfertigen eines Histogramms für jeden von diesen nach dem beim Graustufenbild verwendeten Schema. In Abb. 8 wurde dies für das Originalbild der drei Bälle durchgeführt.

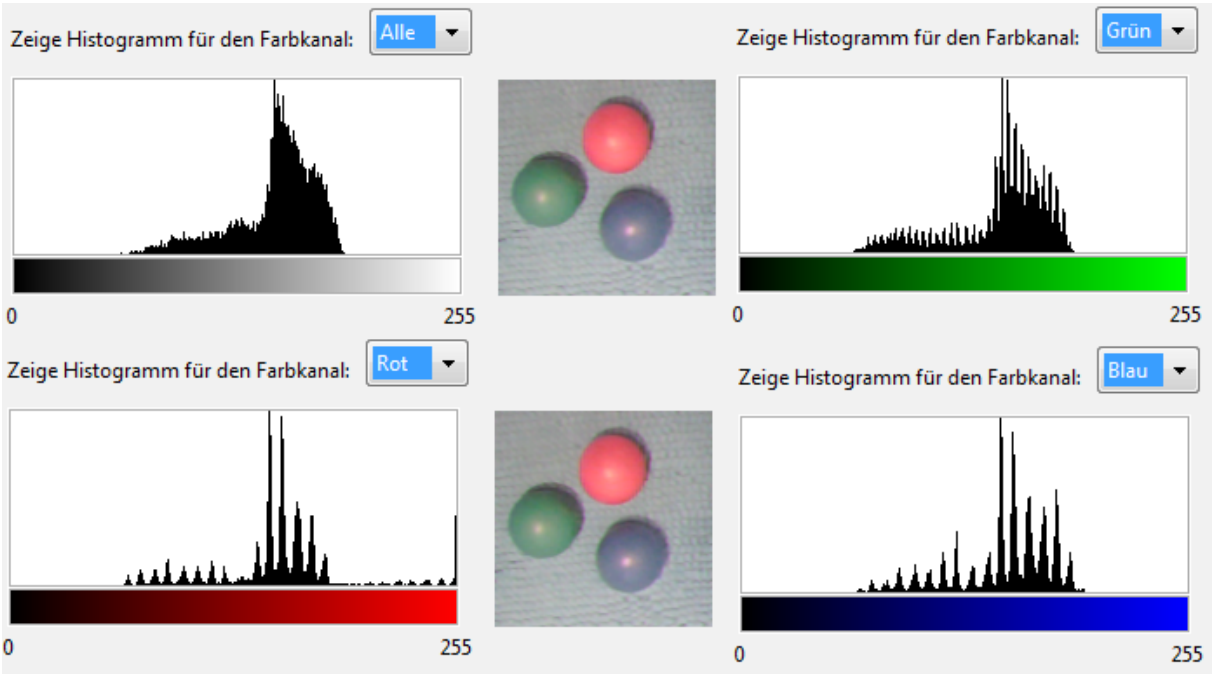

Abbildung 8: Histogramme eines Farbbildes dreier Bälle.

In dieser Darstellung lassen sich wiederum die groben Charakteristika des Bildes ablesen.
Auffällig ist das unter der Einstellung „Alle“ entstandene Histogramm, das alle drei Farbkanäle berücksichtigt und das vollends dem Graustufen-Histogramm entspricht. Hierbei spricht man von einem „Lumineszenz-Histogramm“ [Burger 2009a].

2.4.3. Grenzen der Histogramm-Darstellung

Das einfache Prinzip der Histogramme bringt auch Nachteile mit sich. So enthalten sie z.B. keinerlei Information über die Lage bzw. die Anordnung der Pixel. Besonders deutlich wird dies in Abb. 9. Alle drei Bilder ergeben exakt dasselbe Histogramm, da die Häufigkeitsverteilung der Intensitäten sich nicht unterscheidet. Die jeweilige Struktur der Bilder könnte allerdings unterschiedlicher nicht sein.

Abbildung 9: Drei sehr unterschiedliche Bilder mit identischen Histogrammen [Burger 2009a]

Eine weitere Schwäche tritt bei den Farbbild-Histogrammen auf. Es ist im Nachhinein nicht möglich, auch nur einen einzigen Pixel exakt zu beschreiben, da nicht bekannt ist, welche Werte der einzelnen Farbkanäle zusammengehören [Burger 2009a].

2.5. Filter

Durch die Beschreibung von Pixeln mittels des RGB-Modells ist es einem Computer möglich, einzelne Pixel zu vergleichen und zu unterscheiden. Allerdings liegt beim Arbeiten mit Bildern der Fokus meist nicht auf einzelnen Pixeln, sondern Gruppierungen ähnlicher Pixel, also auf zusammengehörigen Hintergründen, Flächen oder Objekten, oder deren Abgrenzungen bzw. Übergänge zueinander, d.h. Kanten. Hierbei stehen u.a. Filter-Operationen im Vordergrund, die sich immer auf eine ganze Gruppe von Pixeln beziehen.

2.5.1. Beispiel Blurring

Eine wichtige Filteroperation ist das sogenannte *Blurring*, d.h. das Weichzeichnen eines Bildes. Dabei entsteht, wie in Abb. 10 zu sehen, ein unschärferes Bild.

Abbildung 10: Beispiel Blurring:
links: Original-Bild
rechts: bearbeitetes Bild [Burger 2009a]

Dies sieht zunächst nach einer deutlichen qualitativen Verschlechterung aus, ist aber eine einfache, jedoch effektive Möglichkeit, ein Bild auf seine wesentlichen Informationen zu reduzieren. So bleiben starke Kontraste wie z.B. die Abgrenzungen von Objekten, hier der Bus, zu ihrem Hintergrund erhalten, während schwächere Kontraste, wie z.B. ungleichmäßige Oberflächen, verwaschen dargestellt und somit herausgefiltert werden. Dies kann bei der späteren Weiterverarbeitung des Bildes nützlich sein, da der Fokus der Algorithmen nun auf dem Wesentlichen liegt und somit Ergebnisse schneller erzielt werden können [Burger 2009a].

2.5.2. Die Filter-Matrix

Wie oben beschrieben, arbeiten Filter stets mit mehreren Pixeln. Meist werden Filter quadratischer Form und ungerader Seitenlänge verwendet, damit ein eindeutiger Mittelpunkt der Matrix existiert, der sogenannte *Hot Spot*. Die einfachste Filter-Matrix für den oben vorgestellten *Blurring*-Filter hat demnach folgende Form:

$$H(i,j) = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \tag{2.4}$$

Die Indizes i und j stehen für die Koordinaten der jeweiligen Pixel, die vom *Hot Spot* aus gezählt werden. Dies wird in Abb. 11 verdeutlicht.

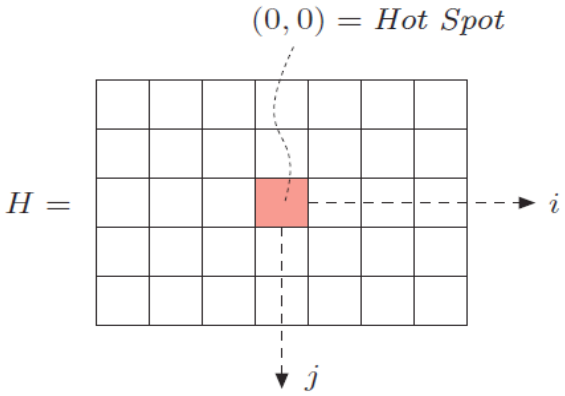

Abbildung 11: Koordinatensystem der Filter-Matrix [Burger 2009a]

Die Anwendung eines solchen Filters erfolgt in drei einfachen Schritten:

1) Die Filter-Matrix *H* wird Pixel für Pixel über das Originalbild I geschoben, sodass ihr *Hot Spot H(0, 0)* mit der aktuellen Bildposition *(u, v)* übereinstimmt.

2) Alle Filterkoeffizienten *H(i, j)* werden mit dem korrespondierenden Bildelement *I(u+i, v+j)* multipliziert und die Ergebnisse addiert.

3) Zuletzt wird die resultierende Summe an der aktuellen Position im neuen Bild *I'(u, v)* gespeichert.

Oder formal:

$$I'(u,v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) * H(i,j) \tag{2.5}$$

Im Folgenden wird diese Operation kurz als *I' = H(I)* angegeben.
Der gesamte Ablauf ist in Abb. 12 schematisch dargestellt.

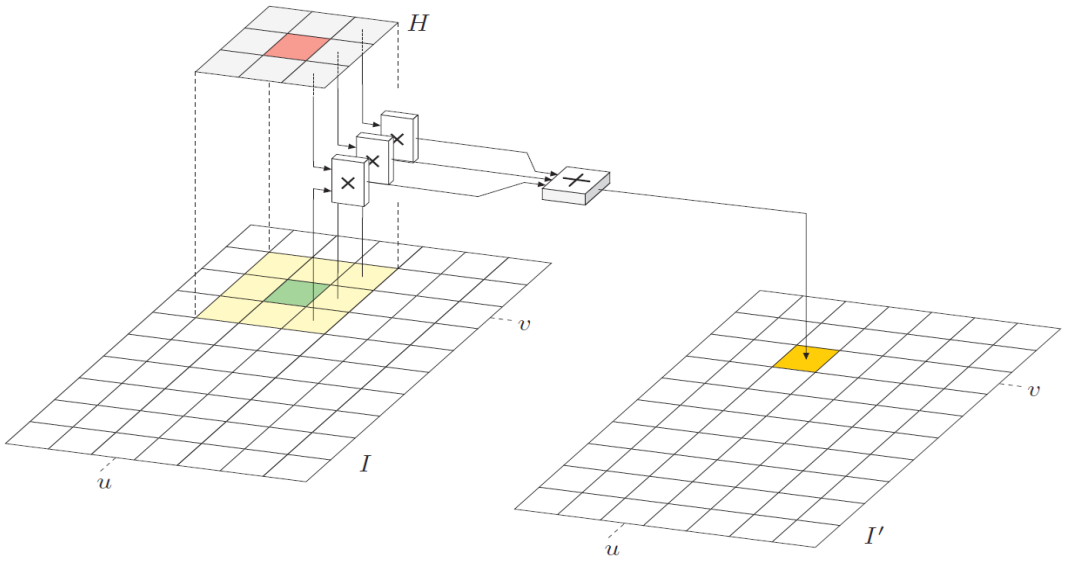

Abbildung 12: Anwendung Filter-Matrix [Burger 2009a]

Hierbei ist zu beachten, dass die neuen Bildwerte immer in ein neues Bild gespeichert werden müssen und nicht den alten Wert im Originalbild ersetzen können, da sonst die bereits gefilterten Werte wieder in die Berechnung der noch folgenden Filterwerte mit einbezogen werden würden [Burger 2009a] [Hermes 2005].

2.5.3. Weitere Filter-Matrizen

Bei der Filter-Matrix (Gl. 2.4) erhält jeder Pixel den Mittelwert aus seinem eigenen Pixelwert und den Werten seiner acht direkten Nachbarn zugewiesen. Es existieren noch viele weitere Filter-Matrizen. Es lässt sich zum einen die Größe fast beliebig variieren, z.B. durch Variieren der Zeilen- und Spaltenanzahl der Matrix auf fünf, sieben usw.. Zum anderen ist es möglich, die Gewichtungen der einzelnen Pixel untereinander zu verändern:

$$H(i,j) = \begin{pmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & 0.200 & 0.125 \\ 0.075 & 0.125 & 0.075 \end{pmatrix} \tag{2.6}$$

Hierbei fallen der *Hot Spot* und dessen direkte horizontalen und vertikalen Nachbarn stärker ins Gewicht als die Eck-Pixel. Anstatt der Verwendung von Gleitkommazahlen bietet sich die Verwendung eines inversen, ganzzahligen Vorfaktors an. Dadurch lässt sich das Innere der Matrix als Integer-Werte darstellen:

$$H(i,j)=\frac{1}{40}\begin{pmatrix}3 & 5 & 3\\ 5 & 8 & 5\\ 3 & 5 & 3\end{pmatrix} \tag{2.7}$$

oder allgemein:

$$H(i,j)=\frac{1}{n}\begin{pmatrix}g_{00} & g_{01} & g_{02}\\ g_{10} & g_{11} & g_{12}\\ g_{20} & g_{21} & g_{22}\end{pmatrix} \tag{2.8}$$

wobei

$$n=\sum_{i=-1}^{i=1}\sum_{j=-1}^{j=1} g_{ij} \tag{2.9}$$

Selbiges gilt auch für die Matrix aus (Gl. 2.4):

$$H(i,j)=\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \tag{2.10}$$

Alle bisher erwähnten Filter zählen zu den sogenannten Linearen Filtern. Sie zeichnen sich dadurch aus, dass die untersuchten Pixel keinerlei Einfluss auf die Filter-Matrix haben, sich also die Vorfaktoren und Werte innerhalb der Matrix nicht ändern.
Dem gegenüber stehen die Nicht-Linearen Filter, deren prinzipielle Vorgehensweise sehr der der linearen Filter ähnelt. Zur Berechnung des neuen Wertes werden für einen Pixel *I(i, j)* dessen Nachbar-Pixel herangezogen. Der einzige Unterschied besteht in der Berechnung des neuen Pixel-Werts *I′(i, j)*. Dazu seien drei häufige Vorgehensweisen als Beispiel gegeben:

1) Minimum-Filter: Setzt als neuen Wert *I′(i, j)* den kleinsten in der betrachteten Umgebung vorkommenden Pixelwert.

2) Maximum-Filter: Setzt als neuen Wert *I′(i, j)* den größten in der betrachteten Umgebung vorkommenden Pixelwert.

3) Median-Filter: Setzt als neuen Wert *I′(i, j)* den Median der in der betrachteten Umgebung vorkommenden Pixelwerte.

Abb. 13 zeigt die Auswirkungen des Minimum- und Maximum-Filters auf ein durch „Salz- und Pfeffer-Rauschen“ gestörtes Bild. Links befindet sich das Originalbild. In der Mitte ist das mit einem Minimum-Filter bearbeitete Bild zu sehen. Dieses erscheint insgesamt dunkler und hat sämtliche weißen Stör-Punkte verloren. Ebenso haben darin die schwarzen Stör-Punkte an Fläche gewonnen. Rechts dagegen sieht man das mit dem Maximum-Filter bearbeitete Bild.

Abbildung 13: Wirkungsweise eines Minimum- und Maximum-Filters:
links: Originalbild
Mitte: Minimum-Filter
rechts: Maximum-Filter [Burger 2009a]

Der Median-Filter eignet sich sehr gut, um Bilder von fehlerhaften Pixeln zu befreien. Deutlich wird dies in Abb. 14. Während ein linearer Filter die Stör-Pixel aus dem Originalbild (links) lediglich abschwächen kann (Mitte), so ist es dem Median-Filter (rechts) sogar möglich, sie für das menschliche Auge komplett unsichtbar zu machen [Burger 2009a] [Hermes 2005].

Abbildung 14: Wirkungsweise eines Median-Filters:
links: Originalbild
Mitte: Linearer Filter
rechts: Median-Filter [Burger 2009a]

2.5.4. Problem mit dem Wirkungsbereich eines Filters

Beim Filtern eines Bildes wandert die Filter-Matrix über alle Pixel. Deren *Hot Spot* liegt jeweils über dem zu ändernden Pixel. Der *Hot Spot* befindet sich mittig im Inneren der Filter-Matrix. Liegt dieser an den Rändern des Bildes, kann dies dazu

führen, dass sich Teile der Filter-Matrix außerhalb des Bildes bzw. des Speicherbereichs, in dem die Bilddaten liegen, befinden:
Eine häufig verwendete Repräsentation eines Bildes in Computersystemen ist das Speichern der einzelnen Pixel in einer Matrix der Größe *(height, width). height* entspricht dabei der Pixelanzahl des Bildes in vertikaler, *width* der in horizontaler Richtung. Referenziert werden dabei die einzelnen Pixel durch ihre jeweiligen Koordinaten *(i, k)*, die im Intervall zwischen 0 und *width* bzw. *height* liegen. Befindet sich der *Hot Spot* einer 3x3-Matrix z.B. über dem Pixel *(0, 0)*, liegen fünf der acht für die Berechnung nötigen Pixelwerte außerhalb des Matrixbereiches. Betroffen davon sind die Pixel mit den Koordinaten *(1, -1), (0, -1), (-1, -1), (-1, 0), (-1, 1)*. In einem Computerprogramm würde dies zu einem Absturz führen.
Abb. 15 zeigt das auftretende Problem. Der Filter kann nur dort angewendet werden, wo die Filter-Matrix *H* der Größe *(2K + 1) x (2L + 1)* komplett innerhalb des Bildes, also im inneren Rechteck, liegt.

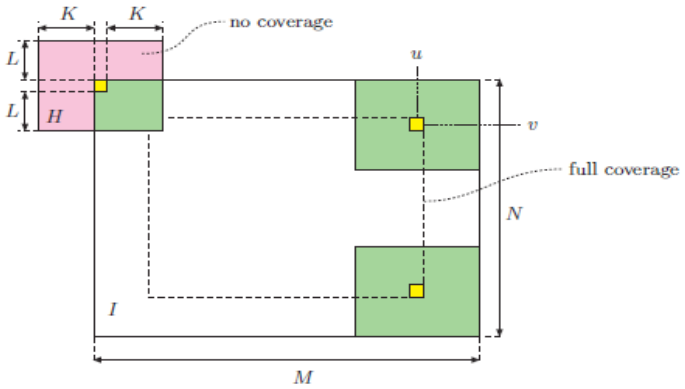

Abbildung 15: Teile der Filter-Matrix außerhalb der Bilddaten [Burger 2009a]

Dieses Problem lässt sich, je nach Anwendung, auf verschiedenste Art und Weise umgehen. Oft ist es möglich, die äußersten Bildbereiche zu ignorieren und den Pixeln im resultierenden Bild *I'* einfach die Werte ihrer Ebenbilder aus Originalbild *I* zuzuweisen. Bei anderen Anwendungen, beispielsweise bei der später vorgestellten Kantendetektion, bietet es sich an, die nicht zu erfassenden Pixel auf einen Fix-Wert, meist 0, d.h. Schwarz, zu setzen. Eine weitere Möglichkeit ist es, den Pixeln, die außerhalb der Filter-Matrix liegen, den Mittelwert der innerhalb liegenden Pixel zuzuweisen und die Berechnung auf gewohnte Weise durchzuführen [Burger 2009a].

2.6. Kantendetektion

Neben der Farbe ist die Form das wichtigste Merkmal, das ein Objekt auszeichnet. Bei der Beschreibung der Form eines Objektes stehen Kanten im Vordergrund. Hier handelt es sich um Bereiche im Bild, bei denen sich die nebeneinander liegenden Pixel-Werte stärker unterscheiden als dies bei einheitlichen Oberflächen der Fall ist.

2.6.1. Grundprinzip der Kantenerkennung

Um diese Änderung aneinander angrenzender Pixel hervorzuheben, verwendet man spezielle Filter-Matrizen, im Folgenden auch Kanten-Operatoren genannt. Dabei ist die prinzipielle Vorgehensweise gleich den im vorherigen Kapitel vorgestellten Filter-Operationen.
Man betrachte den Ausschnitt *P* eines Graustufenbildes mit einer vertikal verlaufenden Kante:

$$P=\begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & \mathbf{50} & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & \mathbf{50} & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Um diese Kante hervorzuheben, wird ein einfacher Laplace-Operator *L* verwendet:

$$L=\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{2.11}$$

Dieser kommt stellvertretend für alle anderen Pixel an den beiden fett markierten *Hot Spots* zum Einsatz: Oben im Bereich einer durchgängigen Fläche, unten an einer Kante.

$$L(P)=\begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 50_{0} & 50_{-1} & 50_{0} & 150 & 150 & 150 & \dots \\ \dots & 50_{-1} & \mathbf{50_{4}} & 50_{-1} & 150 & 150 & 150 & \dots \\ \dots & 50_{0} & 50_{-1} & 50_{0} & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50_{0} & 50_{-1} & 150_{0} & 150 & 150 & \dots \\ \dots & 50 & 50_{-1} & \mathbf{50_{4}} & 150_{-1} & 150 & 150 & \dots \\ \dots & 50 & 50_{0} & 50_{-1} & 150_{0} & 150 & 150 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Für obigen *Hot Spot* ergibt sich der Wert 0. Dies ist eine wichtige Eigenschaft der Kantenerkennung und bei allen Kanten-Operatoren der Fall, wenn sie auf einer einheitlichen Fläche angewendet werden. Für den unten gewählten Kanten-*Hot Spot* ergibt sich dagegen der Wert -100. Da negative Graustufenwerte nicht definiert sind, wird bei der Kantendetektion der Betrag des jeweiligen Ergebnisses verwendet. Dies ändert am Ergebnis nichts, da ausschließlich die Differenz der an der Kante liegenden Pixel entscheidend ist und nicht, ob man sich links oder rechts der Kante befindet. Angewendet auf alle Pixel ergibt sich folgendes Bild, das einer deutlich

helleren, senkrechten Kante auf schwarzem Grund entspricht.

$$P' = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

[Burger 2009a] [Hermes 2005].

Abb. 16 zeigt für den Vorgang der Kantendetektion die zu den Ausschnitten gehörigen Bilder.

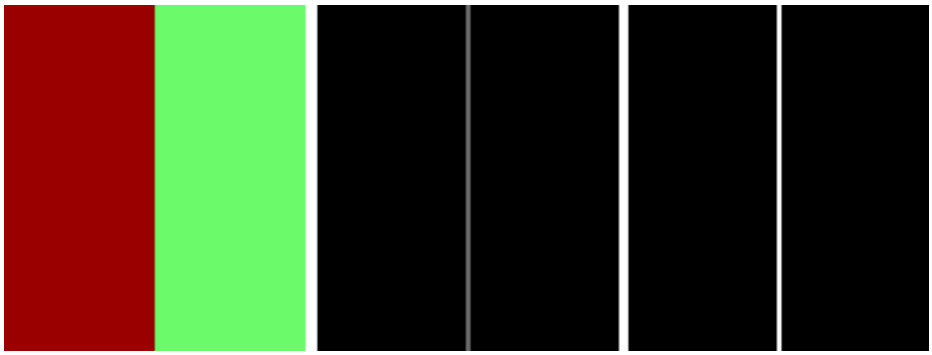

Abbildung 16: Vorgang der Kantendetektion:
links: Originalbild (rot: (150, 0, 0), grün: (100, 250, 100))
Mitte: Detektierte Kante (100, 100, 100)
rechts: Binärdarstellung

2.6.2. Kanten-Operatoren

In der Praxis findet der Laplace-Operator bei der Kantendetektion aufgrund seiner extremen Anfälligkeit für Rauschen nur selten Verwendung. Deswegen bietet es sich an, zwei unterschiedliche Operatoren, je einen für das Hervorheben horizontaler bzw. vertikaler Kanten, zu verwenden und deren Ergebnisse anschließend zu kombinieren. Häufige Verwendung finden dabei Sobel- und Prewitt-Operatoren:

Sobel-Operatoren:

$$H_x^S = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ und } H_y^S = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{2.12}$$

Prewitt-Operatoren:

$$H_x^P = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \text{und} \quad H_y^S = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{2.13}$$

An den Matrizen lässt sich ablesen, dass bei den Sobel-Operatoren die Differenzen der Pixel, die direkt am *Hot Spot* anliegen, doppelt so stark gewichtet werden wie die der schräg anliegenden.
Anschließend werden die beiden so aus dem Bild *I* erzeugten Matrizen D_x und D_y kombiniert, um die Gesamtintensität *E* des jeweiligen Kantenpixels *(u, v)* zu berechnen:

$$E(u,v) = \sqrt{(D_x(u,v))^2 + (D_y(u,v))^2} \tag{2.14}$$

Diese wird auch als Gradient bezeichnet.
Des Weiteren lässt sich die lokale Richtung *Φ* jedes Kantenpixels *(u, v)* bestimmen:

$$\phi(u,v) = \tan^{-1}\left(\frac{D_y(u,v)}{D_x(u,v)}\right) \tag{2.15}$$

Dabei beschreibt der Wert *Φ = 0* eine vertikale Kante. Positive Werte entsprechen einer Drehung gegen den Uhrzeigersinn.
Abb. 17 zeigt den prinzipiellen Ablauf der Kantendetektion, den der Kombination der Teil-Matrizen und den der Richtungsberechnung.

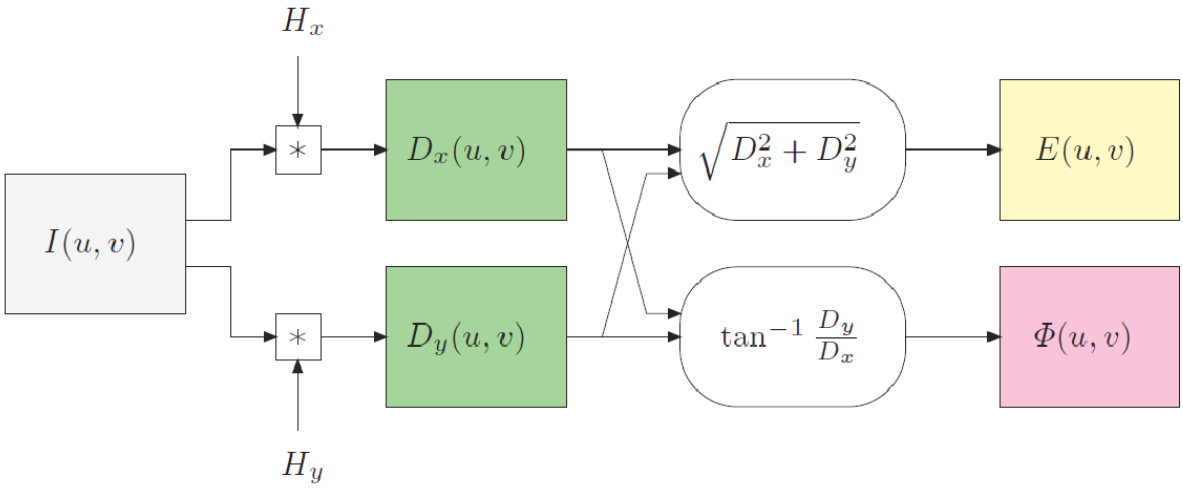

Abbildung 17: Übersicht über den Ablauf Kantendetektion [Burger 2009a]

Die Ergebnisse der beiden vorgestellten Operatoren unterscheiden sich meist, abhängig von der Auflösung des Bildes und der Schärfe der Kanten, nur wenig. Allerdings lässt sich in Abb. 18 eine leicht deutlichere Darstellung der Kanten der Bälle in dem mit Hilfe der Sobel-Operatoren erstellten Bild rechts unten erkennen [Burger 2009a] [Davies 2012] [Hermes 2005].

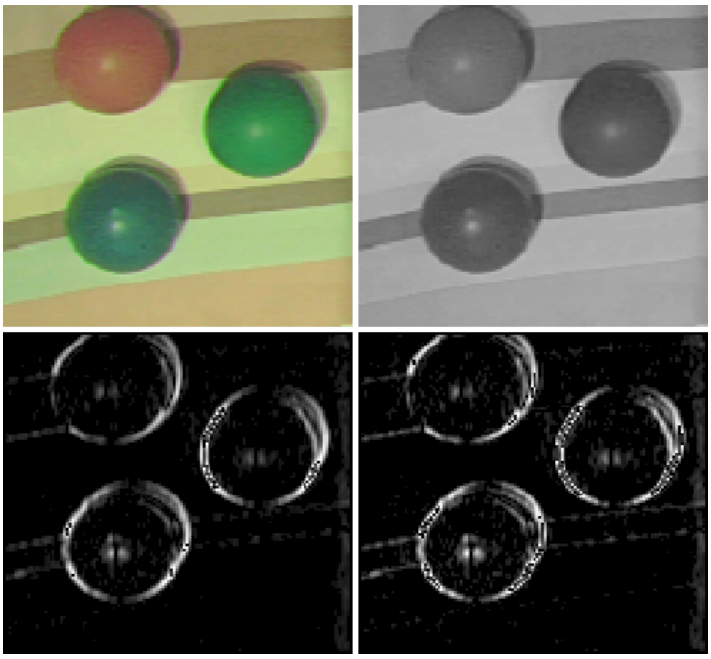

Abbildung 18: Vergleich von Sobel- und Prewitt-Operator:
oben: Originalbild, Graustufenbild
unten: Prewitt-Operator, Sobel-Operator

2.6.3. Spezialfall Canny-Kantendetektion

Speziell als Vorverarbeitungsschritt zur Kreisdetektion unter Verwendung der Hough-Gradienten-Methode (s. 2.7.5.) findet häufig der Canny-Operator Verwendung. Dieser versucht nicht nur alle Kantenpunkte innerhalb eines Bildes zu finden, sondern diese auch zu einer Kontur zu vereinen. Dazu werden zwei Grenzwerte benötigt: ein Oberer und ein Unterer. Der Wert des Unteren entspricht meist der Hälfte des Wertes des Oberen. Liegt der Gradient eines Pixels über dem oberen Grenzwert, wird dieser als Kantenpixel gespeichert. Bei einem Gradienten kleiner als der untere Grenzwert wird der Pixel abgewiesen. Liegt der Gradient zwischen beiden Grenzen, wird er dann als Kantenpixel behandelt, wenn einer seiner direkten Nachbarn über einen Gradienten verfügt, der über dem oberen Grenzwert liegt [Davies 2012] [OpenCV 2008].

2.7. Hough-Kreis-Transformation

Nach dem Hervorheben der Kanten und der Ausblendung von durchgängigen Flächen soll nun die Identifikation der zu den Kanten gehörigen Objekte im Vordergrund stehen. In dieser Arbeit wird dabei der Fokus auf Kreisen liegen. Betrachtet man Abb. 18 wird schnell deutlich, dass es für einen Menschen kein Problem darstellt, alle drei vorhandenen Kreise sofort zu erkennen. Im Nachfolgenden soll ein Algorithmus vorgestellt werden, der einem Computer dasselbe ermöglicht.

2.7.1. Grundlagen

Um einen Kreis in der Ebene eindeutig zu beschreiben, werden drei Parameter benötigt: Die kartesischen Koordinaten des Kreismittelpunktes *M(x_M, y_M)* und der Kreis-Radius r (vgl. Abb. 19).

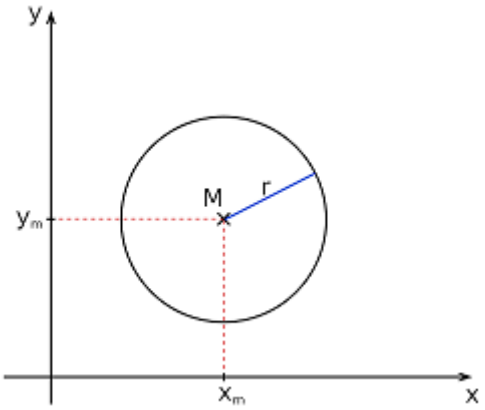

Abbildung 19: Kreis-Parameter [Wikipedia]

Des Weiteren erfüllt jeder Punkt *(x, y),* der sich auf dem Kreis befindet, die Kreisgleichung:

$$(x-x_M)^2+(y-y_M)^2=r^2 \tag{2.16}$$

An jedem dieser Punkte schneidet, wie in Abb. 20 zu sehen, ein Kreis mit dem Radius *r* um diesen Punkt den ursprünglichen Kreismittelpunkt [Burger 2009b] [Rhody 2005].

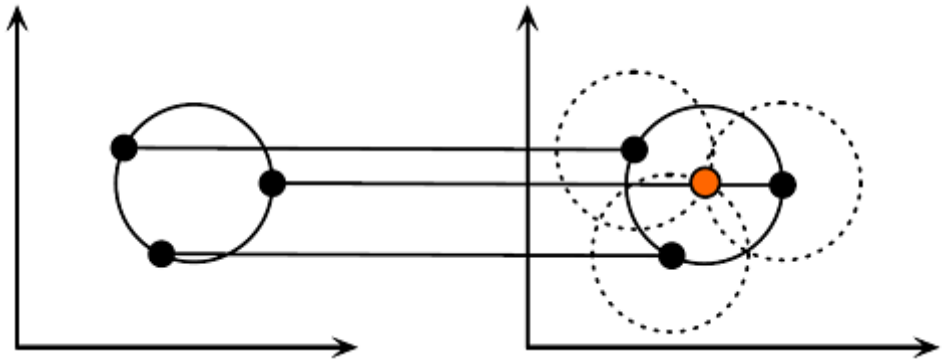

Abbildung 20: Kreis-Schnittpunkte [Rhody 2005]

2.7.2. Transformation in den Hough-Raum

Die Hough-Kreis-Transformation macht sich die oben genannten Eigenschaften wie folgt zunutze: Angewendet auf ein Binärbild, in dem eine 1 einen Kantenpixel und eine 0 keinen Kantenpixel darstellt, werden um jeden Kantenpixel (x_E, y_E) Kreise mit den gesuchten Radien erzeugt. Diese Kreise schneiden sich, wie in Abb. 21 zu sehen, in zahlreichen Punkten sowohl innerhalb als auch außerhalb des

ursprünglichen Kreises. Auffällig ist, dass sich die Schnittpunkte von Kreisen mit einem falschen bzw. nicht gesuchten Radius weitreichend verteilen. Dem gegenüber ergibt sich eine starke Häufung der Schnittpunkte im Inneren des Kreises, je näher man dem gesuchten Radius kommt. Im Optimalfall, also mit ideal gewähltem Radius, schneiden sich alle Kreise in genau einem Punkt, wie in Abb. 20 zu sehen ist.

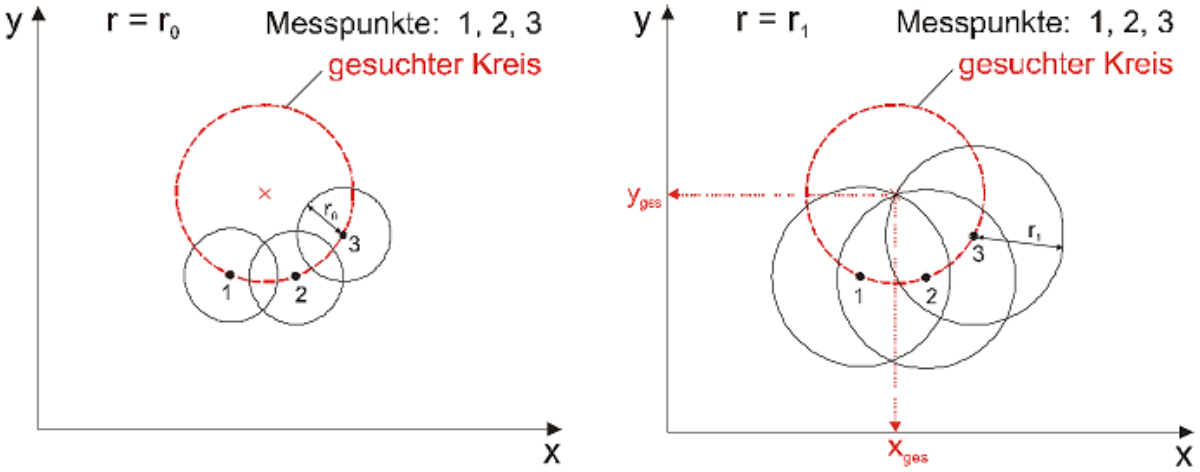

Abbildung 21: Hough-Ebene zweier untersuchter Radien:
links: zu kleiner Radius
rechts: gesuchter Radius [Linß 2008]

Das Ziel der Transformation in den Hough-Raum ist es, genau diesen Punkt zu finden. Da in der Praxis meist weniger idealisierte Bedingungen herrschen, wird sich darauf beschränkt, den Punkt zu finden, in dem sich die größte Anzahl an Kreisen schneidet.
Umgesetzt wird dies mit Hilfe eines Akkumulator-Raums. Darunter versteht man einen dreidimensionalen Raum *H(x, y, r)*, dessen *x*-*y*-Ebenen die Dimension des zu untersuchenden Bildes besitzen.

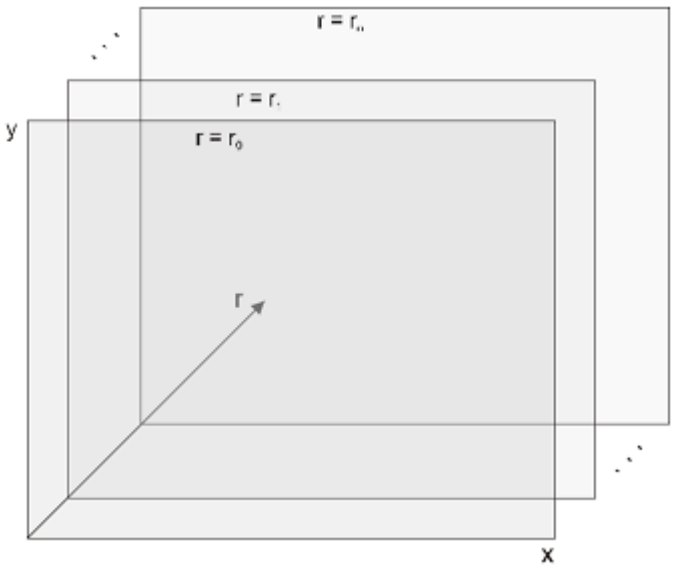

Abbildung 22: Hough-Akkumulator-Raum H(x, y, r) mit n+1 (r_0 bis r_n) x-y-Ebenen [Burger 2009a]

Die Anzahl dieser Ebenen ist gleich der Anzahl der zu untersuchenden Radien. Jeder dieser Punkte wird mit dem Wert 0 initialisiert. Abb. 22 zeigt einen Hough-Akkumulator-Raum der Dimension *(x, y, r)*.
Bei jedem Kantenpixel (x_E, y_E) wird für jeden Punkt *(x, y, r)* überprüft, ob er die Kreisgleichung mit Mittelpunkt (x_E, y_E) erfüllt. Ist dies der Fall, wird der Wert der jeweiligen Zelle um 1 erhöht. Dieser Vorgang sei noch einmal anhand von Abb. 21 näher dargestellt:
Rot eingezeichnet ist der gesuchte Kreis *(x_{ges}, y_{ges}, r_{ges})*, auf dem sich die Kantenpixel 1, 2 und 3 befinden. Von diesen Punkten aus werden zwei Radien, r_0 und r_1, mittels der Kreisgleichung getestet. In diesem Fall hätte der zugehörige Akkumulator-Raum demnach zwei x-y-Ebenen, die die beiden Radien repräsentieren. An allen Koordinaten, durch die die schwarzen Kreise laufen, die also potentielle Kreismittelpunkte darstellen, werden die Werte im Akkumulator-Raum um 1 angehoben. Dabei wird jeweils nur die den x-y-Koordinaten des Kreispunktes entsprechende Zelle erhöht, die sich in der zum getesteten Radius gehörenden Ebene befindet [Burger 2009b] [Davies 2012] [Linß 2010].

2.7.3. Identifizierung im Hough-Raum

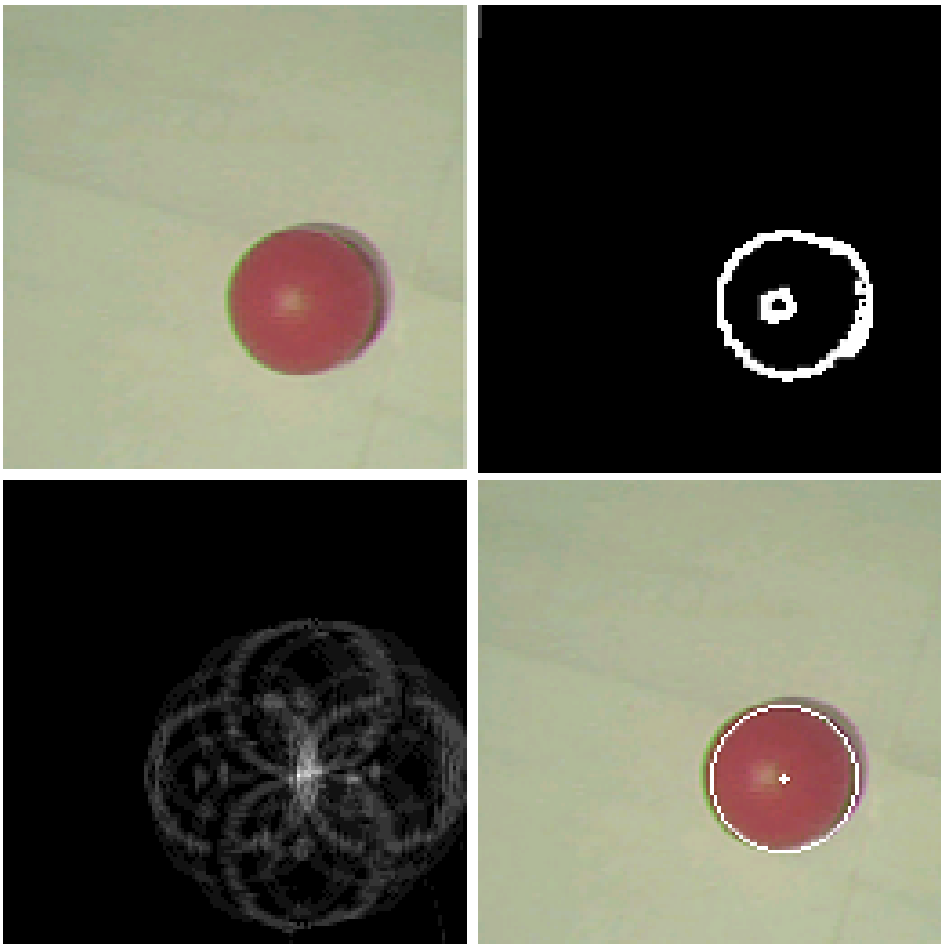

Abbildung 23: Nutzung des Akkumulator-Raums zur Kreisdetektion:
oben: Original-Bild, Ergebnis der Kantendetektion
unten: Akkumulator-Array für gesuchten Radius, Detektierter Kreis

Nach der Transformation in den Hough-Raum müssen die Parameter des Kreises, der in den gegeben Bilddaten am wahrscheinlichsten erscheint, aus dem Akkumulator-Raum ausgelesen werden. Diese entsprechen den Koordinaten des höchsten abgespeicherten Wertes, d.h. dem Punkt, in dem sich die meisten Kreise mit einem bestimmten Radius um Eckpunkte schneiden. Dieser Vorgang ist in Abb. 23 graphisch dargestellt. Je heller ein Punkt ist, desto höher ist der Wert an dieser Stelle im Akkumulator-Array [Burger 2009b] [Davies 2012] [Linß 2010].

2.7.4. Vor- und Nachteile der Hough-Kreis-Transformation

Vorteile:

Es handelt sich um ein robustes Verfahren. Das zeigt sich vor allem durch:

1) Eine hohe Resistenz gegen Rauschen. Pixelfehler und falsche Kanten beeinträchtigen das Ergebnis nur bedingt und meist ist eine Kreisdetektion trotzdem noch möglich. Abb. 24 zeigt eine solche Kreisdetektion trotz vieler Stör-Kanten.
2) Die Robustheit gegenüber fehlenden Kanten, z.B. durch eine teilweise Verdeckung des gesuchten Objektes, wie in Abb. 25 zu sehen.

Abbildung 24: Kreisdetektion in einem Bild mit vielen Stör-Kanten:
links: Original-Bild
Mitte: Ergebnis der Kantendetektion
rechts: Detektierter Kreis

Abbildung 25: Kreisdetektion bei nur teilweise sichtbaren Kreis:
links: Original-Bild
Mitte: Ergebnis der Kantendetektion
rechts: Detektierter Kreis

Nachteile:

1) Die Hough-Kreis-Transformation entspricht größtenteils einem Brute-Force-Ansatz und ist somit sehr rechenaufwändig.

2) Der Speicherplatzbedarf ist, gerade bei Bildern mit hoher Auflösung und vielen in Frage kommenden Radien, extrem groß.

3) Bei der Standard-Implementierung wird, sobald auch nur ein einziger Kantenpixel vorhanden ist, vom Algorithmus immer ein Kreis gefunden. Es ist oft nötig, zusätzliche Anforderungen an gefundene Kreise zu stellen, um diese sicher als Kreise klassifizieren zu können. Auf diese Thematik wird in Abschnitt 4.2.5. noch genauer eingegangen [Wikipedia].

2.7.5. Die Hough-Gradienten-Methode

Es gibt eine Weiterentwicklung der eben vorgestellten Hough-Kreis-Transformation, die sowohl die notwendige Rechenleistung als auch den benötigten Speicherplatz deutlich reduziert. Diese sogenannte Hough-Gradienten-Methode verringert dabei, unter Berücksichtigung der Kantenrichtungen, die Anzahl der möglichen Kreismittelpunkte für jeden Kantenpixel erheblich. Des Weiteren wird auf die Verwendung eines dreidimensionalen Akkumulator-Raums verzichtet und lediglich ein Akkumulator-Array eingesetzt.

Ablauf:

Zunächst durchläuft das zu untersuchende Graustufen-Bild eine Kantendetektion. Hierbei kommt der Canny-Algorithmus zum Einsatz.
Anschließend wird für jeden Kantenpunkt die Richtung der Kante berechnet. Diese entspricht der Richtung der an den gesuchten Kreis angelegten Tangente und ist somit orthogonal zu der Verbindungslinie zwischen Kreismittelpunkt und Kantenpunkt. Mit dem Wissen, dass sich der gesuchte Mittelpunkt auf dieser Linie befinden muss, wird die Erhöhung der Akkumulator-Werte durchgeführt. Dabei werden, nur entlang dieser Linie, die Werte der Zellen angehoben, die sich zwischen dem minimalen und maximalen gesuchten Radius befinden. Abb. 26 zeigt diesen Ablauf für einen Kantenpunkt am linken Rand des gesuchten Kreises. Die Richtung der Kante verläuft vertikal und steht senkrecht auf der gestrichelten Verbindungslinie zwischen Kanten- und Mittelpunkt. Auf dieser Linie befinden sich die die möglichen Mittelpunkte im Abstand vom minimalen Radius r_{min}, über den gesuchten Radius $r_{actual,}$ bis hin zum maximalen Radius r_{max}.
Zusätzlich werden während dieses Durchlaufs die Koordinaten aller Kantenpunkte gespeichert. Danach erfolgt die Auswahl möglicher Mittelpunkte. Dabei werden die Punkte in Betracht gezogen, deren Akkumulator-Wert über einem bestimmten Grenzwert liegt und höher ist als der seiner direkten Nachbarn. Die so gefundenen Punkte werden in absteigender Reihenfolge, entsprechend ihres Akkumulator-Wertes, sortiert.

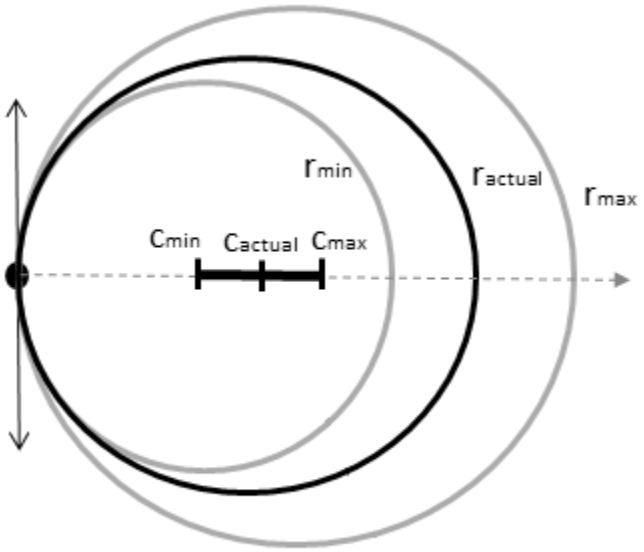

Abbildung 26: Hough-Gradienten-Methode zur Kreisdetektion [MathWorks 2012]

Als Nächstes wird für jeden möglichen Mittelpunkt die Liste der Kantenpunkte betrachtet und diese, entsprechend ihres Abstandes zum jeweiligen möglichen Mittelpunkt, aufsteigend sortiert. Beim Durchlaufen dieser Liste vom kleinsten zum größten Radius wird der Radius gewählt, der von den Kantenpixeln am besten unterstützt wird.
Ein Kreis geht in die endgültige Lösung ein, falls dessen Mittelpunkt durch ausreichend viele Kantenpixel unterstützt wird und einen ausreichend hohen Abstand zu den zuvor detektierten Mittelpunkten hat [OpenCV 2008].

Vor- und Nachteile der Gradienten-Methode

Vorteile:

1) Durch die starke Reduzierung möglicher Kreismittelpunkte sinkt die Rechenzeit um ein Vielfaches.

2) Die Verwendung eines Akkumulator-Arrays an Stelle eines dreidimensionalen Akkumulator-Raums reduziert den Speicherplatzbedarf, gerade bei größeren Radiusbereichen, ganz erheblich.

Nachteile:

1) Die Berechnung des lokalen Gradienten einer Kante und die damit verbundene Annahme, dass die Richtung dieser Kante gleich der angelegten Tangente ist, ist numerisch nicht stabil. Meist trifft diese Annahme zwar zu, falls nicht, kann dies jedoch zu Rauschen in der Ausgabe führen.

2) Da nur ein Kreis pro Mittelpunkt in die Lösung aufgenommen wird, kann, zumindest bei Verwendung der Standard-Implementierung, von konzentrischen Kreisen jeweils nur ein Kreis detektiert werden [OpenCV 2008].

2.8. Begriffsdefinition positiver und negativer Fehler

Im späteren Verlauf dieser Arbeit werden sich Situationen ergeben, in denen die Suche nach einem Objekt nicht das gewünschte Ergebnis liefert. Dabei ist es zum einen möglich, dass zwar ein gesuchtes Objekt, z.B. ein roter Ball, vorhanden ist, jedoch nicht korrekt als Zielobjekt klassifiziert wird. In diesem Fall spricht man von einem „negativen Fehler“ oder einem „Fehler 2. Art“. Zum anderen kann ein Objekt, das eigentlich nicht gesucht wird, fälschlicherweise als Zielobjekt erkannt werden. Dies wird als „positiver Fehler“ oder als „Fehler 1. Art“ bezeichnet [Wikipedia].
Die beiden Fehlerarten sind in Abb. 27 anhand eines Beispiels dargestellt. Ein korrektes Suchergebnis wäre in diesem Fall nur ein markierter Ball rechts. In der Mitte ist ein negativer Fehler zu sehen, da zwar ein Zielobjekt vorhanden ist, jedoch nicht als solches erkannt wird. Rechts wird dagegen zwar das Zielobjekt korrekt detektiert, jedoch auch fälschlicherweise das rote Kästchen. Dies entspricht einem positiven Fehler.

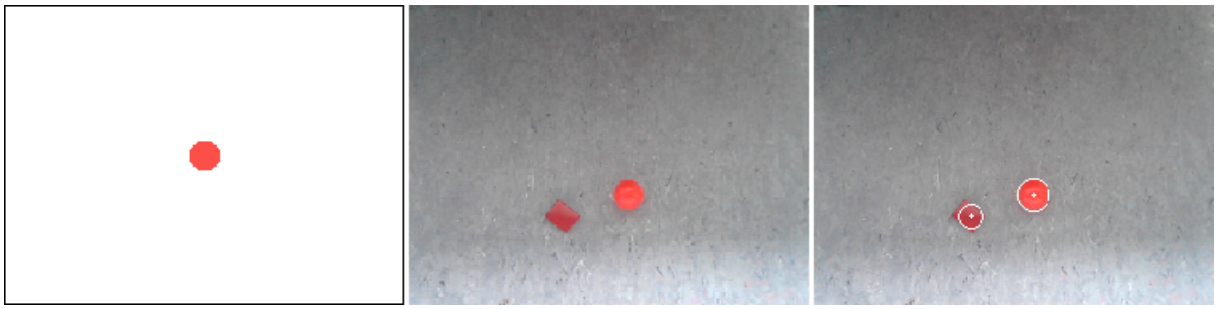

Abbildung 27: Unterschied zwischen positiven und negativen Fehler:
links: zu suchendes Objekt (schematisch)
Mitte: negativer Fehler
rechts: positiver Fehler

2.9. Die OpenCV-*Library*

OpenCV ist eine Open Source Computer-Vision Bibliothek, die in C und C++ programmiert wurde und sowohl für Windows, als auch für Linux und Max OS X verfügbar ist. Die Bibliothek verfügt über mehr als 2500 optimierte Algorithmen mit über 500 verwendeten Funktionen, die verschiedenste Gebiete der Computer-Vision abdecken, darunter auch die Objekterkennung.
OpenCV besteht aus fünf Haupt-Komponenten, von denen vier in Abb. 28 dargestellt sind.

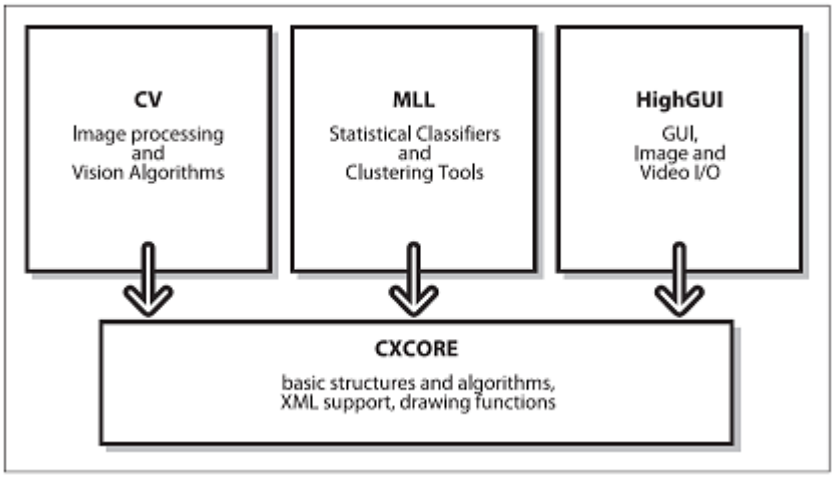

Abbildung 28: Basis-Struktur OpenCV [OpenCV 2008]

Der *CV*-Block besteht aus den grundlegenden Bildverarbeitungs- und fortgeschritteneren Computer-Vision-Algorithmen. Die *Machine Learning Library* (*MLL*) verfügt über zahlreiche Lösungen für *Data-Mining* Probleme wie *Clustering* oder Klassifizierung. Als Benutzeroberfläche steht die *HighGUI* zur Verfügung, die Input/Output Routinen und Funktionen zur Speicherung und zum Laden von Bildern und Videos enthält. *CXCore* stellt die Basisfunktion für die verwendeten Datenstrukturen. In Abb. 28 nicht enthalten ist die *CvAux*-Komponente, die experimentelle Algorithmen, z.B. im Bereich der Gesichtserkennung, enthält [OpenCV 2008].

3. Konzept

Ziel dieser Arbeit ist es, einen Algorithmus für eine Kamera zu implementieren, die sich in einem Quadrocopter befindet, der in etwa einem Meter Höhe fliegt. Dabei soll in einem Farbbild ein bereits bekanntes Objekt wiedererkannt werden. Der Fokus liegt auf Objekten, die in der Draufsicht einem Kreis entsprechen, also Bällen und Kugeln.
Ein zu suchendes Objekt wird durch seine Such-Parameter klassifiziert. Diese setzen sich zusammen aus der Farbe des Objekts und dessen Größe, die wiederum durch den Radius gegeben ist. Festgelegt werden die Such-Parameter durch einen jeweils vorangehenden Scan. Anschließend erfolgt die Suche nach eben diesen Parametern.

3.1. Grundidee

Es liegt in der Natur der Sache, dass man auf der Suche nach einem roten Ball öfter Bilder untersucht, die keinen solchen Ball enthalten, als Bilder, in denen man fündig wird. Des Weiteren enthalten im Schnitt nur wenige Bilder genügend rote Pixel, Zielpixel, um überhaupt einen roten Ball beinhalten zu können. Im Gegensatz dazu existieren in fast jedem Bild Kanten. Nur selten wird mit vollkommen glatten und komplett monochromen Oberflächen gearbeitet werden. Würde jedes Bild zuerst auf Kanten bzw. Kreise untersucht werden, müsste man immer eine komplette Hough-Kreis-Transformation durchführen, um festzustellen, dass sich auf dem Bild weder ein Kreis noch ein roter Ball befindet.
Die Vorteile der Farberkennung sind offenkundig. Zum einen ist der Rechenaufwand, gerade im Vergleich zur aufwändigen Hough-Kreis-Transformation, sehr gering. Jeder Pixel wird einmal durchlaufen und entweder als Zielpixel markiert oder nicht. Sind im Bild zu wenig Zielpixel vorhanden, kann ohne Weiteres abgebrochen und ein neues Bild in Angriff genommen werden.
Wurden jedoch genügend Zielpixel gefunden und als Binärbild, d.h. 1: Zielpixel, 0: kein Zielpixel, abgespeichert, kommt die Hough-Kreis-Transformation zum Einsatz. Deren Ablaufzeit ist deutlich kürzer, da die Anzahl der Kantenpixel auf ein Minimum reduziert wurde. Im Binärbild sind nur noch solche Kanten vorhanden, die zwischen Zielpixeln und Nicht-Zielpixeln liegen. Überflüssige Kanten, wie die von Schatten, Unreinheiten in der Oberfläche oder anderen Objekten, die die Anforderungen nicht erfüllen, z.B. blaue Bälle, sind nicht mehr vorhanden, bremsen den Algorithmus nicht mehr und verfälschen das Ergebnis nicht.

3.2. Überblick

Abb. 29 gibt einen Überblick über das Grundkonzept dieser Arbeit in Form eines Ablaufdiagramms.
Das Konzept sieht vor, zunächst die Parameter für die Suche nach dem gewünschten Objekt festzulegen. Dazu erfolgt ein Scan eines Bildes, das ein Zielobjekt enthält. Aus diesem Bild werden sowohl die gesuchte Farbe, als auch der gewünschte Radius für die Hough-Kreis-Transformation, d.h die gesuchte Größe des

Balls, ausgelesen.

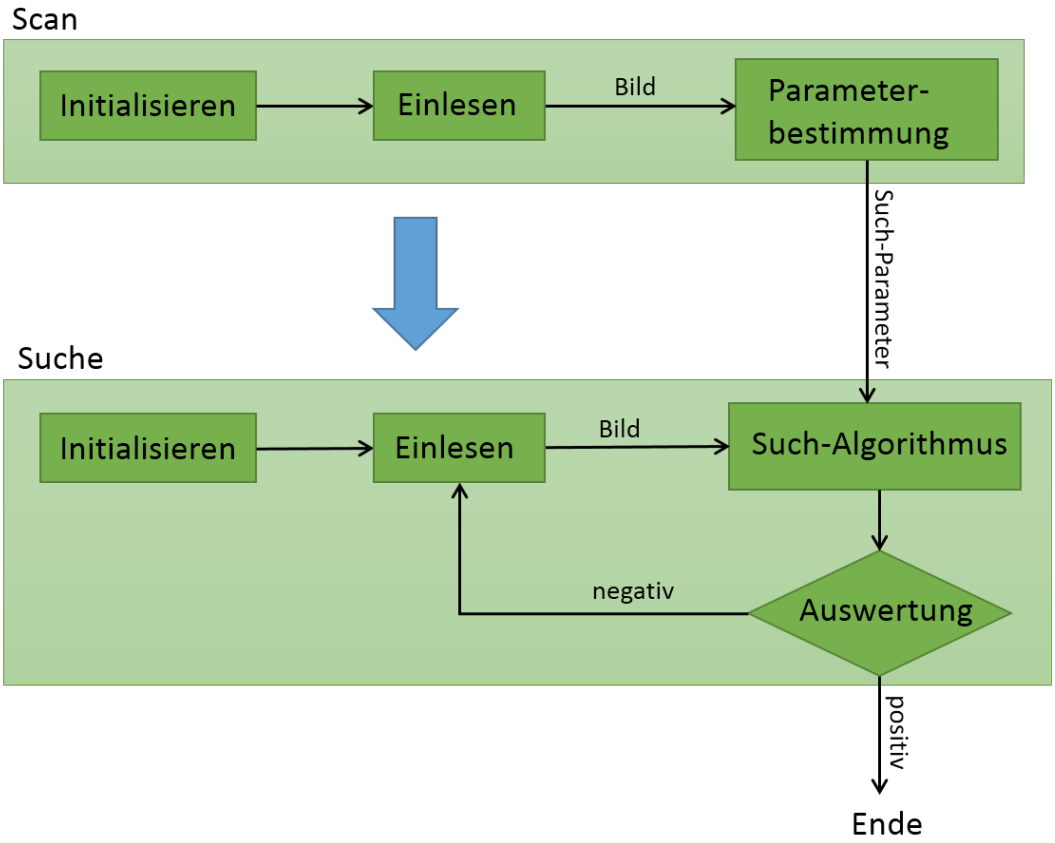

Abbildung 29: Grundkonzept: Scan mit anschließender Suche

Dabei wird meist nicht nur nach einem einzelnen Radius, sondern nach einem ganzen Radius-Bereich gesucht. So können auch Objekte in Bildern erkannt werden, die aus einer anderen Höhe als das Scanbild aufgenommen wurden.
Mit Hilfe dieser Parameter wird der Suchalgorithmus auf das zu untersuchende Bild angewendet. Abschließend wird das Ergebnis ausgegeben. In der Grundfunktion ist hierbei vorgesehen, dass der Algorithmus endet, falls das Objekt gefunden wurde, d.h. die Auswertung ein positives Ergebnis liefert. Wurde kein Objekt gefunden, d.h. die Auswertung liefert ein negatives Ergebnis, beginnt die Schleife von vorne und das nächste Bild wird eingelesen.

3.3. Der Scan

Der im vorherigen Abschnitt kurz erwähnte Ablauf des Scans, vom Einlesen des Scanbilds bis zur Ausgabe der Such-Parameter, ist in Abb. 30 dargestellt.
Nach dem Einlesen eines Bildes erfolgt die Extraktion der Rohdaten. Aus diesen wird eine Graustufen-Matrix berechnet, mit der anschließend die Kantendetektion durchgeführt wird. Um unnötige Rechenzeit zu vermeiden und die Chance auf eine korrekte Identifizierung des Zielobjekts zu steigern, werden vor der Kreissuche die schwächeren Kanten herausgefiltert.

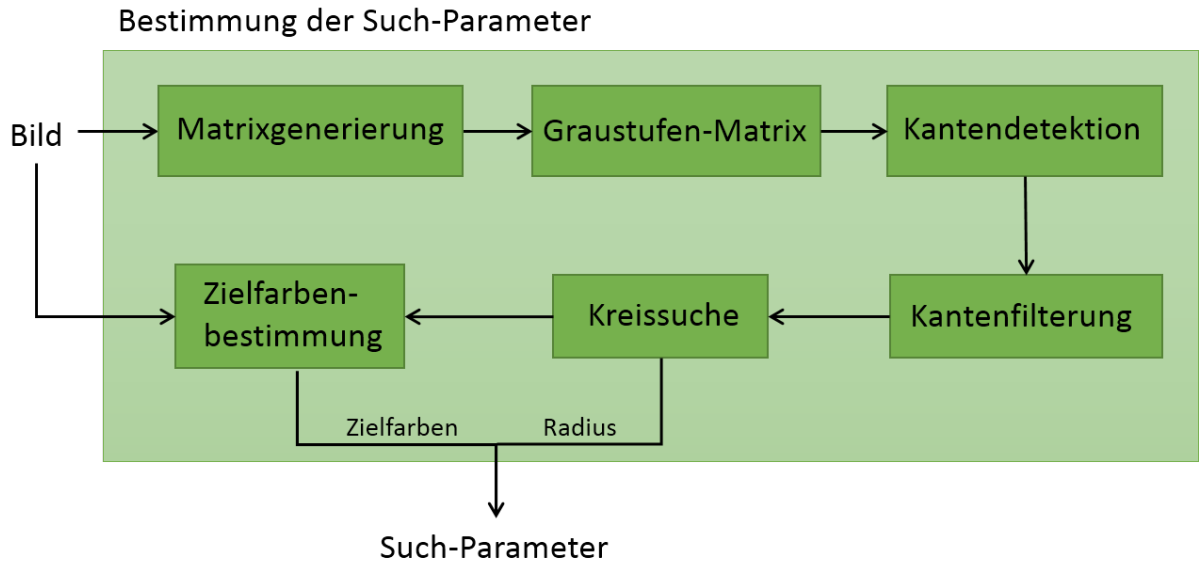

Abbildung 30: Ablauf der Bestimmung der Such-Parameter

Der zu suchende Radius ergibt sich direkt aus der Kreissuche. Verlief die Detektion reibungslos, schließt der Kreis das Zielobjekt ein. Nun können mit Hilfe der Rohdaten die Farben innerhalb des eben ermittelten Kreises bestimmt werden. Dazu werden mittels der Kreisgleichung die Pixel im Inneren des Kreises ermittelt und mittels dreier Histogramme, eines für jeden Farbkanal, analysiert. Die am häufigsten auftretenden Farbwerte jedes Farbkanals ergeben die gesuchten Farben. Zusammen mit dem Radius bilden sie die Such-Parameter.

3.4. Die Suche

Abb. 31 zeigt den detaillierten Ablauf des Suchvorgangs vom Einlesen des Suchbilds bis zur Ausgabe der gefundenen Kreise, falls vorhanden.
Um ein im Scan charakterisiertes Zielobjekt in einem Zielbild zu detektieren, werden zunächst wieder die Rohdaten des Bildes ausgelesen und in einer Matrix gespeichert. Anschließend wird, Pixel für Pixel, überprüft, ob die jeweiligen Pixel-Werte innerhalb der Grenzen der Zielfarben liegen. Das Ergebnis wird in einer Binär-Matrix gespeichert. Hierbei bedeutet der Wert 1, dass der jeweilige Pixel innerhalb der Grenzen liegt. Wird an dieser Stelle festgestellt, dass sich im Suchbild zu wenig Zielpixel befinden, wird die Suche abgebrochen. Innerhalb des Binär-Bildes erfolgt, analog zum Scan, im Anschluss die Kantendetektion und die Kreissuche. Bei Letzterer wird der aus dem Scan bekannte Radius verwendet. Die gefundenen Kreise werden einer Prüfung unterzogen, die sicherstellt, dass nur die Kreise weiter betrachten werden, die sich anhand der Binär-Matrix als am passendsten herausstellen. Qualitätskriterium ist diesbezüglich die Anzahl an Pixeln in der Binär-Matrix, die auf dem Umfang des jeweiligen Kreises liegen. Des Weiteren wird gewährleistet, dass sich nicht mehrere Kreise im selben Flächenabschnitt befinden, sich also nicht schneiden.

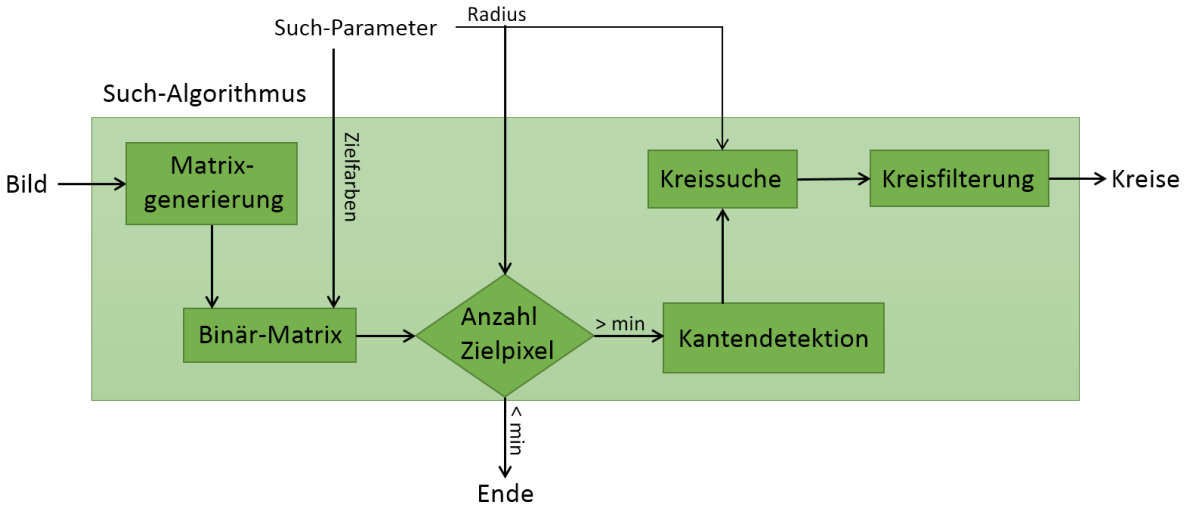

Abbildung 31: Ablauf der Suche

3.5. Die Auswertung

Bei der Auswertung wird die abschließende Evaluierung der einzelnen Kreise vorgenommen. Nur Kreise, deren Flächeninhalt einen ausreichend hohen Prozentsatz an Zielpixeln enthält, werden als endgültiges Zielobjekt klassifiziert. Ist von diesen mindestens eines vorhanden, so wird das Objekt bzw. werden die Objekte im Original-Bild markiert. Ist keines vorhanden, kehrt der Algorithmus, wie in Abb. 29 zu sehen, zum Einlesen eines neuen Bildes zurück. Abb. 32 zeigt am Beispiel eines roten Balls den gesamten Ablauf vom Scannen des gesuchten Objekts bis hin zum Anzeigen des Ergebnisbildes.

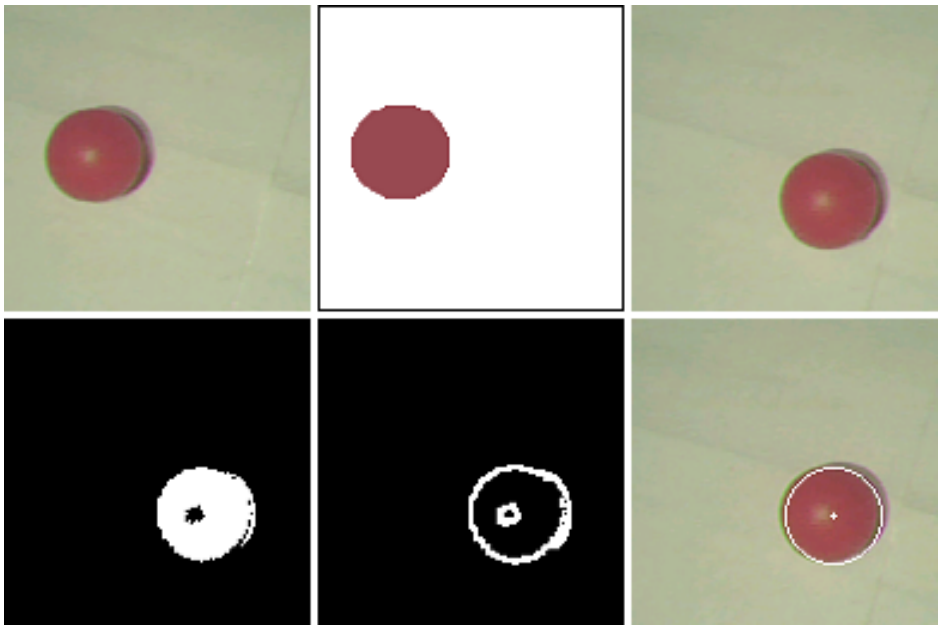

Abbildung 32: Möglicher Ablauf in Bildern:
oben: Scanbild, Ziel-Parameter als Bild dargestellt, Suchbild
unten: Binärdarstellung des Suchbildes, Ergebnis der Kantendetektion, Ergebnisbild mit eingezeichnetem Kreis

4. Implementierung

4.1. Hardware

Zur Durchführung des Scans und zur Suche des Zielobjekts werden Bilder benötigt. Diese können sowohl aus einer Datei stammen als auch direkt von einer angeschlossenen Kamera geliefert werden. Um dies zu gewährleisten wurden zwei verschiedene Hardware-Aufbauten entwickelt: Eines zum Testen der Implementierung während der Entwicklung und eines für den abschließenden Test und die weitere Verwendung im Quadrocopter.

4.1.1. Setup zum Testen des Programms

Abb. 33 zeigt das Setup, das zum Testen der Implementierung und zur Optimierung der Parameter verwendet wird.

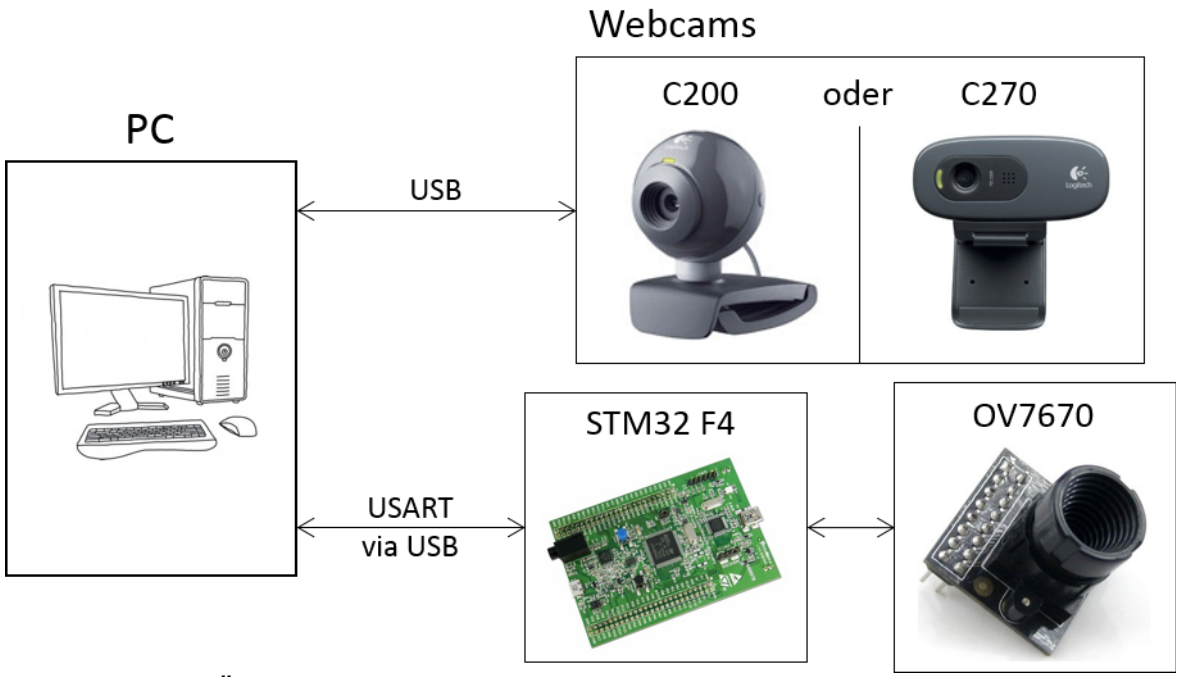

Abbildung 33: Überblick über die zum Testen verwendete Hardware

Das im späteren Verlauf dieser Arbeit vorgestellte Programm wird dabei vollständig auf einem PC ausgeführt. Die Art der Kommunikation zwischen PC und Kamera hängt von der verwendeten Kamera ab. So sind die beiden Webcams, Logitech C200 und C270, direkt per USB mit dem PC verbunden, während die Kommunikation mit der OV7670 über das STM32 F4 Discovery-Board erfolgt, das wiederum per USB mit dem PC verbunden ist.

4.1.2. Setup für den Flug im Quadrocopter

Um die Funktionalität des neu entwickelten Systems unter Realbedingungen, d.h. im freien Flug zu testen, muss es in das bereits bestehende Quadrocoptersystem eingebunden werden. Abb. 34 gibt einen Überblick über die verwendeten Komponenten.

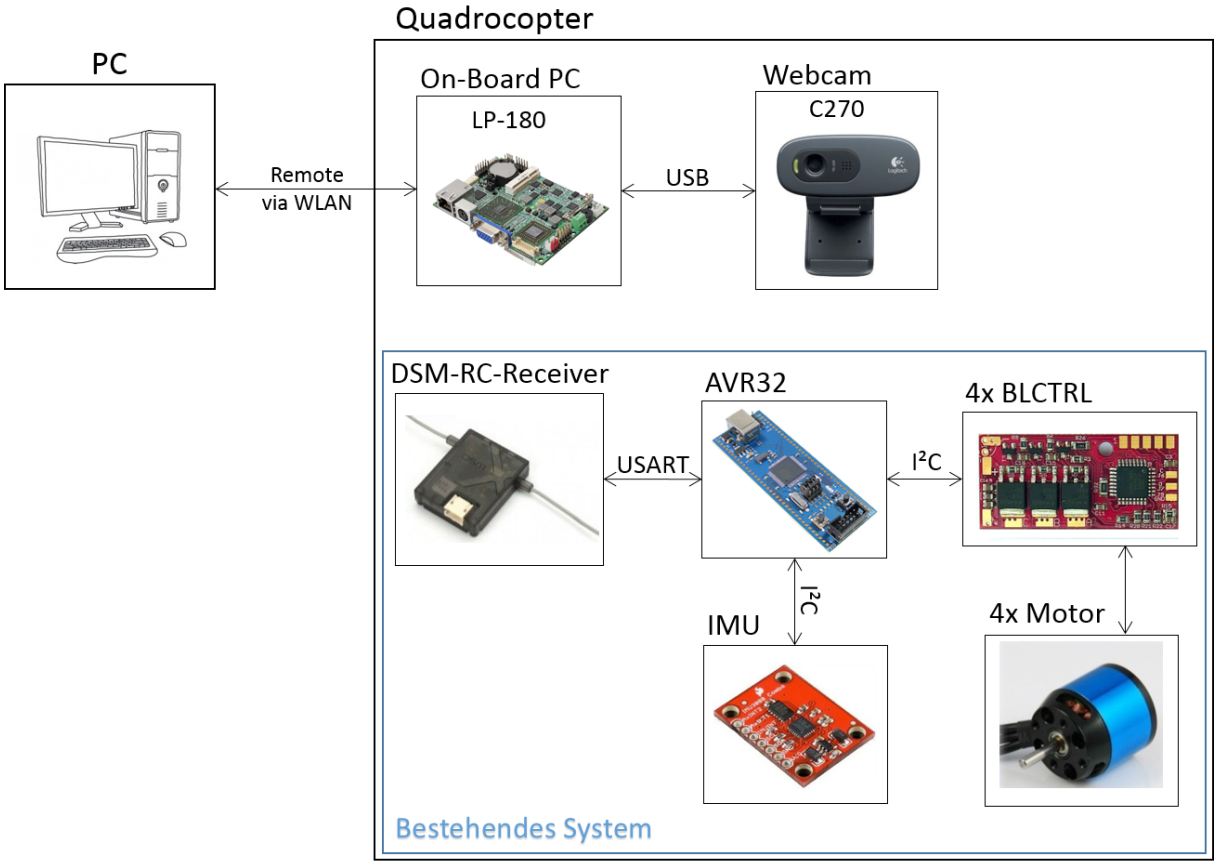

Abbildung 34: Integration des neu entwickelten Systems in das bestehende Quadrocopter-System

Im unteren Bildbereich sind die Komponenten des bereits bestehenden Systems zu sehen. Links befindet sich der DSM-RC-Receiver, der eine Fernsteuerung des Quadrocopters ermöglicht. Dieser kommuniziert per USART mit dem AVR32-Board, das die zentrale Komponente des Systems darstellt und die Regelung im Flug ausführt. Eine inertiale Messeinheit (engl.: *Inertial Measurement Unit*, kurz IMU) liefert per I²C die zur Regelung benötigten Sensordaten an das Board. Die Kombination dieser Informationen mit der Eingabe durch den Receiver erlaubt vier bürstenlosen Controllern die Steuerung der vier Motoren.

Oben im Bild befinden sich die Komponenten des neu entwickelten Systems zur Objekterkennung. Dabei dient ein PC als Bodenstation. Von dort aus ist es während des Flugs möglich, den Ablauf des Programms zu überwachen und ggf. einzugreifen. An Board des Quadrocopters befindet sich der Pico-PC LP-180 der Marke COMMELL. Die Notwendigkeit eines direkt an Board installierten PCs erklärt sich durch die hohe Rechenleistung, die bei Ausführung des Programms benötigt wird. Das Anschließen der C270 Webcam, die während des Flugs Bilder aufnimmt und an den LP-180 sendet, erfolgt per USB. Das Gleiche gilt für die Kommunikation mit der

Bodenstation. Dazu wird eine W-Lan-Verbindung verwendet, durch die der LP-180 per Remote-Desktop-Verbindung ferngesteuert werden kann.

4.2. Software

Der gesamte Ablauf des Scannens und der Suche wurde zweifach implementiert, einmal unter Verwendung der OpenCV-*Library* und einmal ohne.
Bei der OpenCV-Implementierung werden einzelne Funktionen der *Library* verbunden und durch einige eigens programmierte Abschnitte ergänzt, um die gewünschte Funktionalität zu erreichen. Im Gegensatz dazu finden im Ansatz der eigenen Implementierung quasi ausschließlich selbst programmierte Funktionen Verwendung. Ausgenommen davon sind die OpenCV-Funktion zum Einlesen der Webcam-Bilder und Qt-interne Funktionen. Des Weiteren wird der bereits bestehende Algorithmus der Hough-Kreis-Transformation verwendet.

4.2.1. Übersicht

Der gesamte Code ist in C++ geschrieben. Als Entwicklungsumgebung dient der Qt Creator (Version 2.4.1) aus dem Qt SDK (Version 4.7.4) der Firma Nokia. Dort wird das bereits vorhandene QT-Steuerprogramm des AQopterI8-Projekts des Informatik-Lehrstuhls VIII der Universität Würzburg als Basis verwendet. Dabei wird bei der Nutzung der OV7670 auf den von Michael Strohmeier entwickelten Treiber zur Bildaufnahme zurückgegriffen [Strohmeier 2012].
Im Folgenden soll die Implementierung der in Kapitel 3 genannten Blöcke näher vorgestellt werden.

4.2.2. Initialisieren der Bildquelle und Einlesen der Bilddaten

Die zum Einlesen benötigten Funktionen sind bereits in Qt bzw. der OpenCV-*Library* gegeben: Ist eine Bild-Datei das Ziel, so kann diese direkt in das von Qt verwendete Bild-Format *QImage* gespeichert werden. Soll dagegen ein Bild einer der beiden Webcams geladen werden, wird dieses mittels der OpenCV-Funktion *read(Mat)* in das Open-CV interne Bild-Format *Mat* gespeichert, das anschließend in ein *QImage*-Bild umwandelt wird. Bilder der OV7670-Kamera werden durch eine bereits im QT-Steuerprogramm implementierte Funktion *snapshot()* aufgenommen und anschließend ebenfalls ins *QImage*-Format umgewandelt.
Erfolgt das Laden bzw. Aufnehmen eines Bildes während der Scan-Phase, werden dessen Dimensionen, d.h. Höhe und Breite, ausgelesen, gespeichert und in allen folgenden Berechnungen verwendet.

4.2.3. Bestimmen der Such-Parameter

Im Folgenden werden die einzelnen Schritte des Scans näher beschrieben. Zunächst werden die Bilddaten eingelesen. Danach erfolgt eine Detektion der darin enthaltenen Kanten, mit denen anschließend die Kreisdetektion durchgeführt wird. Zuletzt werden die Zielfarben aus dem gefundenen Kreis ermittelt. Die einzelnen

Blöcke entsprechen dabei denen aus dem vorgestellten Konzept (vgl. Abb. 30).

Matrixgenerierung:

Da der Zugriff auf die einzelnen Pixel eines Bildes im *QImage*-Format relativ langsam erfolgt und der Code schnell unübersichtlich wird, werden die Bilder zunächst in *Rgb*-Matrizen gespeichert. *Rgb* ist dabei ein selbst erstelltes *Struct*, das die einzelnen Rot-, Grün- und Blauwerte des jeweiligen Pixels speichert. Wie alle noch folgenden Matrizen besitzt die *Rgb*-Matrix dieselben Abmessungen wie das ursprünglich aufgenommene Scanbild.

Graustufen-Matrix:

Anschließend erfolgt die Umwandlung in das Graustufen-Format. Hierbei wird das in Kapitel 2.3.2 vorgestellte Verfahren des einfachen Mittelwertes verwendet, da dessen Berechnung die einfachste Lösung darstellt, und sich die Ergebnisse der verschiedenen Verfahren fast nicht unterscheiden. Das Graustufenbild wird als *unsigned char*–Matrix gespeichert. Dabei wird der Datentyp *unsigned char* wegen seines Speicherplatzbedarfs von 8 Bit gewählt. Dies entspricht einem Wertebereich von 0 bis 255.

Kantendetektion:

In der folgenden Kantendetektion werden die Sobel-Operatoren H_x^S und H_y^S aus Kapitel 2.6.2. angewendet und kombiniert.

$$H_x^S = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ und } H_y^S = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{4.1}$$

Dabei ist zu beachten, dass die Kantenwerte während der Berechnung zu *int* gecastet werden müssen. Der Definitionsbereich eines *unsigned char* beträgt nur 0 bis 255 und eignet sich damit nicht zur Berechnung der Kantenwerte, da dabei die Obergrenze von 255 leicht übertroffen werden kann. Das so entstandene Kantenbild wird wiederum als *unsigned char*-Matrix gespeichert.

Kantenfilterung:

Vor der Kreissuche werden zunächst die schwächeren Kanten herausgefiltert. Dazu wird der durchschnittliche Wert aller Kanten berechnet und mit einem Kantenfaktor multipliziert. Anschließend werden alle Kanten auf 0 gesetzt, die schwächer als der so entstandene Schwellenwert sind. Der eben erwähnte Kantenfaktor wird als *#define* im Code angegeben. Ein Wert von 5 bis 7 erweist sich dabei je nach Hintergrundrauschen als sinnvoll, wie in Abb. 35 am Beispiel des Scans eines roten Balls zu sehen ist. Durch diese Filterung wird die Anzahl der Kanten im Bild deutlich reduziert. Dies führt zu weniger Rechenaufwand und einem schnelleren Scan. Des Weiteren sinkt die Chance, dass durch Störkanten, wie z.B. Kratzer in der Oberfläche, auf der sich das zu scannende Objekt befindet, ein falscher Kreis detektiert und dadurch ein unbrauchbares Scanergebnis erzielt wird.

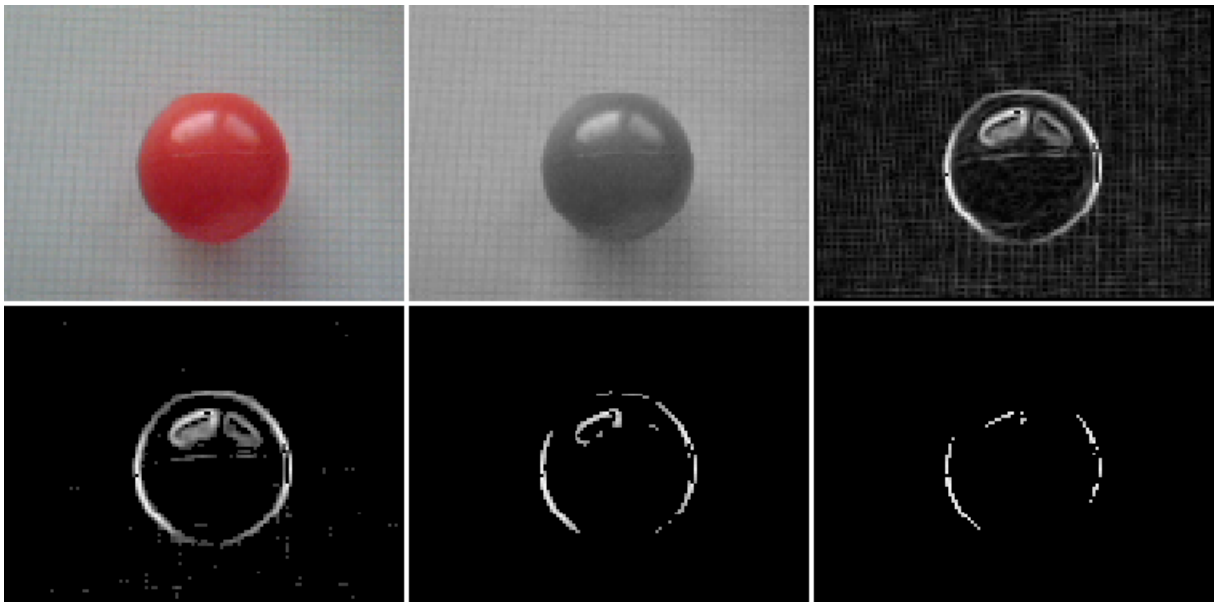

Abbildung 35: Scan eines roten Balls mit unterschiedlichen Kantenfaktoren:
oben: Originalbild, Graustufenbild, Faktor 0
unten: Faktor 2, Faktor 5, Faktor 7

Die jeweilige Anzahl der Kantenpunkte und Dauer des Scans sind in Tabelle 2 zu sehen.

Kanten-Faktor	Anzahl der Kantenpunkte	Dauer des Scans (ms)
0	11706	970
2	752	120
5	180	50
7	77	40
10	0	10

Tabelle 2: Kanten-Faktor, Anzahl der Kantenpunkte und Dauer des Scans

Trotz dieser Möglichkeit, Stör-Kanten herauszufiltern, sollte beim Scan darauf geachtet werden, einen möglichst neutralen, einfarbigen Hintergrund zu wählen, der einen hohen Kontrast zum Zielobjekt ausweist. Des Weiteren sollte der Scan aus geringer Höhe erfolgen, sodass sich für den gefundenen Radius möglichst kein Wert unter 10 Pixeln ergibt. Ein größerer Radius führt zu besseren Scanergebnissen, da im Inneren des Kreises mehr Pixel zur Verfügung stehen, um eine repräsentative Farbe zu bestimmen.

Kreisdetektion:

Anschließend erfolgt die Kreisdetektion. Diese ist im Scan darauf ausgelegt, dass genau der Kreis, der die meisten Kanten-Pixel schneidet, gefunden wird. Als Eingabe erhält die Kreissuche-Funktion eine *unsigned char*-Matrix, die die Kanten enthält, und zwei Radien, r_{min} und r_{max}, als *Integer*-Werte. Mit diesen Werten wird nun die Hough-

Kreis-Transformation, wie in Kapitel 2.7. beschrieben, durchgeführt. Dabei werden zwei Änderungen am Original-Algorithmus vorgenommen, um die Laufzeit zu verbessern und im Schnitt bessere Ergebnisse zu erzielen:

Zum einen wird nicht für jeden Kantenpunkt das gesamte Array überprüft, ob der jeweilige Punkt die Kreisgleichung erfüllt. Stattdessen wird der zu untersuchende Bereich auf ein Quadrat reduziert, das den Kreis mit dem jeweiligen Radius um den aktuellen Kantenpunkt gerade noch einschließt.
Dazu werden für jeden zu untersuchenden Kantenpunkt je zwei Grenzen für x- (*east, west)* und y-Wert (*north*, *south*) berechnet, innerhalb derer die Punkte liegen müssen, die die Kreisgleichung erfüllen. Befindet sich z.B. ein Kantenpunkt *P(x, y)* bei den Koordinaten *(50, 70),* so müssen die Punkte, die auf einem Kreis mit Radius *R = 10* um *P* liegen, sich alle innerhalb eines Quadrates mit Mittelpunkt *P* und Seitenlänge *2 * R* befinden. Dies entspricht einem x-Wert von 40 (*west*) bis 60 (*east*) und einem y-Wert von 60 (*north*) bis 80 (*south*). In diesem Beispiel müssten also, bei einer angenommenen Bildgröße von 128 x 128, 400 ((60-40) x (80-60)) statt 16384 (128 x 128) Punkte untersucht werden.
Zum anderen wird der Akkumulator-Wert eines Punktes nicht nur erhöht, wenn dieser die Kreisgleichung exakt erfüllt, sondern auch, wenn sich eine Abweichung von einem Pixel ergibt. Dabei wird der Akkumulator-Wert um 3 angehoben, falls die Kreisgleichung exakt erfüllt wird, ansonsten um 1. Gerade bei Bildern mit sehr niedriger Auflösung können mit dieser Erweiterung oft bessere Ergebnisse erzielt werden, da der Einfluss von Stör-Kanten reduziert und ausgeprägte Kreisstrukturen sicherer erkannt werden.

Der detektierte Kreis wird als eigens erstelltes *Struct Circle* gespeichert, bestehend aus den x- und y-Koordinaten des Mittelpunktes, dem Radius des Kreises und dem Akkumulator-Wert an dieser Stelle. Aufgrund der im letzten Absatz erwähnten Erweiterung entspricht dieser im Allgemeinen nicht der wirklichen Anzahl der vorhandenen Kanten des Kreises, sondern liegt meist etwas höher. Der Radius stellt die erste Hälfte der Such-Parameter dar, es folgen die zu suchenden Farben.

Bestimmen der zu suchenden Farben:

Zum Bestimmen der Farben werden die Pixel betrachtet, die im Originalbild innerhalb des eben berechneten Kreises liegen. Analog zur Kreissuche wird mittels der Kreisgleichung überprüft, ob sich die in Frage kommenden Pixel innerhalb des Kreises befinden. Dabei wird allerdings nicht der volle Radius verwendet, sondern nur ein um den Faktor 0,9 reduzierter Teil davon. Der Grund dafür sind häufig auftretende Schatteneffekte im Randbereich der zu scannenden Bälle, die die Farben verfälschen könnten.
Um die tatsächlichen Farbwerte zu bestimmen, werden drei Histogramme der im Inneren liegenden Pixel erstellt. Jeweils eines für Rot, Grün und Blau. Dabei kommt das in 2.4.1. erwähnte *Binning* zum Einsatz. Für jeden Farbkanal wird der Wertebereich von 0 bis 255 in 32 *Bins* zu je acht Werten unterteilt, d.h. 0 bis 7, 8 bis 15 usw. Anschließend werden die Pixel überprüft und jeweils der Zähler des *Bins*, in dem der jeweilige Wert liegt, um 1 erhöht. Das Ziel ist es, dabei die Varianz zu vermindern und das Verfahren robuster zu gestalten. So wird verhindert, dass ein einzelner, zufällig am häufigsten vorkommender Wert die Zielfarbe bestimmt. Dagegen werden mehrere, wiederholt auftretende und nah beieinanderliegende Werte gestärkt, auch wenn jeder einzelne von ihnen seltener auftritt als der eben

erwähnte Ausreißer. 32 *Bins* erweisen sich dabei als guter Kompromiss, da damit ein ausreichender Schutz gegen Ausreißer gewährleistet wird, jedoch die zu einem *Bin* zusammengefassten Farbwerte noch nicht zu unterschiedlich sind. Der abschließende Farbwert ergibt sich aus dem Durchschnitt der Werte des *Bins*, dessen Zähler nach dem Durchlauf aller Pixel den höchsten Wert besitzt.
Die so erhaltenen, am häufigsten vorkommenden Werte für Rot, Grün und Blau sind als Scanergebnis noch nicht ausreichend. Im Zielbild soll schließlich nicht nach einzelnen Werten, sondern nach Werten innerhalb bestimmter Bereiche gesucht werden.
Zunächst wird für jeden Farbwert *C* dessen Scantoleranz S_T wie folgt berechnet:

$$S_T(C) = max(T_{min}, \frac{C * T_S}{255}) \tag{4.2}$$

Dabei ist *C* der Grundwert der jeweiligen Farbe. Der Toleranzfaktor T_S ermöglicht es, den Toleranzbereich proportional zum Grundwert zu dimensionieren, sodass große Grundwerte später bei der Suche größere Abweichungen zulassen als kleine. Hier erweist sich 100 für die meisten Anwendungen als passender Wert. Ein zu hoher Wert kann zu Fehldetektionen führen, da evtl. auch der Zielfarbe relativ unähnliche Farben positiv detektiert werden. Dagegen macht ein zu niedriger Wert das System sehr anfällig gegenüber Veränderungen der Lichtverhältnisse. Die Minimaltoleranz T_{min} entspricht einem *#define* und gibt den Wert der Mindesttoleranz an. Dadurch soll verhindert werden, dass der Toleranz-Bereich einer Farbe zu gering ausfällt, wenn deren Grundwert sehr niedrig ist. 20 erweist sich dabei als guter Wert.
Anschließend werden die Grundwerte und die neu berechneten Grenzwerte, d.h. die Minimum- und Maximum-Werte der jeweiligen Farbe, in einem selbst erstellten *Struct MinMaxRgb* gespeichert. Minimum und Maximum berechnen sich dabei jeweils aus:

$$min = satLow(C - S_T(C)) \tag{4.3}$$

$$max = satHigh(C + S_T(C)) \tag{4.4}$$

Die Funktionen *satLow* und *satHigh* stellen sicher, dass sich die Werte nicht außerhalb des Definitionsbereichs, d.h. von 0 bis 255, befinden. Bei der Bestimmung der Maximalwerte ist zu erwähnen, dass die dominierende Farbe auf 255 gesetzt wird. Diese Maßnahme soll bei der Suche eine größere Toleranz gegenüber stärkerem Lichteinfall gewährleisten, da dieser zu einer Erhöhung aller Farbwerte führt. So soll das Setzen der dominanten Farbe auf 255 bewirken, dass bei der Suche nach einem roten Ball ein mögliches Zielobjekt nicht abgewiesen wird, weil es zu rot ist.
Damit ist die Bestimmung der Farbwerte abgeschlossen und zusammen mit dem bei der Kreissuche gefundenen Radius bilden diese die Such-Parameter.

4.2.4. Der Such-Algorithmus

Im Folgenden sei der Ablauf des Such-Algorithmus näher dargestellt. Dieser erhält als Eingabe das Suchbild im *QImage*-Format und die vorher bestimmten Such-Parameter bestehend aus Farben und Radius. Die einzelnen Blöcke beziehen sich

dabei auf das in Abb. 31 zu sehende Blockdiagramm.

Matrixgenerierung:

Analog zu der Bestimmung der Such-Parameter wird zunächst das Bild, in dem die Suche erfolgen soll, vom *QImage*-Format in eine *Rgb*-Matrix umgewandelt, die die Roh-Daten der Pixel enthält.
Anschließend erfolgt die Berechnung der Binär-Matrix. Dabei wird ein Pixel, der innerhalb der Grenzen der Such-Parameter liegt, nicht mit 1, sondern mit 255 kodiert. Dies hat keinen Einfluss auf die folgende Kantendetektion, ermöglicht aber später eine einfachere Darstellung des Bildes.

Kantendetektion:

Der Ablauf der Kantendetektion entspricht dem bei der Bestimmung der Such-Parameter. Es werden wiederum Sobel-Operatoren (vgl. Gl. 4.1) eingesetzt, die Werte während der Berechnung zu *int* gecastet und die Ergebnis-Werte als *unsigned char*-Matrix gespeichert.

Kreisdetektion:

Bei der folgenden Kreisdetektion werden ebenfalls die bei der Bestimmung der Such-Parameter vorgestellten Erweiterungen zum Standard-Verfahren verwendet. Der größte Unterschied besteht in der Anzahl der gefundenen Kreise. Während bei der Bestimmung der Such-Parameter immer nur ein Kreis detektiert wird, können es bei der Kreisdetektion der Suche durchaus mehrere sein. Diese werden in einem Kreis-*Vector* gespeichert.

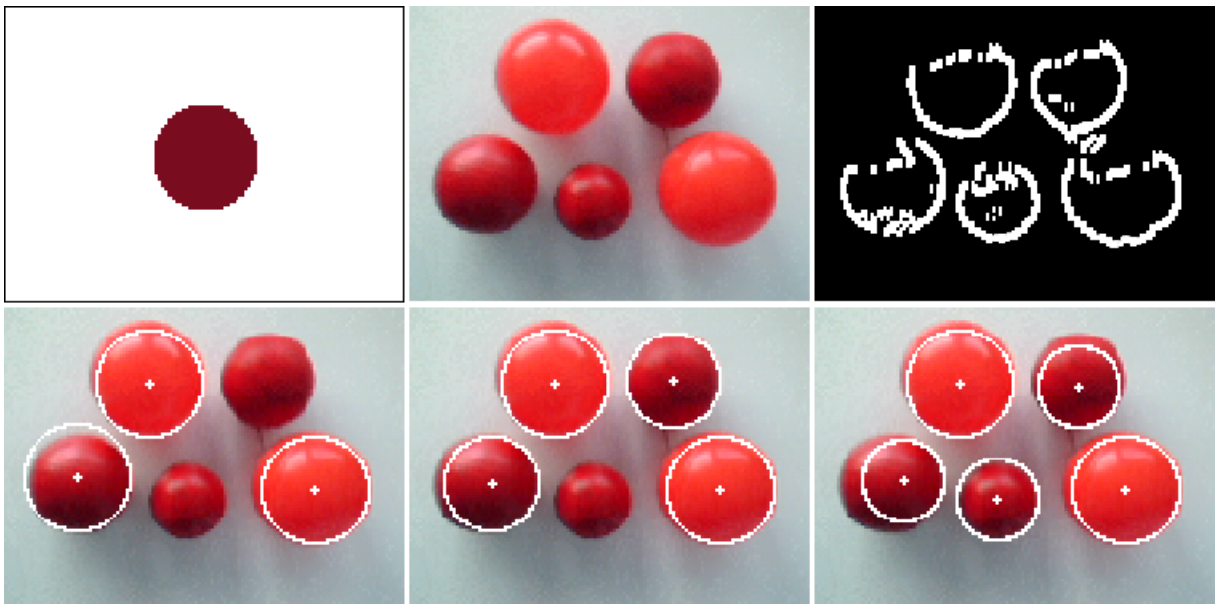

Abbildung 36: Auswirkung der Anzahl der Radien in der Hough-Kreis-Transformation auf die Anzahl der detektierten Bälle:
oben: Scanergebnis, Suchbild, Ergebnis der Kantendetektion
unten: Ergebnisbild für Radius +-0, für Radius +-2, für Radius +-4 bis +-16

Die Kreissuche erfolgt dabei nicht ausschließlich mit dem in den Such-Parametern festgelegten Radius *R*, der lediglich den Radius des zu scannenden Objektes im Scanbild darstellt. Erfolgt die Suche in einer anderen Höhe als der Scan, ändert sich

der optimale Radius. Aus diesem Grund werden neben *R* noch dessen Nachbarn (R+-1, R+-2 usw.) untersucht. Deren Anzahl ist variabel und hat einen starken Einfluss auf die Laufzeit des Algorithmus. Zu sehen ist dies in Tabelle 3. Die Werte beziehen sich auf die in Abb. 36 gezeigte Suche in einem Einzelbild mit fünf Bällen und Original-Radius 17.

Abweichung vom Radius *R*	Anzahl Radien	Laufzeit (ms)	Anzahl detektierter Bälle
+-0	1	30	3
+-2	5	80	4
+-4	9	110	5
+-6	13	150	5
+-8	17	190	5
+-10	21	230	5
+-12	25	310	5
+-14	29	580	5
+-16	33	3990	5

Tabelle 3: Auswirkung der Anzahl der Radien in der Hough-Kreis-Transformation auf die Laufzeit der Suche

Auffällig ist, dass mit den beiden kleinsten Radius-Bereichen nicht alle Bälle detektiert werden. Dies lässt sich dadurch erklären, dass die wahren Radien der beiden kleinsten Bälle nicht innerhalb des Radius-Bereichs liegen und dadurch nicht die Mindest-Kantenzahl (s.u.) erreicht wird.
Des Weiteren fällt auf, dass die Laufzeit beim Anstieg von +-14 zu +-16 Radien einen großen Sprung macht. Der Grund dafür ist das Erreichen des Radius der Größe 1 Pixel (Original-Radius 17 – 16 = 1). Dadurch steigt die Anzahl der gefundenen Kreise stark an. Da ein Radius dieser Größe keinerlei Sinn ergibt und allgemein sehr kleine Radien sich nicht gut zur Detektion eignen, wird der Mindestradius auf 5 Pixel festgelegt.
Nach der Füllung des Akkumulator-Raums wird dieser durchlaufen und nach Zellen durchsucht, die eine gewisse Mindest-Kantenzahl aufweisen. Diese berechnet sich dabei aus dem verwendeten Radius *R* und dem benötigten Mindest-Prozentsatz des Kreisumfangs $Circ_{min}$, der vorhanden sein muss:

$$E_{min} = 2 * \pi * R * \mathrm{Circ}_{min} \tag{4.5}$$

Der minimale Kantenwert E_{min} entspricht demnach einem durch den *#define* bestimmten Anteil des Umfangs eines Kreises mit Radius *R*. Dabei erweist sich ein eher kleiner Wert von 0,3 bis 0,4 als angebracht. Ein zu großer Wert führt dazu, dass korrekte Zielobjekte nicht erkannt werden, falls ihr Umfang durch eine Überdeckung oder ungünstige Lichtverhältnisse reduziert erscheint. Ist der Wert zu klein gewählt, werden häufig andere Formen wie z.B. Rechtecke fälschlicherweise als Kreis detektiert. Abb. 37 zeigt, wie unterschiedlich die Kreisdetektion bei nur leichten Veränderungen des benötigten Mindest-Prozentsatzes des Kreisumfangs ausfallen

kann. Die korrekte Dimensionierung dieses Wertes stellt eine der größten Herausforderungen dieser Arbeit dar.

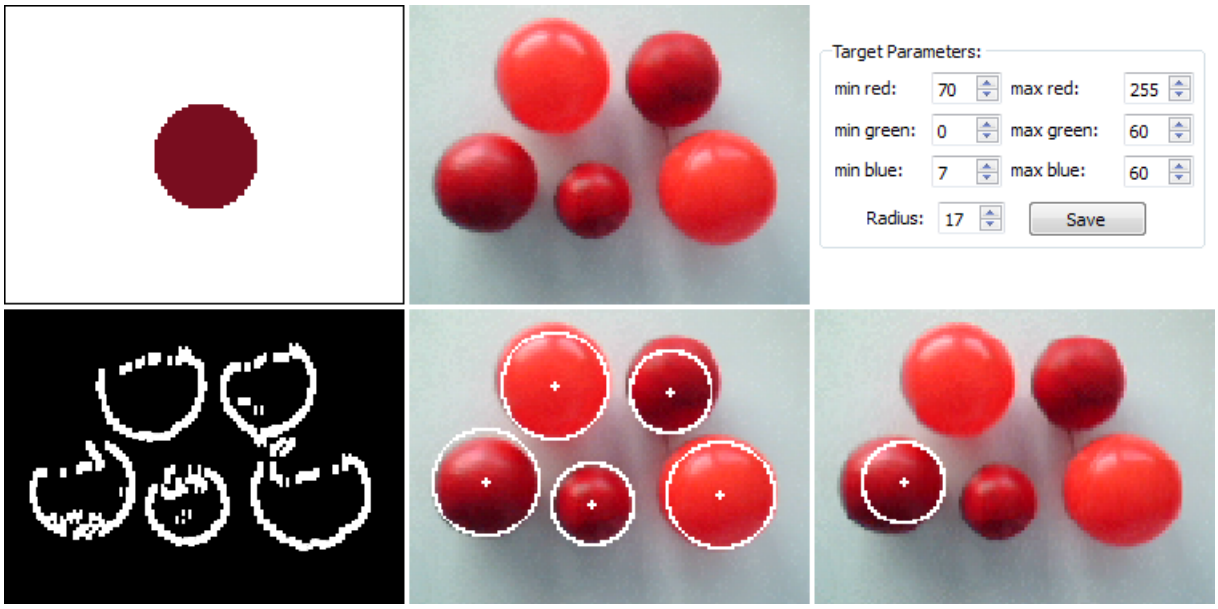

Abbildung 37: Auswirkungen des benötigten Mindest-Prozentsatzes des Kreisumfangs:
oben: Scanergebnis, Suchbild, Such-Parameter
unten: Ergebnis der Kantendetektion, Suchergebnis mit Wert 0,30, mit Wert 0,35

Wird eine Zelle gefunden, die einen ausreichend hohen Kantenwert enthält, wird überprüft, ob der daraus resultierende Kreis der „beste“ in diesem Bereich ist. Ist der Kreis-*Vector* noch leer, wird der aktuelle Kreis direkt dort gespeichert. Sind bereits Kreise vorhanden, wird für jeden Kreis im *Vector* geprüft, ob er sich mit dem zu untersuchenden Kreis schneidet. Ist dies bei keinem der Fall, wird der neue Kreis dem *Vector* hinzugefügt. Schneiden sich zwei oder mehr Kreise, wird der neue Kreis nur dann hinzugefügt, wenn er prozentual zum jeweiligen Radius mehr Kanten-Punkte enthält als jeder der anderen Kreise.

4.2.5. Auswertung der Kreise, Zeichnung und Abbruch

Area-Test:

Der im Such-Algorithmus erstellte *Vector* enthält Kreise, die bereits das erste Qualitätskriterium, das der Mindestanzahl von Kantenpunkten, erfüllen. Zum Abschluss werden die Kreise im Area-Test darauf untersucht, ob sie genügend Pixel der gesuchten Farbwerte enthalten. Dies ist nötig, da nur so entschieden werden kann, ob ein gefundener Kreis durch ein Zielobjekt entstanden ist oder ob lediglich eine gewisse Anzahl zufälliger Kanten der Kontur eines Zielobjekts nahekommt. Dargestellt ist diese Problematik in Abb. 38.

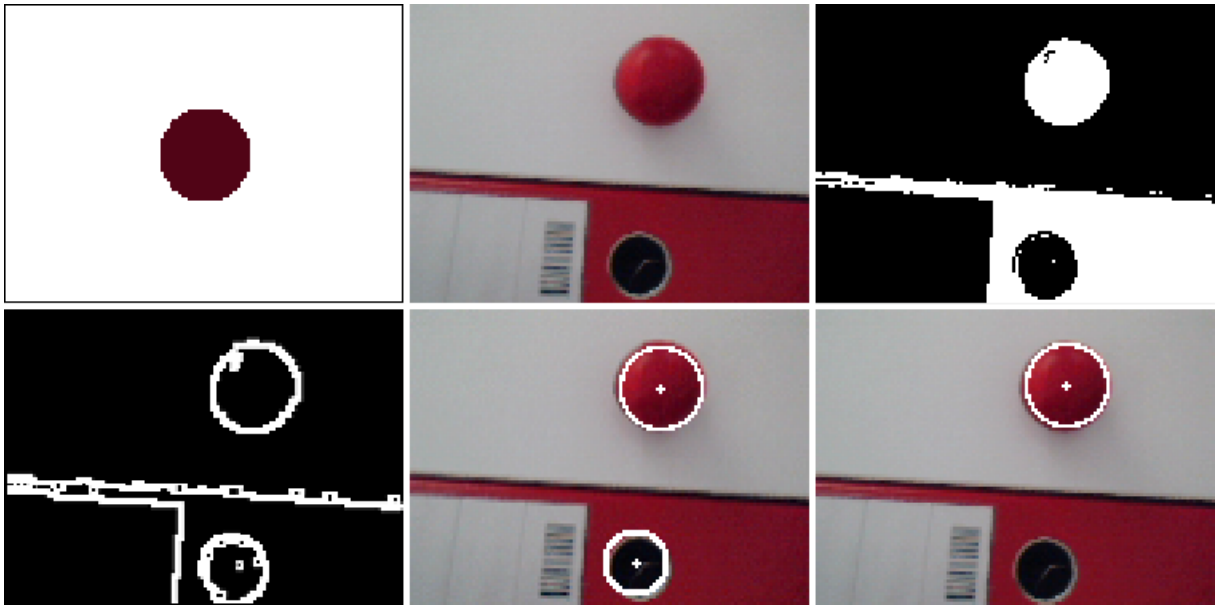

Abbildung 38: Auswirkungen des Area-Tests:
oben: Scanergebnis, Suchbild, Binärdarstellung
unten: Ergebnis der Kantendetektion, Suchergebnis ohne Area-Test, mit Area-Test

Das Loch des Ordners ist dabei von der zu suchenden Farbe umgeben. Es fällt auf, dass dies zwar in der Binärdarstellung noch deutlich zu erkennen ist, es jedoch in der Darstellung der Kantendetektion keinen Unterschied mehr macht, ob gesuchte Farbe von nicht gesuchter Farbe umgeben war oder umgekehrt. Ohne den Area-Test wird also das Loch als Zielobjekt klassifiziert, obwohl es keinen einzigen Pixel mit korrekten Farbwerten enthält. In der Praxis treten solche Extremfälle nur sehr selten auf. Dennoch ist es möglich, dass Strukturen der gesuchten Farbe, wie z.B. das Muster eines Teppichs, genügend auf einem Kreis liegende Kantenpunkte liefern, um als mögliches Zielobjekt detektiert zu werden. Aus diesem Grund wird jeder gefundene Kreis zuletzt dem Area-Test unterzogen. Dessen Vorgehensweise ist analog zum bereits vorgestellten Radius-Test. Die minimale Anzahl Pixel $\#P_{min}$, die ein Kreis mit Radius *R* besitzen muss, berechnet sich aus:

$$\#P_{min} = R^2 * \pi * A_{min} \tag{4.6}$$

A_{min} ist dabei ein variabler Wert, der festlegt, wie viel Prozent der Fläche des Kreises mit Zielpixeln bedeckt sein müssen. Aufgrund der bisher gewonnen Erkenntnisse erweist sich für diesen Parameter 0,4 als guter Wert.
Der so berechnete Wert wird mit der tatsächlichen Anzahl von Zielpixeln innerhalb des Kreises verglichen. Dazu wird nicht im Original-Such-Bild jeder im Kreis liegende Pixel erneut überprüft, sondern lediglich die Anzahl der Zielpixel, die sich im Inneren des Kreises befinden, berechnet. Dies wird anhand des Binärbilds durchgeführt, da dort bereits die Zielpixel markiert sind. Ist der so berechnete Wert kleiner als der Mindestwert, wird der Kreis aus dem *Vector* gelöscht.

Zeichnen der Kreise und Abbruch:

Nach dem Prüfen aller Kreise wird die Größe des *Vectors* ermittelt. Ist diese 0, wurde kein Objekt gefunden und der Algorithmus bearbeitet, je nach Einstellung, entweder das nächste Bild oder stoppt. Enthält der *Vector* Kreise, werden die verbliebenen

Kreise unter der Verwendung der Kreisgleichung in das Ursprungsbild eingezeichnet. Danach besteht wiederum die Möglichkeit, den Algorithmus zu stoppen oder mit der Bearbeitung des nächsten Bildes fortzufahren.

4.2.6. Besonderheiten der Implementierungen

Im Folgenden sollen die Gemeinsamkeiten und Unterschiede der beiden Implementierungen untersucht werden. Es ist zu erwähnen, dass sich hierbei größtenteils auf konzeptionelle Unterschiede beschränkt wird. So mag sich z.B. die Vorgehensweise bei der Umwandlung eines Farbbildes in ein Graustufenbild oder beim Herausfiltern bestimmter Farbwerte nicht zu 100% gleichen, jedoch ist das Prinzip dasselbe.

Form der Datenrepräsentation:

Während in der OpenCV-*Library* Bilder grundsätzlich im Mat-Format gespeichert werden, verwendet die eigene Implementierung für die Berechnungsschritte größtenteils *unsigned char*-Matrizen.
Das Mat-Konzept bietet vielfältige Möglichkeiten, eine Menge an Datentypen darzustellen. Die Matrix kann sich z.B. aus 32-Bit-*float*-Werten (*CV_32FC1*), aus *unsigned int* 8-Bit-Werten (*CV_8UC1*) oder zahllosen anderen Elementen zusammensetzen. Dabei muss ein Matrix-Element nicht zwingend aus einem einzelnen primitiven Datentyp bestehen. Auch eine festgelegte Menge dessen kann pro Matrix-Zelle gespeichert werden. Dies ist beispielsweise im *CV_8UC3*-Format der Fall, in dem jede Zelle ein Tripel aus *unsigned char*-Werten beschreibt. Dies stellt wiederum eine günstige Repräsentation eines Farbbildes dar [OpenCV 2008].
Im Ansatz der eigenen Implementierung wird dies mittels des bereits erwähnten *Structs Rgb* gelöst. Für alle anderen Repräsentationen und Rechenschritte reicht die normale *unsigned char*-Darstellung vollkommen aus. Das *Mat*-Format bietet zwar weitaus vielfältigere Möglichkeiten der Darstellung, aber da in dieser Arbeit ausschließlich mit 24-Bit-Farbbildern, deren zugehörigen Graustufenbildern und Binärbildern gearbeitet wird, sind, abgesehen von dem eigens erstellten *Rgb-Struct*, keine weiteren Anpassungen nötig, um die volle Funktionalität des Algorithmus zu gewährleisten.

Kantendetektion und Kreissuche:

In der eigenen Implementierung kommen die in 2.6. und 2.7. theoretisch vorgestellten Konzepte der Kantendetektion mittels Sobel-Operatoren und der Hough-Kreis-Transformation größtenteils in ihrer klassischen Version zum Einsatz. Abweichungen wurden in Abschnitt 4.2.3. bereits erläutert. Die Kreis-Detektion der OpenCV-*Library* verfolgt einen etwas anderen Ansatz. Hier findet die in 2.7.5. vorgestellte Hough-Gradienten-Methode Verwendung.

4.2.7. Die Benutzeroberfläche (Gui)

Abb. 39 zeigt die Benutzeroberfläche des QT-Steuerprogramms nach dem Einfügen dieser Arbeit. Die Reiter am oberen Rand ermöglichen die Kommunikation mit dem Quadrocopter. Dabei können sowohl Befehle gesendet als auch Daten empfangen werden. Neu hinzugefügt wurde der Reiter *Objekterkennung*, der sich wiederum in drei weitere Reiter untergliedert, auf deren Aufbau und Funktion im Folgenden kurz eingegangen werden soll.

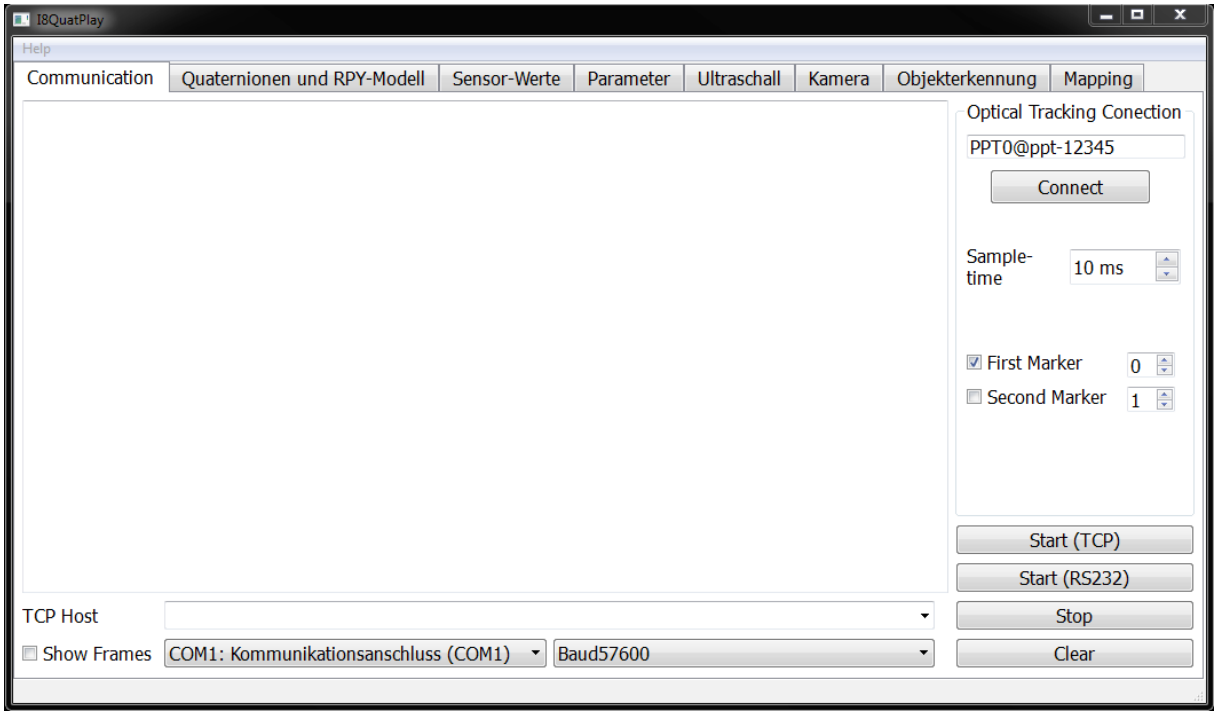

Abbildung 39: Benutzeroberfläche des QT-Steuerprogramms

4.2.8. Der Reiter *Scan*

In Abb. 40 ist ein erfolgreich durchgeführter Scan eines roten Balls zu sehen.
Auf der linken Seite werden die Einzelschritte des Scans visualisiert. *ScanTarget* zeigt das zu scannende Bild, *GrayScale* das zugehörige Graustufenbild, *Edges* das Ergebnis der Kantendetektion (nur eigene Implementierung) und *ScanResult* das Ergebnis des Scans, d.h. das zu suchende Objekt.
Am rechten Rand bietet sich die Möglichkeit, aus einem gespeicherten Bild (*Load Image From File*) oder einem neu aufgenommenen Kamerabild (*Take Picture*) mittels eines Scans (*Perform Scan*) die Farb-Parameter und den Radius des sich im Bild befindlichen Objektes zu erhalten. Dabei kann per *Checkbox* zwischen der OpenCV- und der eigenen Implementierung gewählt werden. In der *Spinbox ToleranceScale* kann der aus 4.2.3. bekannte Wert T_S dimensioniert werden. Die so erhaltenen Parameter werden in den *Spinboxen* rechts angezeigt und können dort bei Bedarf manuell verändert und gespeichert werden (*Update*). Ist das gewünschte Scanergebnis erreicht, kann mit der Suche fortgefahren werden. Des Weiteren ist es mit *Save Picture* möglich, ein bestehendes *ScanResult* im *Bitmap*-Format abzuspeichern und später wieder zu laden. Die Dauer des gesamten Vorgangs wird im *LCD-Number*-Feld *Time elapsed* angegeben.

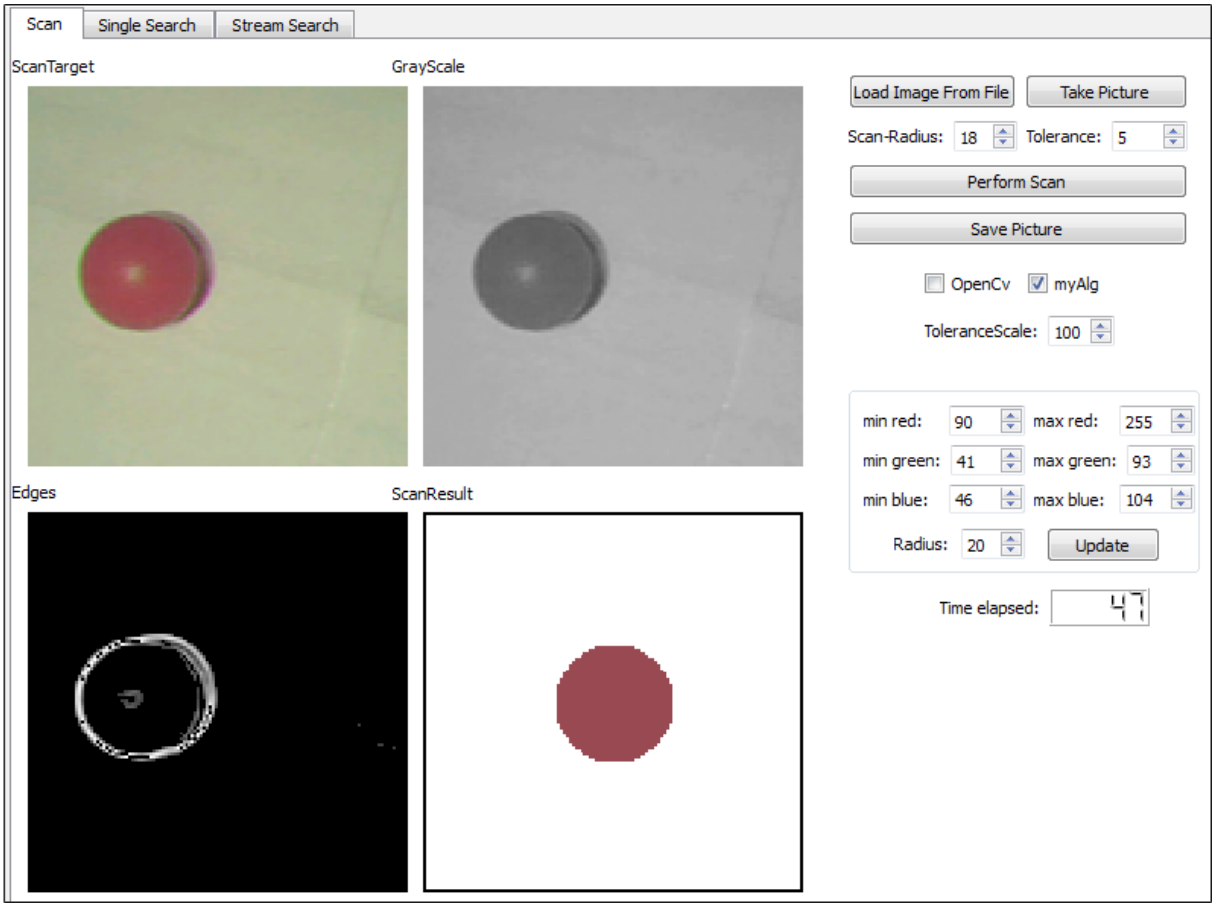

Abbildung 40: Scan eines roten Balls unter Verwendung der eigenen Implementierung

4.2.9. Der Reiter *Single Search*

Der Aufbau des Reiters *Single Search* ist analog gestaltet. Abb. 41 zeigt eine erfolgreiche Suche nach den in Abb. 40 festgestellten Parametern. Wiederum kann ein Bild aus einer Datei geladen oder direkt per Kamera aufgenommen werden. Dieses wird unter *SearchTarget* angezeigt. S*canTarget* enthält das Scanergebnis aus dem *Scan*-Reiter und wird beim Scannen, ebenso wie die Such-Parameter (*Target Parameters)*, automatisch übernommen. Diese können wiederum manuell verändert werden. Die *Checkboxen OpenCV* und *myAlg* bestimmen, welche Implementierung ausgeführt wird. Wird die OpenCV-Implementierung gewählt, kann mittels der *Spinboxen* Hough Param. 1 (*HP1*) und Hough Param. 2 (*HP2*) die Empfindlichkeit bei der Kreiserkennung beeinflusst werden. Durch die *Checkboxen Binary* und *Edges* (*myAlg*) bzw. *Grayscale* und *Binary* (*OpenCV*) können die jeweiligen Bilder der Verarbeitungsschritte zu- oder abgeschaltet werden. Gleiches gilt für das *Label SearchResult*, in dem das Originalbild angezeigt und, falls die Suche erfolgreich war, das gefundene Objekt markiert wird. In der *Spinbox Time elapsed* wird die vergangene Zeit vom Klicken des *Perform Search–Buttons* bis hin zur Ausgabe des Resultats in Millisekunden ausgegeben.

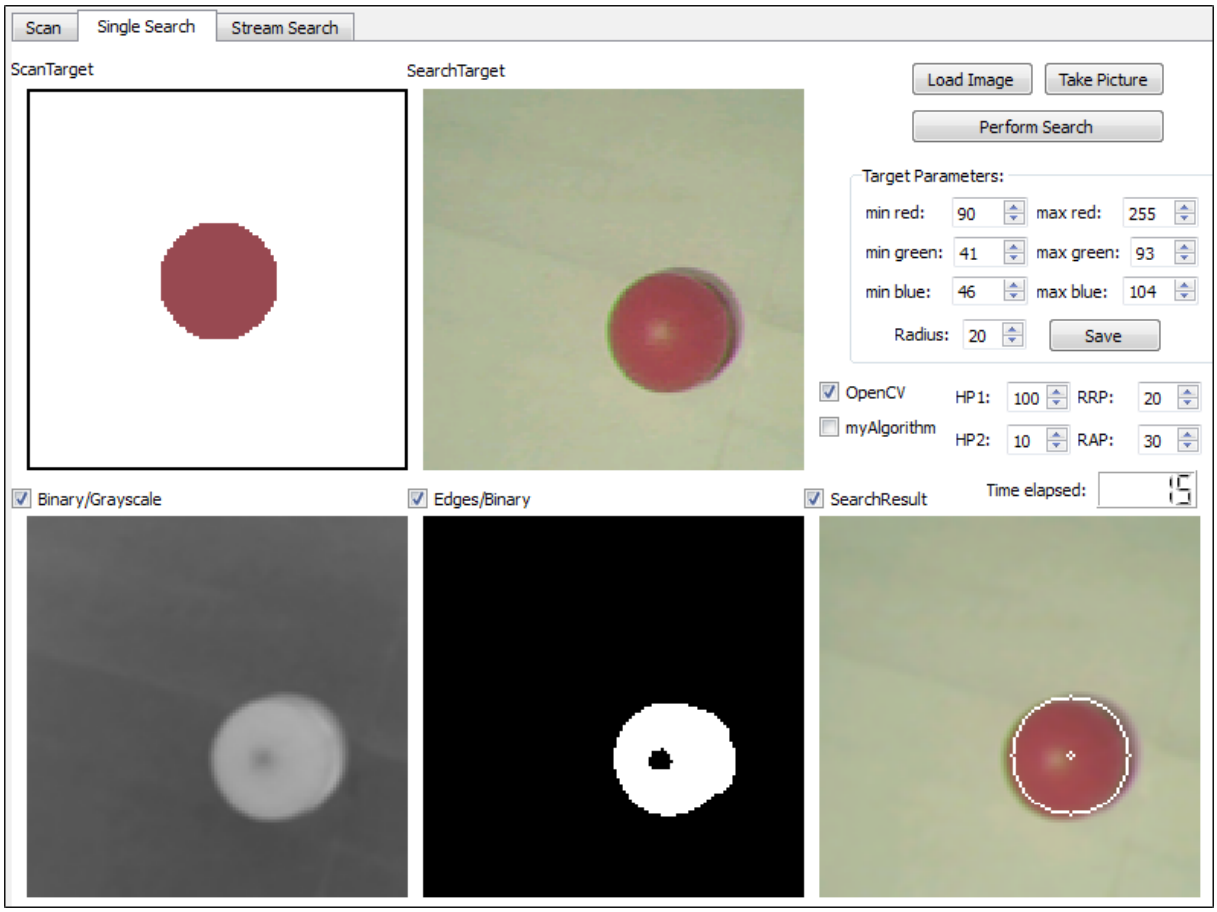

Abbildung 41: Erfolgreiche Suche nach einem roten Ball unter Verwendung der OpenCV-Implementierung

4.2.10. Der Reiter *Stream Search*

Der letzte Reiter *Stream Search* ermöglicht die Suche eines zuvor gescannten Objekts in einem fortlaufenden Stream von Einzelbildern. Hierzu wird eine Kamera benötigt. Der Aufbau des Reiters ähnelt dem des *Single Search* – Reiters stark. Der Hauptunterschied besteht darin, dass unter *Binaray/Grayscale* und *ShowStream* ein fortlaufender Stream zu sehen ist. Ist die Checkbox *Stop When Found* nicht angewählt, zeigt *SearchResult* fortlaufend den Eingangsstream und markiert gefundene Objekte mittels eines weißen (255, 255, 255) Kreises. Bei Aktivierung der *Checkbox* stoppt der Stream, sobald erstmalig ein Objekt gefunden wurde. Dies ist in Abb. 42 zu sehen. Werden ein oder mehrere Objekte gefunden, erscheint, je nach Anzahl der Objekte, rechts des Streams eine Anzeige *1 Object found*, *2 Objects found* usw..

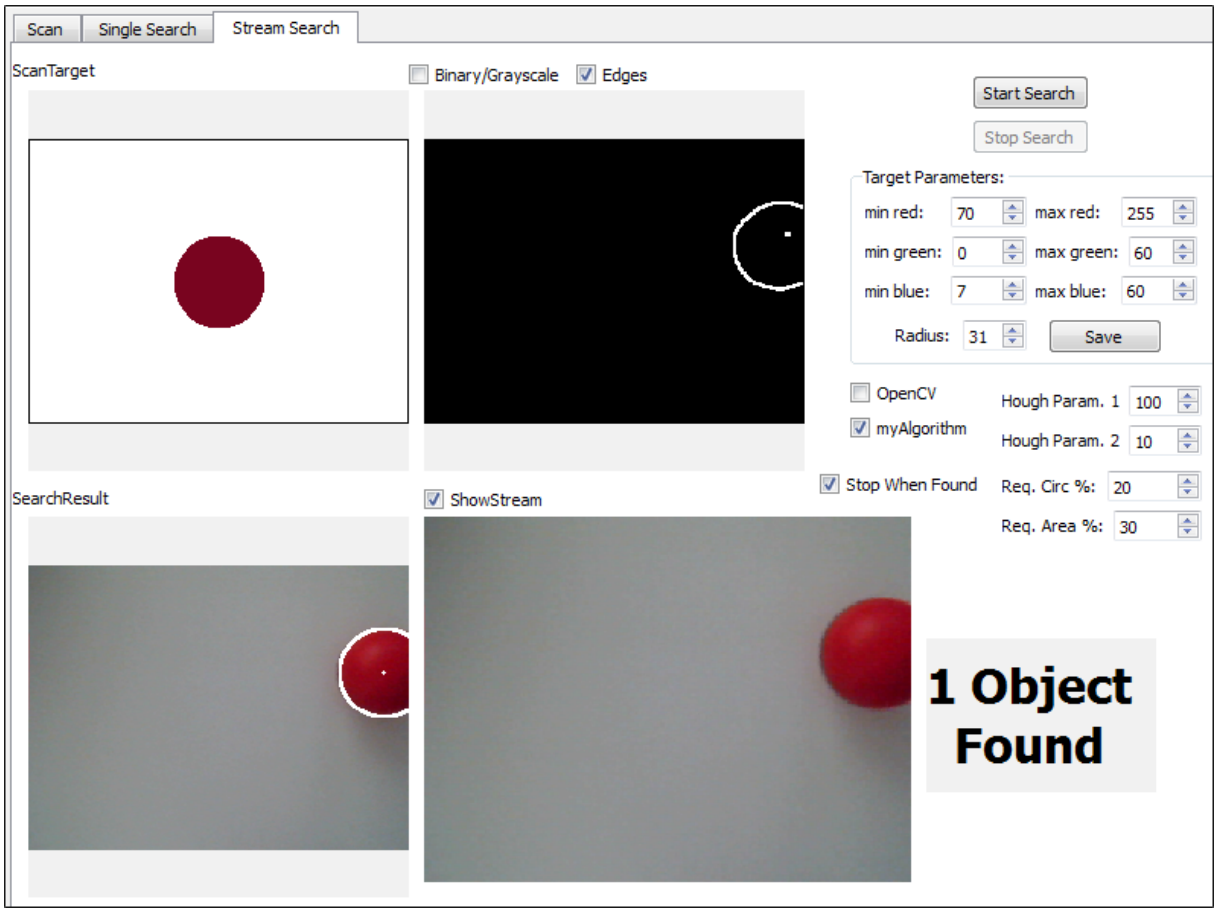

Abbildung 42: Erfolgreiche Suche bei aktivierter Stop When Found – Checkbox

5. Evaluierung

Im Folgenden soll die Leistungsfähigkeit des Systems näher untersucht werden. Ziel ist es, einen Überblick über die Stärken und Schwächen des Systems zu geben. Dadurch soll aufgezeigt werden, was das System zu leisten imstande ist und was nicht. Es werden sowohl die OpenCV-, als auch die eigene Implementierung ausführlich getestet. Als Kameras dienen jeweils die beiden Webcams Logitech C200 und C270, im weiteren Verlauf abgekürzt durch C200 und C270. Auf die Nutzung der OV7670 wird verzichtet, da sie sich durch die umständliche Bedienung und unflexible Auflösung als nicht geeignet erwies.
Zunächst findet eine Evaluierung unter Laborbedingungen, d.h. unter konstant guten Lichtverhältnissen und im Stillstand des Systems statt. Dabei werden auch die Auswirkungen unterschiedlicher Auflösungen und Radiusbereiche untersucht. Anschließend erfolgen Tests bei extremen Lichtverhältnissen, d.h. bei starker oder sehr geringer Beleuchtung. Es folgt eine Untersuchung des Einflusses bewegter Zielobjekte, in diesem Fall rollende Bälle, auf das Suchergebnis. Zum Abschluss wird das System in einen Quadrocopter integriert und seine Eigenschaften im Flug evaluiert.

5.1. Vergleich der Implementierungen unter Laborbedingungen

Bevor ein quantitativer Vergleich auf Basis der Laufzeiten der beiden Implementierungen erfolgt, soll zunächst deren Leistung unter Laborbedingungen qualitativ bewertet werden.

5.1.1. Versuchsaufbau

Es wird eine höhenverstellbare Tafel verwendet, an deren Unterkante die jeweilige Webcam mit Blick nach unten befestigt wird. Abb. 43 zeigt den verwendeten Aufbau.

Die am linken Rand der Tafel erstellte Skala ermöglicht es, durch Verschieben der Tafel unterschiedliche Höhen von 60cm bis etwas mehr als 150cm zu simulieren. Als Zielobjekte dienen fünf rote Bälle. Die verwendeten Parameter sind auf eine Höhe von etwa 90cm bis 100cm und eine Auflösung von 256x192 optimiert und werden im Laufe des Versuchs nicht verändert.

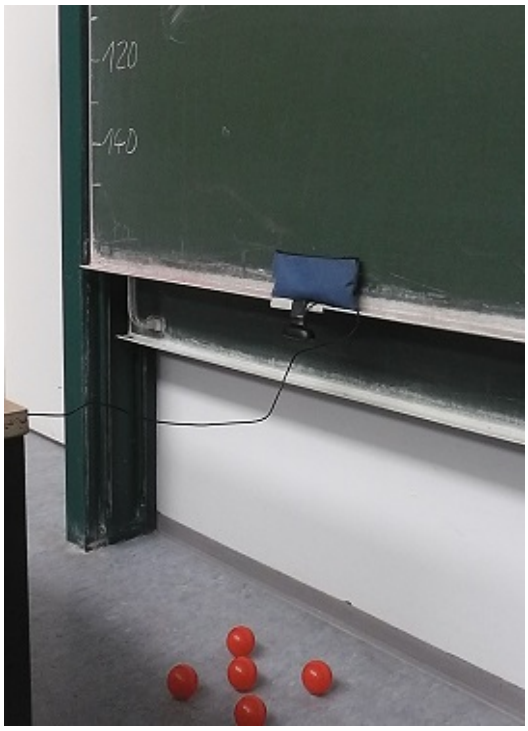

Abbildung 43: Versuchsaufbau - Laborbedingungen

5.1.2. Verwendung der OpenCV-Implementierung

Es erfolgt eine sichere Detektion aller Bälle in einer Höhe von etwa 70cm bis 120cm. Während in diesem Bereich durchgehend fünf Bälle detektiert werden, treten darüber, bis etwa 150cm, starke Schwankungen auf. Es werden im ständigen Wechsel drei bis fünf Bälle erkannt. Knapp unterhalb von 60cm kommt es zu ersten Doppeldetektionen. Dabei werden z.B. in einem Kreis mit einem Durchmesser von 20 Pixeln zwei Kreise mit einem Durchmesser von 10 Pixeln detektiert. Bei einer Höhe unterhalb von 50cm ist der Bildbereich nicht mehr groß genug, um alle fünf Bälle zu erfassen.

In allen Bereichen variiert die Größe der detektierten Kreise zum Teil stark. Dies führt teilweise zu schlecht an das detektierte Objekt angepassten Radien, wie in Abb. 44 rechts unten zu sehen ist. Im Bereich von etwa 70cm bis 120cm treten diese Änderungen etwas weniger häufig in Erscheinung.

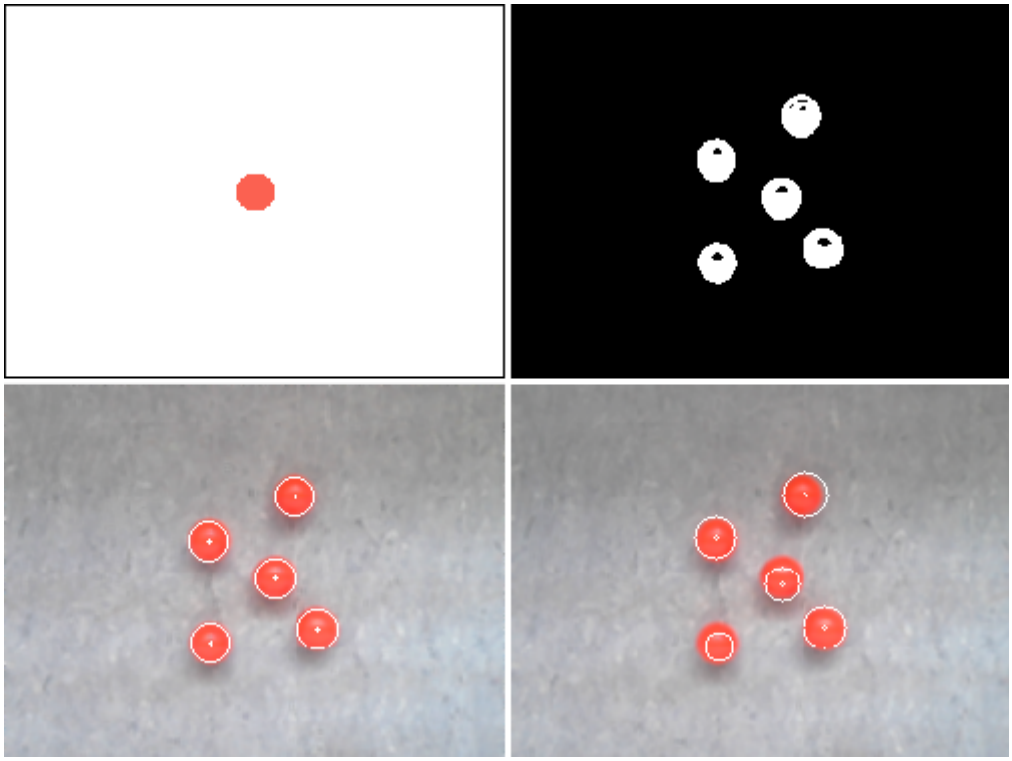

Abbildung 44: Vergleich der erzielten Detektionen der beiden Implementierungen unter Laborbedingungen:
oben: Scanergebnis, Binärbild
unten: Detektionen der eigenen Implementierung, Detektionen der OpenCV-Implementierung

5.1.3. Verwendung der eigenen Implementierung

Alle fünf Bälle werden im Bereich von 70cm bis knapp 130cm zuverlässig detektiert. Oberhalb treten, ähnlich wie bei der OpenCV-Implementierung, schwankende Detektionen zwischen vier und fünf Bällen auf. Unterhalb einer Höhe von 60cm erfolgen die bereits erwähnten Doppeldetektionen. Diese treten etwa dreimal

häufiger als bei der OpenCV-Implementierung auf.

Die Radien sind insgesamt deutlich besser an die detektierten Objekte angepasst. Das oben erwähnte ständige Variieren der Größe der detektierten Kreise tritt wesentlich seltener auf. Gerade im Bereich von 70cm bis 120cm erfolgt eine sehr gute, stabile Detektion, wie auch in Abb. 44 zu sehen ist.

5.1.4. Vergleich der beiden Kameras C200 und C270

Bei Nutzung der C200 fällt der aufgenommene Bildbereich etwas kleiner aus als bei der C270. Dies führt u.a. zu einer kleineren Darstellung der Bälle, die eine Detektion erschwert. Gezeigt ist dies in Abb. 45.

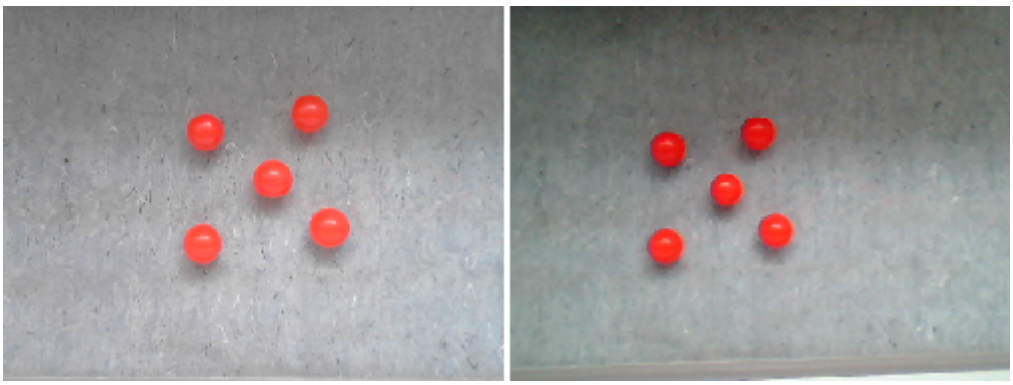

Abbildung 45: Vergleich des Bildausschnitts der beiden verwendeten Webcams:
links: C270
rechts: C200

Beide Aufnahmen erfolgten aus einer Höhe von 100cm bei einer Auflösung von 256x192 Pixeln.

Bei den oben erwähnten Tests ähnelt das Verhalten der C200 stark dem der C270. Durch den kleineren Bildausschnitt müssen lediglich alle Höhenangaben etwa 10cm nach unten korrigiert werden. Insgesamt lässt sich eine etwas schlechtere Detektion der Bälle in allen Bereichen feststellen. Besonders im Bereich oberhalb von 100cm treten häufiger negative Fehler und stärkere Änderungen des detektierten Radius auf. Insgesamt ist die C200 unter den hier auftretenden Bedingungen schlechter für die Detektion geeignet. Aus diesem Grund wurden die folgenden Tests mit verschiedenen Auflösungen nur noch mit der C270 durchgeführt.

5.1.5. Fazit: Vergleich OpenCV mit eigener Implementierung

Die Leistungsfähigkeit beider Implementierungen ist als hoch zu bezeichnen. Besonders in dem Bereich, für den die Parameter optimiert wurden (ca. 80cm bis 120cm), lassen sich sehr gute Ergebnisse erzielen. Negative Fehler treten nicht auf. Dabei liefert die eigene Implementierung eine stabilere Detektion mit etwas besser an die wahre Größe des Objekts angepassten Radien. Die OpenCV-Implementierung besitzt leichte Vorteile außerhalb des oben genannten Bereichs.

5.1.6. Variation der Auflösung

160x140:
Die Qualität der Detektion lässt oberhalb von 70cm bis 80cm stark nach. Speziell für die Detektion kleiner Objekte erscheint diese Auflösung eher ungeeignet.

192x144:
Zwischen dieser Auflösung und einer von 256x192 lässt sich kaum ein Unterschied feststellen. Das Verhalten entspricht dem unter den Abschnitten 5.1.2. und 5.1.3. Beschriebenem, wenn alle Höhenangaben um etwa 10cm nach unten korrigiert werden.

Für höhere Auflösungen kann ausschließlich die OpenCV-Implementierung sinnvoll eingesetzt werden, da es bei der eigenen Implementierung bei der Erstellung des Akkumulator-Raums zu einem Speicherüberlauf und einem Absturz kommt.

320x240 (OpenCV only):
Diese Auflösung ermöglicht eine sichere Detektion der Bälle in einem Bereich von etwa 90cm bis 150cm Höhe. Dabei ist durchaus noch ein flüssiger Stream möglich. Für einen Einsatz bei größerer Flughöhe stellt diese Auflösung eine sinnvolle Alternative dar.

520x390 (OpenCV only):

Eine sichere Detektion der Bälle ist auch noch oberhalb von 150cm möglich. Jedoch sind die Radien den gefundenen Objekten deutlich schlechter angepasst und die Framerate sinkt merklich ab.

720x540 und darüber (OpenCV only):
Ein flüssiger Stream ist nicht mehr möglich.

5.2. Einfluss der Lichtverhältnisse auf die Detektion

Da die Lichtverhältnisse große Auswirkungen auf die Farbwerte eines Objekts und damit auf dessen Detektierbarkeit nehmen, soll im Folgenden dieser Einfluss näher untersucht werden.

5.2.1. Versuchsaufbau

Abb. 46 zeigt den verwendeten Versuchsaufbau. Hierbei befinden sich sowohl links als auch rechts jeweils vier Bälle. Es ist deutlich zu erkennen, dass bei Messpunkt 1 wesentlich weniger Lichteinstrahlung vorliegt als bei Messpunkt 2. Offensichtlich hat der Ort des Scans großen Einfluss auf die Detektierbarkeit der Objekte in den jeweiligen Bereichen. In diesem Versuch wird daher der Scan etwa mittig zwischen den beiden Messpunkten ausgeführt, um eine ähnlich große Abweichung der Farbwerte zu gewährleisten. Anschließend wird die jeweilige Kamera ca. 80cm bis

100cm über die beiden zu untersuchenden Positionen geführt.

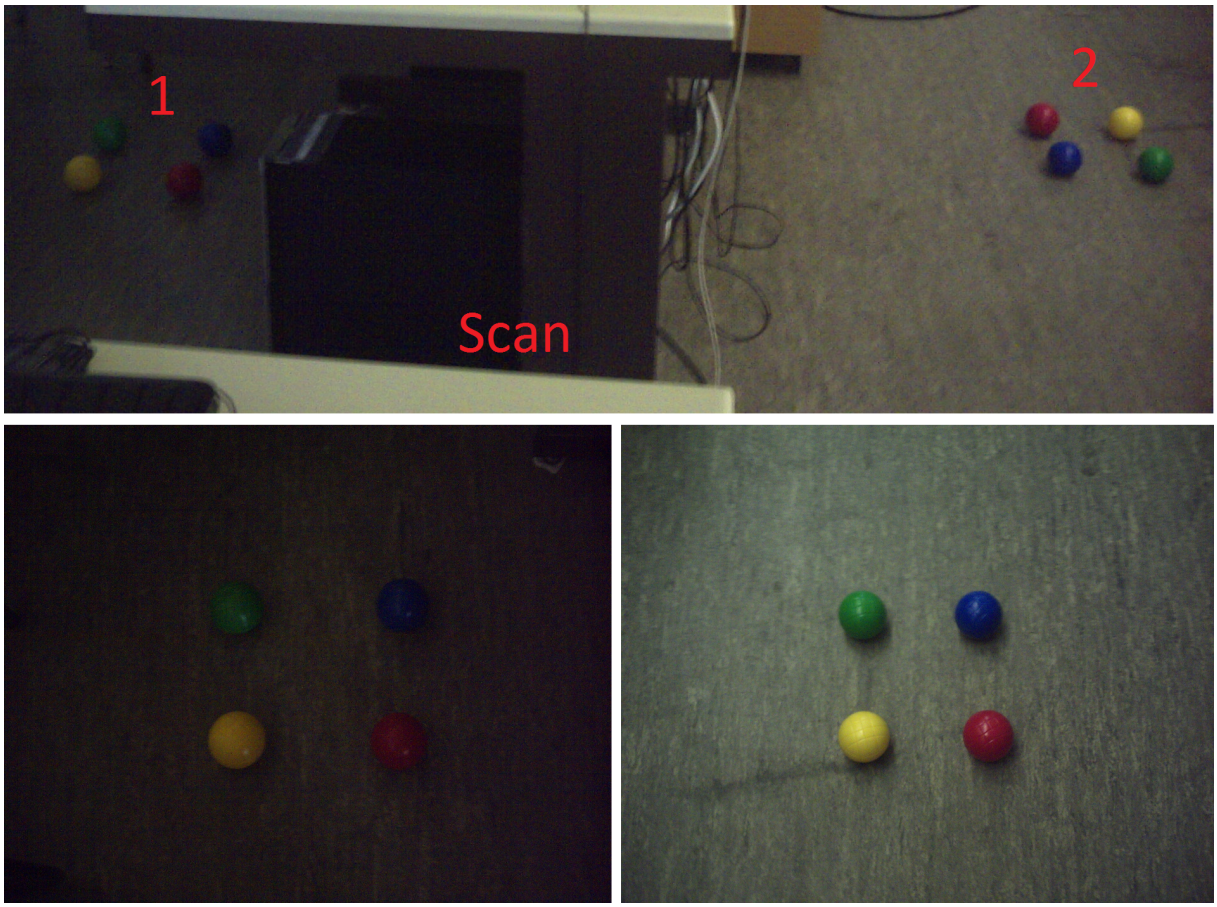

Abbildung 46:
oben: Versuchsaufbau, um den Einflusses der Helligkeit zu testen
unten: Messpunkt 1 in der Draufsicht, Messpunkt 2 in der Draufsicht

5.2.2. Ergebnisse: Kameravergleich C200 mit C270

Mit der C270 ist die Detektion des roten Balls sowohl im Dunkeln als auch im Hellen sehr gut möglich. Selbiges gilt für den gelben Ball. Bei diesem treten jedoch starke positive Fehler an den weißen Tischplatten und den hellen Bereichen des Bodens auf. Dies ist durch die durchgehend hohen Farbwerte des gelben Balls zu erklären, die sich nur wenig von Weiß unterscheiden. Der blaue Ball ist im hellen Bereich sehr gut detektierbar, wird jedoch im Dunkeln von beiden Implementierungen nicht detektiert. Ein möglicher Grund dafür könnte ein etwas zu hell ausgefallener Scan sein. Der grüne Ball ist mit beiden Implementierungen sehr gut detektierbar und die beste Farbe im Test. Hierfür lässt sich kein offensichtlicher Grund erkennen.
Insgesamt ergibt sich bei der C200 ein deutlich schlechteres Bild als bei der C270. Wirklich gut lässt sich lediglich der grüne Ball im hellen Bereich erkennen. Die Detektion der Bälle im Dunkeln schwankt stark. Gelb und Rot werden nur in etwa 50% der Fälle richtig erkannt, Grün in weniger als 20% und Blau lässt sich gar nicht mehr detektieren.
Dies ist wahrscheinlich darauf zurückzuführen, dass der interne Helligkeitsausgleich der C200 für diese Anwendung deutlich schlechter geeignet ist als der Ausgleich der C270.

5.2.3. Interner Helligkeitsausgleich der Webcams

Durch die Verwendung von handelsüblichen Webcams treten interessante Phänomene auf, da diese über einen internen Helligkeitsausgleich verfügen. Dieser ist in der Lage, Bilder bei zu geringer oder zu starker Helligkeit dementsprechend aufzuhellen oder abzudunkeln. Dies ermöglicht es der Kamera, z.B. eine Person in einem stark abgedunkelten Raum noch gut sichtbar darzustellen.
Bei der Verwendung der Kameras zur Objekterkennung hat dieser Ausgleich sowohl Vor- als auch Nachteile. Zum einen wird es der Kamera ermöglicht, in einer sehr dunklen Umgebung zu operieren. Finden sowohl der Scan, als auch die Suche bei etwa gleichen Lichtverhältnissen statt, sind Suchen in fast völliger Dunkelheit möglich. Zum anderen kann der Helligkeitsausgleich aber dazu führen, dass die Farben unter normalen Lichtverhältnissen so angeglichen werden, dass deutlich erkennbare Objekte nicht mehr detektiert werden können. Besonders stark tritt dieser Effekt beim raschen Übergang zwischen zwei stark unterschiedlich hellen Gebieten auf. Des Weiteren scheinen sich die Webcams bei der Optimierung der Lichtverhältnisse besonders auf die inneren Bereiche des Bildes zu fokussieren, also die Bereiche, in denen sich bei herkömmlicher Anwendung der Kopf des Nutzers befindet. Dies kann z.B. bewirken, dass zwei identische Objekte, eines am Rand des Bildes und eines in der Mitte, bei nur leicht variierenden Lichtverhältnissen an diesen Positionen sehr unterschiedlich detektiert werden.

5.3. Laufzeit der beiden Implementierungen

Die Laufzeit des Suchalgorithmus, d.h. die Zeitspanne, die benötigt wird, um ein Einzelbild auf kreisförmige Objekte der gesuchten Farbe und Größe zu prüfen, spielt eine große Rolle bei der fortlaufenden Detektion im Stream. Je kürzer die Verarbeitung eines Bildes dauert, desto mehr Bilder können pro Zeiteinheit untersucht werden.

5.3.1. Abhängigkeit der Laufzeit von der Auflösung

Versuchsaufbau:

Verwendet werden die in Abb. 47 zu sehenden Anordnungen von Bällen.

Abbildung 47:
links: Scanergebnis
Mitte: Suchbild mit einem Zielobjekt
rechts: Suchbild mit fünf Zielobjekten

Untersucht wird jeweils die Laufzeit des Suchalgorithmus für verschiedene Auflösungen für ein Bild mit einem Ball bzw. fünf Bällen. Die Durchführung erfolgt mit der C270 aus einer Höhe von 80cm. Pro Suchbild, Auflösung und Implementierung werden zehn Messungen vorgenommen und deren Mittelwert gebildet. Alle Messwerte und die sich daraus ergebenden Diagramme sind in Anhang 8.4.1. zu sehen.

Ergebnisse:

Es ist deutlich zu erkennen, dass die OpenCV-Implementierung wesentlich schneller ist als die eigene Implementierung. Dabei steigt die Laufzeit der eigenen Implementierung mit größer werdender Auflösung etwas schneller an als die der OpenCV-Implementierung. Bei nur einem Zielobjekt im Bild unterscheidet sich die Laufzeit in etwa um das dreifache. Interessant ist auch, dass sich die Laufzeit der OpenCV-Implementierung kaum erhöht, wenn die Anzahl der Zielobjekte von eins auf fünf steigt. Bei der eigenen Implementierung ist dagegen ein deutlicher Anstieg zu erkennen.

5.3.2. Abhängigkeit der Laufzeit von der Anzahl zu untersuchender Radien

Versuchsaufbau:

Abb. 48 zeigt die untersuchten Anordnungen von Bällen. Ähnlich wie in Abschnitt 5.3.1. variiert auch hier die Anzahl der Bälle und somit die der Kantenpunkte. Als Kamera wird die C270 verwendet. Die Auflösung beträgt 256x192 Pixel.

Abbildung 48:
links: Scanergebnis
Mitte: Suchbild mit einem Zielobjekt
rechts: Suchbild mit zwei Zielobjekten

Es werden wiederum pro Suchbild und Radiusbereich zehn Messungen vorgenommen und deren Mittelwert gebildet. Alle Messwerte und die sich daraus ergebenden Diagramme sind in Anhang 8.4.2. zu sehen.

Ergebnisse:

Wiederum fällt sofort auf, dass die OpenCV-Implementierung einen deutlichen Geschwindigkeitsvorteil hat. Selbst bei der Untersuchung von elf Radien ist diese, in dem Bild mit nur einem Zielobjekt, fast doppelt so schnell wie die eigene Implementierung bei der Untersuchung von nur einem einzigen Radius.

Des Weiteren steigt die Laufzeit bei Verwendung der eigenen Implementierung

deutlich mit der Anzahl der zu untersuchenden Kantenpunkte an. Dem gegenüber bleibt die Laufzeit bei Verwendung der OpenCV-Implementierung nahezu identisch. Hierbei fällt außerdem auf, dass sich eine starke Erhöhung des Radiusbereichs nur wenig in der Laufzeit niederschlägt. So führt eine Erhöhung um 10 Radien, von 11 auf 21, lediglich zu einem Anstieg der Laufzeit um 1,4 ms. Dies entspricht nur etwa 8%.

5.4. Kontrast des Zielobjekts zum Hintergrund

Das Konzept dieser Arbeit sieht vor, die Farbinformationen vor der Kantendetektion auf ein Minimum zu reduzieren. Dies geschieht durch das Erstellen eines Binärbilds, das nur noch zwischen Zielfarbe und Nicht-Zielfarbe unterscheidet. Nach dieser Reduktion spielt es keine Rolle mehr, ob sich ein roter Ball beispielsweise auf einem pinkfarbenen oder einem blauen Untergrund befindet, solange die Farbwerte des Hintergrunds außerhalb des Zielfarben-Bereichs liegen. Abb. 49 verdeutlicht diesen Sachverhalt.

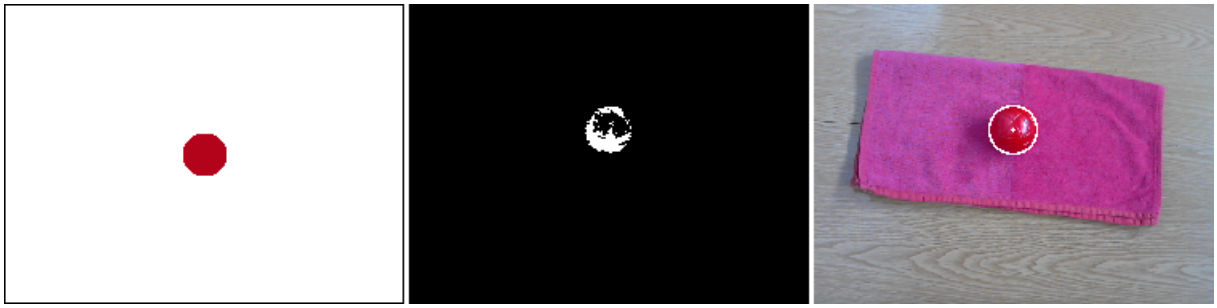

Abbildung 49:
links: Scanergebnis
Mitte: Binärbild
rechts: detektiertes Objekt

Die Farbe des Handtuchs im Hintergrund unterscheidet sich nicht stark von der Zielfarbe, liegt jedoch minimal außerhalb des Zielbereichs. Somit werden die Handtuch-Pixel im Binärbild mit 0 kodiert, erzeugen keine zusätzlichen Kantenpixel, und der Ball kann leicht detektiert werden. Was in diesem Beispiel als großer Vorteil erscheint, kann sich allerdings auch sehr negativ auswirken. Würde z.B. das Handtuch nur leicht andere Farbwerte besitzen, könnten diese innerhalb des Zielbereichs liegen. Die Folgen wären eine Kodierung mit 1 im Binärbild und damit ein Verlust der Kanten des Balls. Somit wäre eine Detektion unmöglich, und das, obwohl sich der Ball durchaus noch sichtbar von seinem Hintergrund abgrenzt.
Eine Möglichkeit dieses Problem zu umgehen bzw. abzuschwächen ist es, die Kantendetektion nicht in einem Binärbild, sondern in einem Graustufenbild durchzuführen. Dabei entstehen Kanten unterschiedlicher Stärke, je nachdem wie sehr sich an die Kante angrenzenden Farben unterscheiden.

5.5. Probleme bei der Formdetektion

Die verwendeten Algorithmen zur Kreisdetektion liefern als Ergebnis nicht zwingend ausschließlich Kreise. Um die Suche robuster gegen äußere Störungen wie z.B. schlechte Lichtverhältnisse, teilweise Verdeckung der Zielobjekte oder eine fortlaufende Bewegung der Kamera zu machen, müssen die Minimum-Parameter wie der minimale Radius, die minimale Fläche oder der Hough-Parameter 1 etwas niedriger angesetzt werden als es für einen idealen Kreis der Fall wäre. Dies kann zu einer Reihe von positiven Fehlern führen. Diese traten insgesamt bei der eigenen Implementierung häufiger und stärker auf als bei der OpenCV-Implementierung.

5.5.1. Detektion nicht kreisförmiger Objekte

Ein Beispiel für die eben erwähnten positiven Fehler ist die Detektion eines Vierecks, in diesem Fall eines Kästchens, als Kreis, wie in Abb. 50 zu sehen.

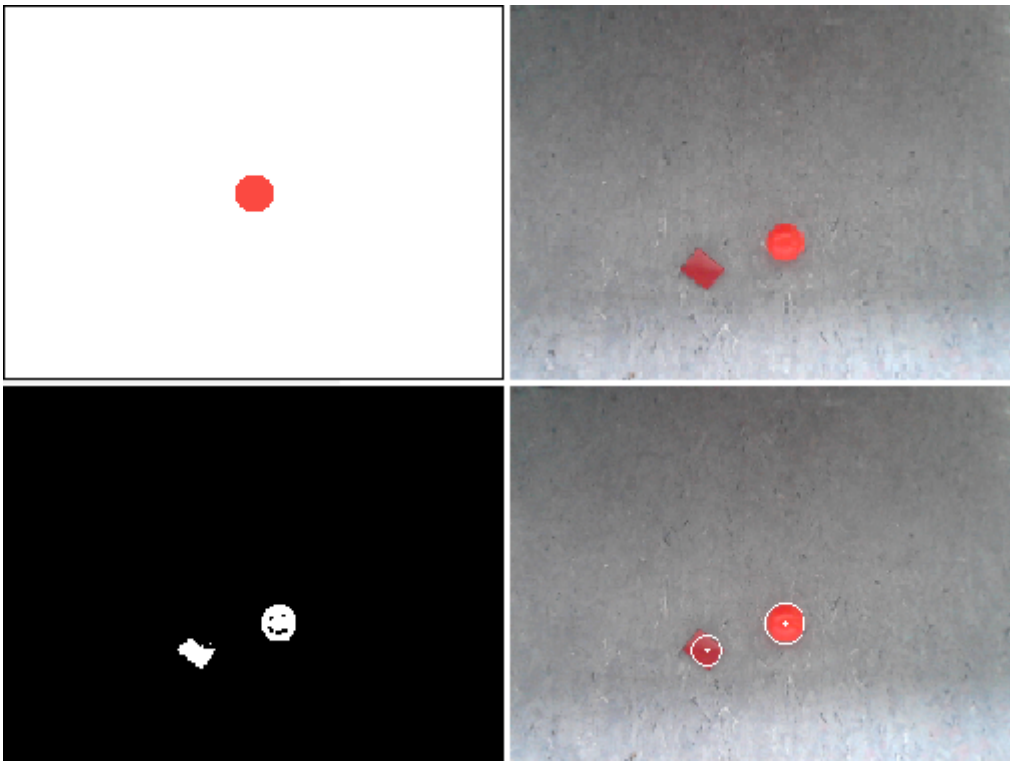

Abbildung 50:
oben: Scanergebnis, Suchbild
unten: Binärdarstellung, Suchergebnis

Während das menschliche Auge das Kästchen klar vom Ball unterscheiden kann, ist dies dem Algorithmus teilweise nicht möglich. Grund dafür sind die hinreichend vorhandenen Kantenpunkte, die ebenso zu einem Kreis gehören könnten, als auch die ausreichende Menge an Zielpixeln innerhalb des gefundenen Kreises. Natürlich könnten diese Parameter für den vorliegenden Fall optimiert und eine Detektion des Kästchens dadurch ausgeschlossen werden, jedoch würde dies in anderen Situationen zu einer schlechteren Detektion führen.

5.5.2. Detektion von Kreisen in einem verrauschten Hintergrund

Eine weitere, wiederholt auftretende Form der Fehldetektion ist in Abb. 51 zu sehen. Diese entspricht dem *worst case*. Im oberen Teil des Handtuchs findet ein ständiger Wechsel zwischen Zielfarbe und Nicht-Zielfarbe statt. Dies führt zum einen zu zahlreichen Kanten, die bewirken, dass Kreise mit ausreichend hoher Kantenzahl gefunden werden, zum anderen sind genügend Zielpixel vorhanden, um den Anforderungen an die Mindestfläche gerecht zu werden.

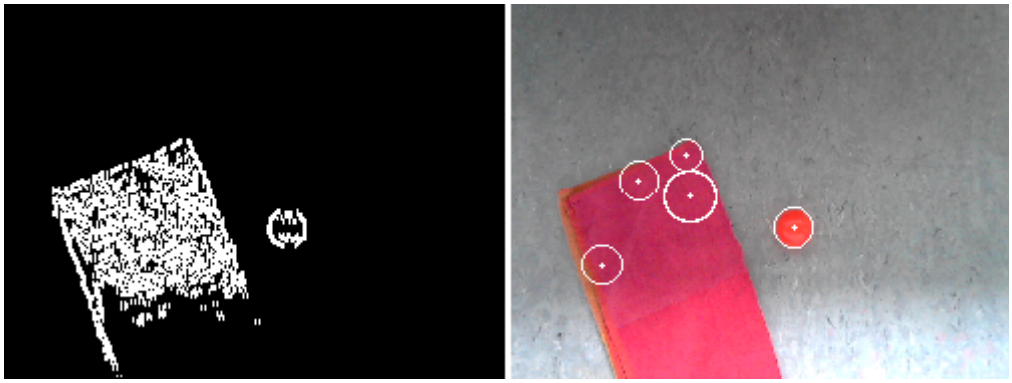

Abbildung 51:
links: Ergebnis der Kantendetektion
rechts: Ergebnisbild

5.5.3. Fehlgeschlagene Lösungsversuche

Für die in Abschnitt 5.5.1. und 5.5.2. beschriebenen Probleme gab es eine Reihe von Lösungsansätzen, die leider erfolglos blieben:
Zum einen wurde versucht, die Umgebung des detektierten Kreises in dessen Bewertung mit einzubeziehen. Dazu wurde ein größerer Kreis um den detektierten Kreismittelpunkt gelegt. Anschließend wurde die Anzahl der Zielpixel innerhalb des detektierten Kreises und innerhalb des Bereichs zwischen den beiden Kreisen ermittelt. Aus dem Verhältnis dieser beiden Werte sollte sich eine Aussage über die Qualität des detektierten Kreises treffen lassen. Hintergrund dafür ist, dass sich bei einem perfekt detektierten Kreis alle Zielpixel im inneren und keine im äußeren Bereich befinden dürften. Dem gegenüber liegen in dem Beispiel aus Abb. 50 die Ecken des Kästchens außerhalb des detektierten Kreises. Auch in dem in Abb. 51 dargestellten Beispiel sollte sich dieses Verhältnis stark von dem eines wahren Kreises unterscheiden.
In der Praxis ergab sich leider ein anderes Bild. So konnte zwar für einzelne Situationen jeweils ein guter Grenzwert für das oben erwähnte Verhältnis ermittelt werden, jedoch führte jeder dieser Werte zu Nicht-Detektionen regulärer Zielobjekte in anderen Situationen.
Ein weiterer Ansatz war bei Verwendung der eigenen Implementierung das Anpassen der Parameter der Hough-Kreis-Transformation. So wurden z.B. beim Inkrementieren der Werte im Akkumulator-Raum Pixel, die exakt zum zu untersuchenden Radius passten, um einen deutlich höheren Wert angehoben. Überdies wurde die Schwelle für den Akkumulator-Wert erhöht, die erreicht werden muss, um einen regulären Kreis zu bilden.
Dies führte dazu, dass Teile von stark verrauschten Bildern, wie in Abb. 51 zu sehen,

zufällig bessere Kreise ergaben als reale Zielobjekte. Bälle, deren Umriss nur leicht durch Schatten o.ä. gestört war, wurden nicht mehr als Kreise erkannt.
Auch eine Kombination der beiden Lösungen führte nicht zum Erfolg. Abhilfe könnte eine deutlich höhere Auflösung oder eine Kantendetektion vor der Umwandlung in ein Binärbild (vgl. Abschnitt 5.4.) schaffen. Auf Letzteres wird in Kapitel 6. Ausblick noch näher eingegangen.

5.6. Detektion sich bewegender Objekte

Im nächsten Versuch soll der Flug eines Quadrocopters über das Zielobjekt simuliert werden, da das jetzige Konzept vorsieht, dass der Quadrocopter die Zielobjekte im Vorbeiflug erkennt. Dabei soll er sich mit einer Geschwindigkeit von etwa 0,5m/s in einer Höhe von etwa 100cm bewegen. Im Folgenden wird die Qualität der Detektion für verschiedene Höhen und Geschwindigkeiten evaluiert.

5.6.1. Versuchsaufbau

Um die Bälle mehrmals mit einer bestimmten Geschwindigkeit das Sichtfeld der Kamera durchqueren zu lassen, wurde der in Abb. 52 zu sehende Versuchsaufbau verwendet.

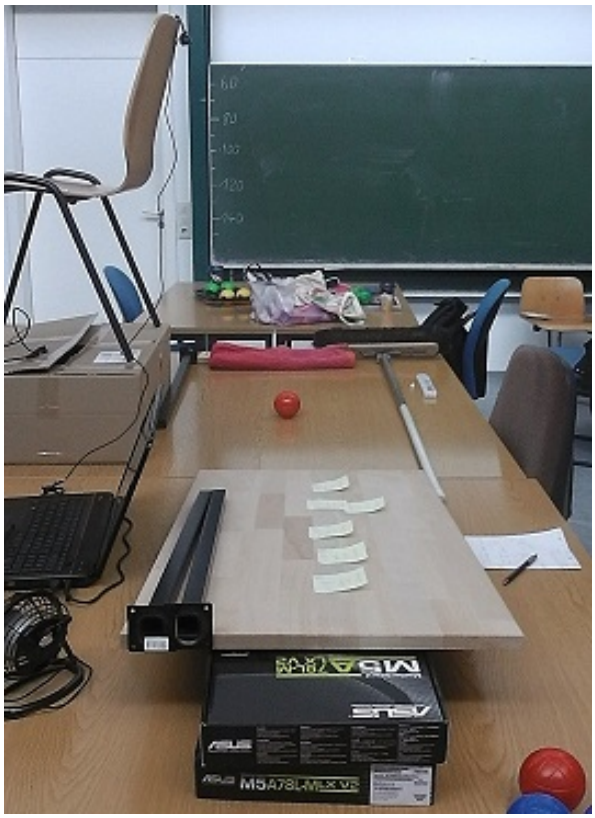

Abbildung 52: Aufbau, um den Vorbeiflug des Quadrocopters zu simulieren

Dabei ist die Kamera an einer Stuhllehne in etwa 100cm Höhe befestigt, da dies der späteren Flughöhe des Quadrocopters entspricht. Die Bälle erhalten die gewünschte Geschwindigkeit, indem sie von einer bestimmten Position auf der Holzschräge aus hinunterrollen. Diese Positionen werden auf der Holzplatte markiert, um zuverlässig mehrere Messungen mit identischer Geschwindigkeit durchführen zu können. Die im Aufnahmebereich der Kamera vorliegende Geschwindigkeit wird dabei per Hand ermittelt, indem die Zeit gestoppt wird, die der Ball für eine Strecke von einem Meter benötigt. Die so erzielte Genauigkeit der Geschwindigkeit ist für die hier durchgeführten Versuche absolut ausreichend. Des Weiteren sei angemerkt, dass sich die Bälle nicht durch den gesamten Aufnahmebereich der Kamera mit konstanter Geschwindigkeit bewegen, da die Reibung des Tisches den Ball abbremst.

5.6.2. Geringe Geschwindigkeit von ca. 0,5m/s

Versuchsdurchführung:

Eine Geschwindigkeit von 0,5m/s entspricht in etwa der vorgesehenen Fluggeschwindigkeit des Quadrocopters bei der Suche. In diesem Test wird die C270 mit einer Auflösung von 256x192 verwendet. Als Zielobjekt dient ein roter Ball, wie er in Abb. 52 unterhalb der Kamera zu sehen ist.

Sowohl für die eigene Implementierung als auch die der OpenCV werden jeweils zehn Messungen durchgeführt. Dabei wird zunächst der Punkt untersucht, an dem der Ball zum ersten Mal detektiert wird. Hierzu wird der x-Wert des detektierten Mittelpunkts als Pixelwert angegeben. Dabei entspricht ein Wert von 255 dem ersten Punkt, an dem eine Detektion möglich ist, und 0 dem letzten. Anschließend wird untersucht, wie oft der Ball bei einem kompletten Durchlauf des Aufnahmebereichs der Kamera detektiert wird.

Ergebnisse:

Die Ergebnisse der Messung sind in Anhang 8.5.1. zu sehen.

Zunächst fällt auf, dass bei beiden Implementierungen eine sehr frühe Erkennung stattfindet. So detektiert die eigene Implementierung im Schnitt den Ball bereits bei Pixelwert 243, was etwa 5% des gesamten Aufnahmebereichs entspricht. Der OpenCV-Implementierung gelingt eine noch etwas frühere Detektion bei durchschnittlich 248 Pixeln, also nach nur etwa 3% des Weges.
In Abb. 53 sind zwei Aufnahmen zum Zeitpunkt der erstmaligen Detektion zu sehen.
Für die frühere Detektion bei Verwendung der OpenCV-Implementierung gibt es eine naheliegende Ursache: Die Laufzeit für die Untersuchung eines Einzelbildes ist deutlich geringer. Dies führt zu einer höheren Framerate und zu einer statistisch höheren Chance, das Zielobjekt früh zu detektieren.
Die höhere Framerate der OpenCV-Implementierung erklärt selbstverständlich auch die größere Anzahl an Detektionen beim Durchlauf des gesamten Aufnahmebereichs der Kamera. Hierbei liegt die Anzahl in diesem Versuch etwa 50% höher als die der eigenen Implementierung.

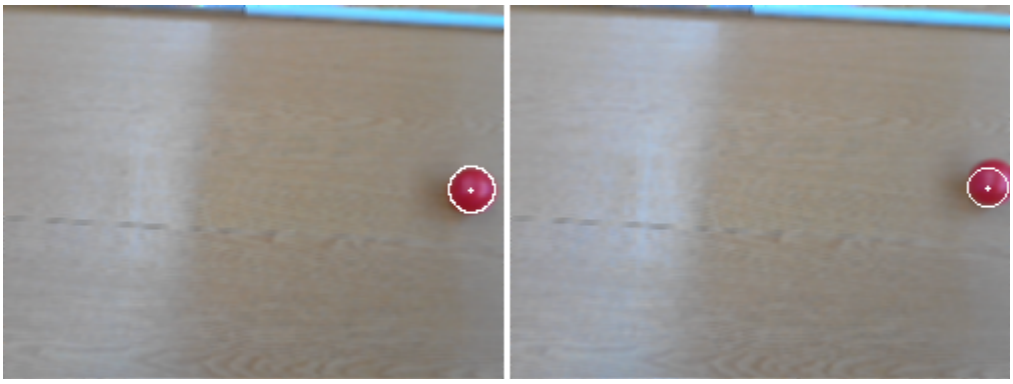

Abbildung 53:
links: Detektion bei Pixel 239
rechts: Detektion bei Pixel 246

5.6.3. Mittlere Geschwindigkeit von ca. 1m/s

Versuchsdurchführung:

Abgesehen von der Verdopplung der Geschwindigkeit entspricht der Ablauf der Messungen dem aus 5.6.2.. Es werden keine Änderungen an Höhe und Auflösung vorgenommen. Die Messungen werden diesmal sowohl mit der C270 als auch mit der C200 durchgeführt.

Ergebnisse:

Die Ergebnisse der Messung sind in Anhang 8.5.2. bzw. 8.5.3. zu sehen.

Wie erwartet, erfolgt die erste Detektion unter Verwendung beider Implementierungen etwas später als bei der Messung mit einer Geschwindigkeit von 0,5m/s. Dabei findet die Detektion jeweils um etwa 1,5% (absolut) später statt. Auch die Anzahl der Detektionen sinkt, wie bereits vermutet, ab. So werden bei doppelter Geschwindigkeit etwa 2 bis 2,4 mal weniger Detektionen durchgeführt.

Bei Verwendung der C200 treten für die Erst-Detektion sehr ähnliche Werte auf wie bei der C270. Die Abweichung ist nicht signifikant. Die Anzahl der Detektionen im gesamten Aufnahmebereich der beiden Kameras weichen mitunter stark voneinander ab. Insbesondere bei Verwendung der OpenCV-Implementierung besteht ein signifikanter Unterschied, der sich zunächst nicht erklären lässt. Ein möglicher Grund für diese Abweichung sind sich während des Versuchs ändernde Lichtverhältnisse. Grund zu dieser Annahme geben die in Tabelle 4 zu sehenden Werte eines Durchlaufs bei Verwendung der C270.

Detektion	1	2	3	4	5	6	7	8	9	10	11	12
Position	237	227	215	207	194	186	173	152	39	18	10	3

Tabelle 4: Orte aller Detektionen bei einem kompletten Durchlauf des Aufnahmebereichs der Kamera

Während im Schnitt etwa alle zehn bis 15 Pixel eine Detektion stattfindet, ergibt sich ein großer Abstand von 113 Pixeln zwischen Detektion acht und neun. Dies entspricht genau dem Bereich, in dem eine deutlich höhere Lichteinstrahlung vorherrschte, wie in Abb. 54 zu sehen ist.

Abbildung 54: Mögliche Störung der Messung durch sich ändernde Lichtverhältnisse

Ähnliche Lichtverhältnisse herrschten auch bei den vorangegangenen Messungen und beeinflussten die Detektionen nicht bzw. weniger stark. Ein möglicher Grund dafür ist ein neu durchgeführter Scan beim Wechsel der Kamera, dessen Farbbestimmung etwas dunkler ausfiel und dadurch zu einer größerer Anfälligkeit gegen direkte Lichteinstrahlung führte.

5.6.4. Bestimmung der Grenzgeschwindigkeit

Um die maximale Geschwindigkeit zu bestimmen, bei der eine Suche noch erfolgreich verlaufen kann, wird die Steigung der in Abb. 52 gezeigten Holzplatte so lange erhöht bis nur noch eine Detektion pro Durchlauf des Balls zu verzeichnen ist. Diese Grenzgeschwindigkeit entspricht ca. 2,5m/s, was in etwa dem Fünffachen der vorgesehenen Fluggeschwindigkeit des Quadrocopters bei der Suche entspricht.

Dabei ist anzumerken, dass diese Geschwindigkeit deutlich von der zur Verfügung stehenden Rechenleistung abhängt. Selbiges gilt für alle durchgeführten Versuche in Kapitel 5.6..

5.7. Echtzeitflug im Quadrocopter

Abschließend wurde die C270 im Quadrocopter verbaut und im freien Flug getestet.

5.7.1. Auswahl der zu verwendenden Kamera

Unter Laborbedingungen erzielen beide Kameras sehr gute Ergebnisse. Der größere Bildausschnitt der C270 ist dabei, gerade bei Höhen oberhalb von 100cm, von Vorteil. Des Weiteren verschlechtert sich das Detektionsverhalten der C200 deutlich, sobald starke Änderungen der Lichtverhältnisse auftreten. Bei der Simulation des Vorbeiflugs eines Quadrocopters ist kein signifikanter Unterschied bei der Leistungsfähigkeit der beiden Kameras festzustellen.

Somit fällt aufgrund des größeren Bildausschnitts und der höheren Toleranz gegenüber Helligkeitsschwankungen die Wahl auf die C270.

5.7.2. Integration des Systems in den Quadrocopter

Wegen der Rechenintensität ist es nötig, einen portablen PC (LP-180) an Board des Quadrocopters zu installieren, der im Flug alle nötigen Berechnungen durchführt. Um die Ergebnisse während des Flugs verfolgen zu können, wird mittels einer W-Lan-Antenne eine Remote-Desktop-Verbindung zu dem auf dem Quadrocopter vorhandenen PC hergestellt. Die C270 Webcam wird in die bereits vorhandene Bodenplatte eingepasst, um einen möglichst senkrechten Blick nach unten zu gewährleisten. Der finale Aufbau ist in Abb. 55 zu sehen.

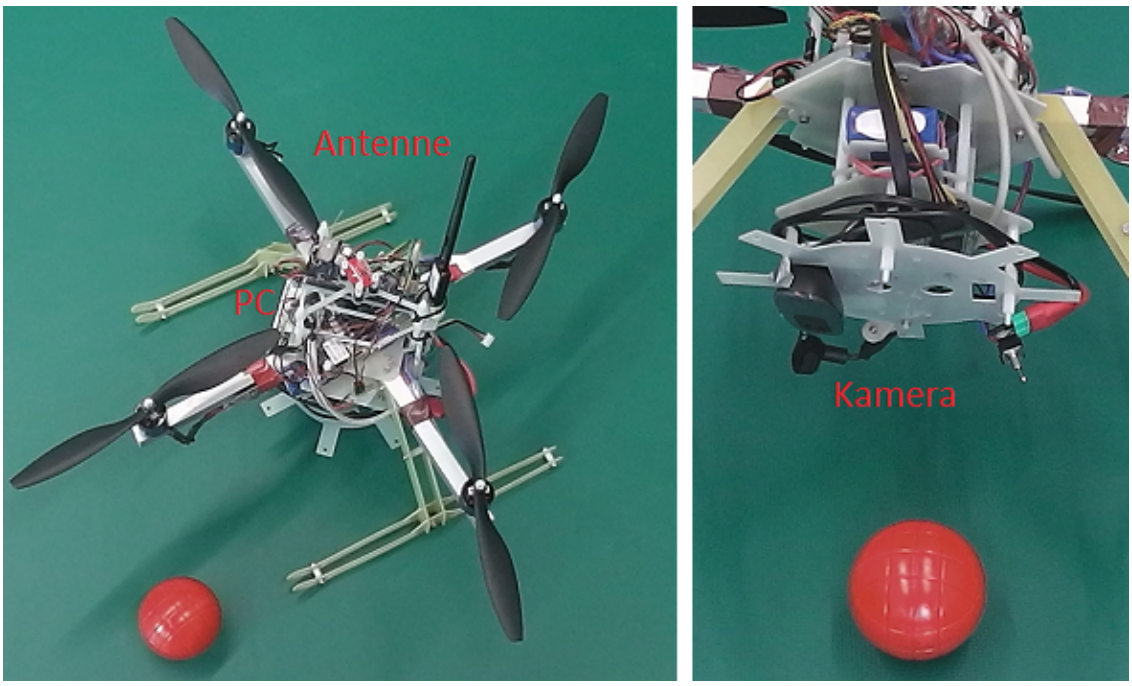

Abbildung 55: Quadrocopter-Setup zur Objekterkennung:
links: W-Lan-Antenne und mittig eingepasster PC
rechts: durch die Bodenplatte nach unten gerichtete Kamera

5.7.3. Versuchsdurchführung

Die Flugversuche wurden im Fluglabor des Lehrstuhls für Informatik VIII der Universität Würzburg durchgeführt (vgl. Abb. 56). Als Zielobjekte dienten rote Bälle, wie in Abb. 55 zu sehen. Diese wurden zusammen mit einigen Bällen anderer Farben verteilt und am Boden befestigt. Dies war nötig, da ansonsten der starke, vom Quadrocopter erzeugte Luftstrom die Bälle stets aus dem Aufnahmebereich der Kamera getrieben hätte. Nun wurde der Quadrocopter per Handsteuerung, d.h. ohne Höhenregelung, über die Bälle geflogen. Dabei variierte die Flughöhe zwischen ca. 50cm und 250cm. Die Geschwindigkeit reichte von etwa 0,5m/s bis hin zu schätzungsweise 3m/s. Per Remote-Desktop-Verbindung wurde der Sichtbereich der Kamera und die darin stattfindenden Detektionen verfolgt.

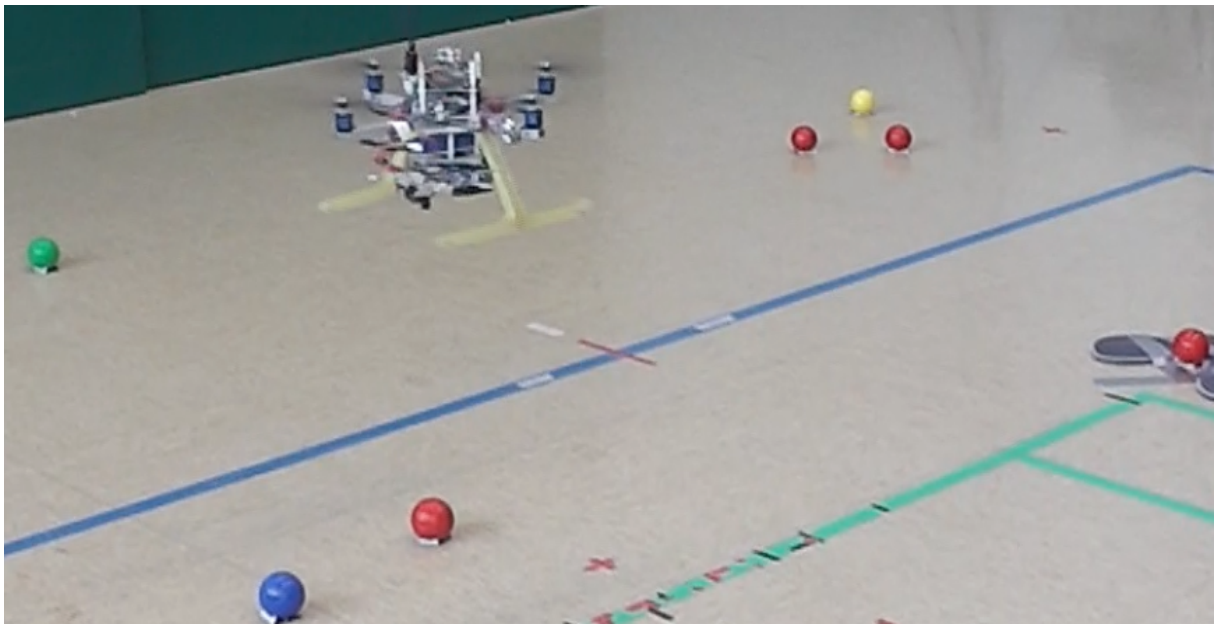

Abbildung 56: Testbereich für den Quadrocopterflug

Es wurden sowohl die eigene, als auch die OpenCV-Implementierung jeweils mit einer Auflösung von 192x144 getestet.

5.7.4. Ergebnisse

Trotz der widrigen Bedingungen konnten im Schnitt mehr als 80% der Bälle erfolgreich detektiert werden. Eine genaue Auflistung der auftretenden Fehldetektionen und deren wahrscheinlichster Ursache ist in Anhang 8.6. zu sehen. Die Algorithmen zeigen sich robust gegenüber den ständig wechselnden Höhen, so lange sich der Quadrocopter in dem in Abschnitt 5.1. ermittelten optimalen Bereich von etwa 60cm bis 120cm befindet. Auch die zum Teil deutlich über dem Durchschnittswert von 0,5m/s liegende Vorbeiflug-Geschwindigkeit reduziert meist nur die Anzahl der erfolgreichen Detektionen, führt aber nur sehr selten zu einem „Übersehen“ eines Zielobjekts. Große Unterschiede zwischen den beiden verwendeten Implementierungen sind dabei nicht festzustellen. Lediglich bei Höhen deutlich oberhalb des optimalen Bereichs besitzt die OpenCV-Implementierung leichte Vorteile. Dagegen ermöglicht die eigene Implementierung gerade beim

zeitgleichen Überfliegen zweier Zielobjekte eine etwas stabilere Detektion.
Später im Test verschlechterten sich die Lichtverhältnisse, da sich die nur von einer Seite einfallende Sonneneinstrahlung deutlich erhöhte. Die Auswirkungen sind in Abb. 57 zu sehen.
Die stark aufgehellte rechte Seite der Bälle führt zu weniger Zielpixeln im Binärbild Dies hat wiederum eine geringere Anzahl von Kantenpixeln zur Folge, die auf dem Umriss des wahren Kreises liegen. Dadurch wird der eigentlich zu findende Kreis von zu wenig Kantenpixeln unterstützt, und ein kleiner Kreis wird im linken Bereich des Balls detektiert. Dieses Ergebnis ist zwar nicht optimal, jedoch deutlich besser als wenn gar keine Detektion stattgefunden hätte.

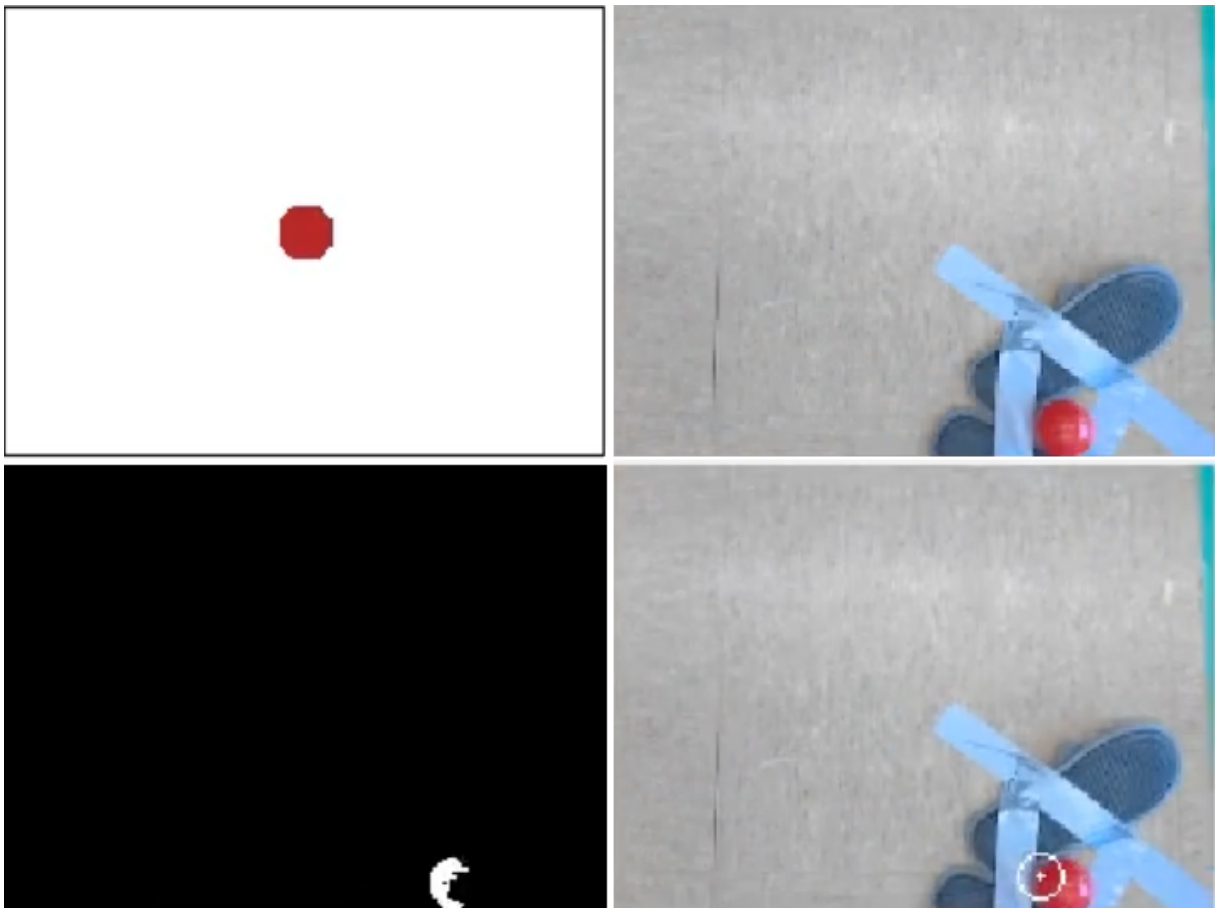

Abbildung 57: Auftretende Probleme bei einseitiger Sonneneinstrahlung:
oben: Scanergebnis, Suchbild
unten: Binärbild, detektierter Kreis

6. Diskussion und Ausblick

Im Folgenden werden zum einen noch einmal die erzielten Ergebnisse dieser Arbeit zusammengefasst, zum anderen wird ein Ausblick über mögliche Verbesserungen des Systems und deren Auswirkungen gegeben.

6.1. Diskussion der Ergebnisse

Insbesondere die Versuche unter Laborbedingungen haben gezeigt, dass das bestehende System die Anforderungen an eine Objekterkennung erfüllt. Kugelförmige Objekte verschiedener Farbe und Größe können gescannt und zuverlässig wiedererkannt werden. Dabei ist auch eine Variation der Höhe innerhalb bestimmter Grenzen möglich. Ebenso ist das System schnell genug, um sich zügig durch den Aufnahmebereich der Kamera bewegende Objekte sicher zu erkennen. Dabei ist eine Detektion bei Geschwindigkeiten von bis zu 2,5m/s noch möglich, ohne dass das System versagt.
Diese Arbeit hat gezeigt, dass variierende Helligkeitsverhältnisse zu Problemen führen können. Jedoch wurden die Parameter so optimiert, dass sich das System relativ robust gegenüber leichten Helligkeitsveränderungen verhält. Einen wesentlich stärkeren und v.a. unberechenbareren Einfluss auf die Detektion eines Objekts hat die direkte Lichteinstrahlung auf dieses. Wie in Abb. 57 zu sehen, können dabei große Teile des zu detektierenden Objekts im Binärbild verloren gehen, was eine korrekte Detektion erheblich erschwert. Eine weiteres Problem stellt, speziell in Verbindung mit schwierigen Lichtverhältnissen, das Unterscheiden von Formen dar. So ist es beispielsweise bisher nicht immer möglich, mit allgemein gültigen Parametern eine sichere Nicht-Detektion eines Quadrates oder Rechteckes in der Zielfarbe und ähnlicher Größe zu gewährleisten.
Die Integration in den bestehenden Quadrocopter-Aufbau wurde erfolgreich durchgeführt. Die Verwendung einer Remote-Desktop-Verbindung per W-Lan ermöglicht zum einen weiterhin eine nahezu uneingeschränkte Steuerung des Quadrocopters. Zum anderen ist eine ständige Überwachung der Vorgänge auf dem integrierten PC möglich. Auch die mit Hilfe dieses Aufbaus erzielten Ergebnisse erfüllen die Erwartungen an das System, auch wenn die eingezeichneten Kreise nicht immer optimal an das korrekt detektierte Objekt angepasst sind.

6.2. Ausblick

Es sind sowohl im Bereich des Konzepts, als auch der Implementierung Verbesserungen des Systems vorstellbar. Einige davon werden im Folgenden kurz vorgestellt:
Zunächst ist es denkbar, die jeweils aktuelle Höhe des Quadrocopters in die Kreis-Detektion miteinzubeziehen. Der im Kamerabild erscheinende Radius des zu suchenden Objekts hängt stark vom Abstand der Kamera zum Objekt, also von der Flughöhe des Quadrocopters, ab. Bisher wurde versucht, einen bestimmten Höhenbereich, z.B. 70cm bis 130cm, abzudecken, indem dem zu suchenden Radius eine gewisse Toleranz, z.B. +-4, zugesprochen wurde. Dies führt zum einen zu einem

größeren Rechenaufwand und erhöhten Speicherbedarf. Zum anderen werden regelmäßig zu kleine Radien in gefundene Objekte eingepasst. Eine mögliche Lösung wäre es, für jede Höhe nur einen einzigen Radius oder einen sehr kleinen Radiusbereich vorzusehen und diesen je nach aktueller Flughöhe anzupassen.
Ein weiteres, sehr Erfolg versprechendes Konzept stellt eine Aufteilung der Suche in zwei Phasen dar. In der ersten Phase sind dabei die Kriterien an das zu findende Objekt sehr gering. Es könnte beispielsweise zunächst nur sehr grob nach Farbe und ungefährer Größe des Zielobjekts gesucht werden. Dabei steht lediglich im Vordergrund, dass kein mögliches Zielobjekt übersehen wird. Hat das System einen Kandidaten ausgemacht, beginnt die zweite Phase mit der genaueren Untersuchung des Objekts. Dazu stoppt der Quadrocopter und reduziert gegebenenfalls seine Flughöhe. Da durch das Abstoppen und das Verweilen über einer bestimmten Position nun die Framerate, und damit auch die Rechenzeit, eine eher untergeordnete Rolle spielt, können anschließend Bilder mit höchster Auflösung aufgenommen und eingehend analysiert werden.
Des Weiteren ist ein erneutes Überdenken der Grundidee des Konzepts notwendig. Durch die Reduktion des Zielbildes auf ein Binärbild gehen Informationen verloren, die im weiteren Verlauf noch von Nutzen hätten sein können. Dies kann zu Schwierigkeiten in der nachfolgenden Kantendetektion führen, da alle Kanten gleich stark gewichtet werden und z.B. ein Übergang von Rot zu Rosa ebenso schwer wiegt wie einer von Rot zu Blau, solange sich Rosa außerhalb des Zielfarbenbereichs befindet. Durch das Verwenden einer fließenden, im Gegensatz zur bisher digital durchgeführten, Farbfilterung wäre eine Kantendetektion denkbar, die die Stärke der Kanten anhand ihrer Differenz zur Zielfarbe gewichtet.
Einen weiteren Ansatzpunkt für Verbesserungen stellen die verwendeten Kameras und deren Funktionen dar. So könnte ein besseres Verständnis für die in modernen Webcams zum Einsatz kommenden Funktionen, wie die Autofokussierung oder die Anpassung der Helligkeit, zur weiteren Steigerung der Leistungsfähigkeit des Systems beitragen.

7. Quellenverzeichnis

[Burger 2009a] Burger, W., Burge, M.: Principles of Digital Image Processing – Fundamental Techniques. Springer. 2009.

[Burger 2009b] Burger, W., Burge, M.: Principles of Digital Image Processing – Core Algorithms. Springer. 2009.

[Davies 2012] Davies, E.R.: Computer & Machine Vision – Theory, Algorithms, Practicalities. Academic Press. 2012.

[Ebner 2007] Ebner, M.: Color Constancy. Wiley. 2007.

[Efford 2000] Efford, N.: Digital Image Processing – A Practical Introduction Using Java. Pearson Education Limited. 2000.

[Forsyth 2012] Forsyth, D.: Computer Vision – A Modern Approach. Pearson Education. 2012².

[Gageik 2012] Gageik, N., Montenegro, S., Müller, T.: Obstacle Detection And Collision Avoidance Using Ultrasonic Distance Sensors For An Autonomous Quadrocopter. University of Würzburg, Aerospace Information Technology. UAVveek. 2012.

[Hermes 2005] Hermes, T.: Digitale Bildverarbeitung. Carl Hanser Verlag. 2005.

[Johnson 2006] Johnson, S.: Stephen Johnson on Digital Photography. O´Reilly. 2006.

[Linß 2010] Linß, G.: Praktische Ausbildung und Training Qualitätsmanagement – Objekterkennung mit Hough-Transformation. Technische Universität Ilmenau. 2010.

[Lordemann 2003] Lordemann, C., Lambers, M.: Objekterkennung in Bilddaten. Universität Münster. 2003.

[OpenCV 2008] Bradski, G., Kaehler, A.: Learning OpenCV. O´Reilly. 2008.

[Rhody 2005] Rhody, H.: Lecture 10: Hough Circle Transform. Rochester Institute of Technology. 2005.

[AlphaCard 2012] AlphaCard: *Homepage*. 2012. - URL http://www.alphacard.com/media/catalog/product/cache/3/image/400x225/9df78eab33525d08d6e5fb8d27136e95/l/o/loh-960-000415_logitech-camera_main.jpg

[DRadioWissen 2010] DRadio Wissen: Fliegende Spürnasen. *Homepage*. 2010. - URL http://wissen.dradio.de/forschung-fliegende-spuernasen.35.de.html?dram:article_id=1435

[MathWorks 2012] The MathWorks, Inc.: *Homepage*. 2012. - URL http://www.mathworks.se/help/images/ref/imfc_alonggradient.png

[Microdrones 2013] Microdrones GmbH: *Homepage*. 2013. - URL http://www.microdrones.com/index-de.php

[MyBitcom 2013] BitcomTech: *Homepage*. 2013. - URL http://mybitcom.de/images/logitech_c270_hdwebcam.jpg

[SpiegelOnline 2010] Spiegel Online Wissenschaft: Katastrophen-Kommunikation: Mini-Helis sollen kaputte Handy-Netze flicken. *Homepage*. 2010. - URL http://www.spiegel.de/wissenschaft/technik/katastrophen-kommunikation-mini-helis-sollen-kaputte-handy-netze-flicken-a-708173.html#ref=rss

[Strohmeier 2012] Strohmeier M.: Implementierung und Evaluierung einer Positionsregelung unter Verwendung des optischen Flusses. Universität Würzburg. 2012. - URL http://www8.informatik.uni-wuerzburg.de/fileadmin/10030800/user_upload/quadcopter/Videos/Optischer_Fluss_Michael_Strohmeier_BA.pdf

[Universität Bochum 2007] Universität Bochum: Institut für Neuroinformatik: Research Group Real-time Computer Vision: *Homepage*. 2007. - URL http://wwwold.ini.rub.de/thbio/group/vision/images/Detection.png

[Universität München 2013] Universität München: Lehrstuhl für Geographie und Geographische Fernerkundung: Internet-Vorlesung Geographie und Fernerkundung: *Homepage*. 2013. - URL http://www.geographie.uni-muenchen.de/internetvorlesung/arbeitsmethoden/Graphik/AB1-5.JPG

[Wikipedia] Wikipedia: *Homepage*. Wikimedia Foundation. - URLs
http://de.wikipedia.org/wiki/Datei:Kreisgleichung.png
http://de.wikipedia.org/wiki/Falsch_negativ#Wahrheitsmatrix:_Richtige_und_falsche_Klassifikationen
http://de.wikipedia.org/wiki/Hough-Transformation
http://de.wikipedia.org/wiki/Objekterkennung

[ZeitOnline 2011] Zeit Online Wissen: Fliegende Augen: *Homepage*. 2011. - URL http://www.zeit.de/2011/35/T-Flugroboter/seite-2

8. Anhang

8.1. PC und Laptop

Um die Laufzeiten der Algorithmen beurteilen zu können, ist in Tabelle 5 eine kurze Übersicht über die Hardware des auf den Quadrocopter verwendeten PCs und des Laptops gegeben.

Gerät	Prozessor	Arbeitsspeicher	Betriebssystem	Systemtyp
On-Board PC	AMD G-T56N 2 x 1.65 GHz	4 GB	Windows 7	32 bit
Laptop	AMD A6 2 x 2.7 GHz	4GB	Windows 7	64 bit

Tabelle 5: Übersicht über die technischen Daten des verwendeten PCs und Laptops

Dabei stammen die Laufzeit-Daten, wenn nicht explizit anders erwähnt, von der Ausführung der Software auf dem Laptop.

8.2. Verwendete Kameras

In dieser Arbeit werden drei Kameras zur Objekterkennung eingesetzt.
Zum einen zwei Webcams mit USB-Anschluss:

- Logitech C200 Webcam
- Logitech C270 Webcam

Zum anderen der OV7670 CMOS Sensor von OmniVision. Abb. 58 zeigt die drei verwendeten Kameras.

Abbildung 58: Verwendete Kameras:
links: Logitech C200 [Alphacard]
Mitte: Logitech C270 [MyBitcom]
rechts: OV7670 [Strohmeier 2012]

In Tabelle 6 ist eine Übersicht über die wichtigsten Parameter der verwendeten Kameras zu sehen.

Kamera	Auflösung	Scharfstellung
Logitech C200 Webcam	640 x 480	manuell
Logitech C270 Webcam	1280 x 720	automatisch
OV7670	128 x 128	manuell

Tabelle 6: Übersicht über einige technische Daten der verwendeten Kameras

8.3. Parameter der *cvHoughCircles()*-Funktion

CvSeq cvHoughCircles(Mat* image, void* circle_storage, int method, double dp, double min_dist, double param1, double param2, int min_radius, int max_radius)*

- *Mat* image*: Bilddaten des Suchbilds im *Mat*-Format. Das Eingangsbild muss ein Graustufenbild sein. Farb- oder Binärbilder werden nicht akzeptiert.
- *void* circle_storage*: Datenstruktur, in der die gefundenen Kreise gespeichert werden. Dabei bietet sich ein *Vector* an, der für jeden Kreis den x- und y-Wert des Mittelpunktes und den Kreisradius speichert.
- *int method*: Gibt die bei der Kreiserkennung verwendete Methode an. CV_HOUGH_GRADIENT ist im Moment die einzige in OpenCV implementierte Methode.
- *double dp*: Der Parameter *dp* entspricht der Auflösung des Akkumulator-Arrays. Dabei entspricht der Wert 1 der Auflösung des Such-Bildes. Ein Wert größer 1 reduziert die Auflösung um diesen Faktor.
- *double min_dist*: Der Mindestabstand zweier Kreismittelpunkte. Wird dieser unterschritten, werden die beiden Kreise nicht als zwei unterschiedliche Kreise detektiert.
- *double param1*: Definiert die Grenzwerte für die Canny-Kantendetektion.
- *double param2*: Gibt an, von wie vielen Kantenpixeln ein Mittelpunkt unterstützt werden muss, um als endgültige Lösung zu gelten.
- *int min_radius*: Minimaler Radius der zu suchenden Kreise.
- *int max_radius*: Maximaler Radius der zu suchenden Kreise.

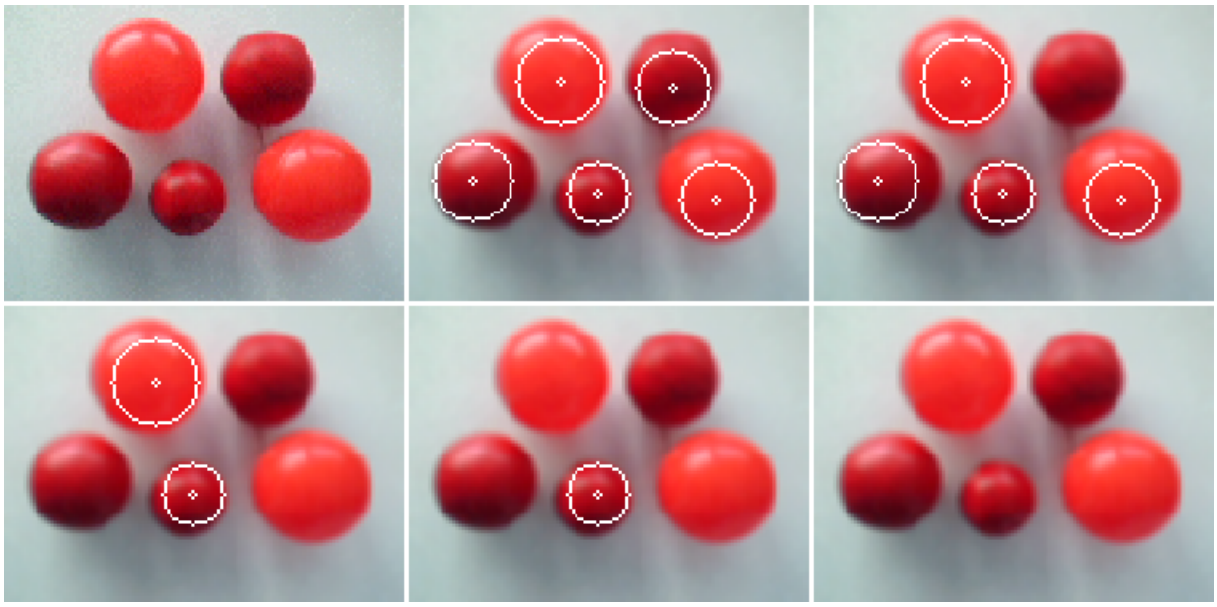

Abbildung 59: Auswirkungen unterschiedlicher param2-Werten bei der Suche:
oben: Suchbild, Ergebnisbild mit Wert 12, Wert 14
unten: Ergebnisbild mit Wert 16, Wert 18, Wert 20

Der Eingabeparameter *param2* nimmt großen Einfluss auf die Kreisdetektion, da schon minimale Änderungen zum Erkennen bzw. Nicht-Erkennen eines Objektes führen können. Werte zwischen 10 und 20 erscheinen in den meisten Situationen als angebracht. Abb. 59 soll die Empfindlichkeit der Kreissuche in Bezug auf *param2* verdeutlichen. Verwendet werden die Scanergebnisse und das Suchbild aus Abb. 36 [OpenCV 2008].

8.4. Laufzeit der verwendeten Implementierungen

8.4.1. Einfluss der Auflösung auf die Laufzeit

Verwendete Implementierung	Anzahl Bälle	gewählte Auflösung	Laufzeiten der Einzelmessungen (ms) 1	2	3	4	5	6	7	8	9	10	Mittelwert (ms)
eigene Impl.	1	128x96	21	22	24	23	20	20	21	23	21	23	21,8
		192x144	33	38	32	28	41	30	37	41	40	32	35,2
		256x192	58	68	57	56	54	55	58	55	46	54	56,1
eigene Impl.	5	128x96	34	26	29	32	23	31	33	30	31	32	30,1
		192x144	69	60	63	69	60	71	58	54	76	54	63,4
		256x192	105	100	95	90	97	92	101	96	94	87	95,7
OpenCV	1	128x96	9	7	9	9	8	7	8	8	7	7	7,9
		192x144	12	12	13	12	13	11	13	11	11	12	12
		256x192	19	17	21	19	17	18	18	20	17	18	18,4
		320x240	30	26	24	28	30	28	28	29	28	22	27,3
		720x540	73	72	79	82	81	77	81	78	81	79	78,3
		1280x720	186	192	196	179	191	195	200	198	195	192	192,4
OpenCV	5	128x96	9	8	8	9	8	9	8	9	10	9	8,7
		192x144	12	12	13	11	13	11	14	13	13	13	12,5
		256x192	18	19	19	23	18	20	17	18	18	18	18,8
		320x240	29	26	25	30	28	26	28	26	28	28	27,4
		720x540	73	81	81	84	74	88	74	87	69	81	79,2
		1280x720	198	197	204	179	194	199	203	195	191	193	195,3

Tabelle 7: Übersicht über die Laufzeiten beider Implementierungen bei verschiedenen Auflösungen

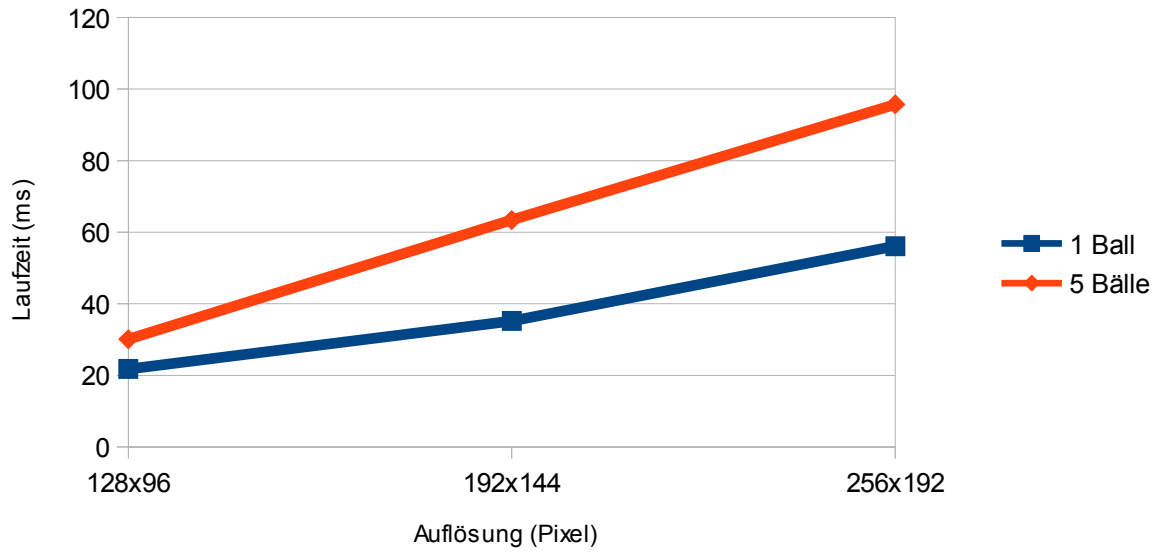

Abbildung 60: Einfluss der Auflösung auf die Laufzeit unter Verwendung der eigenen Implementierung

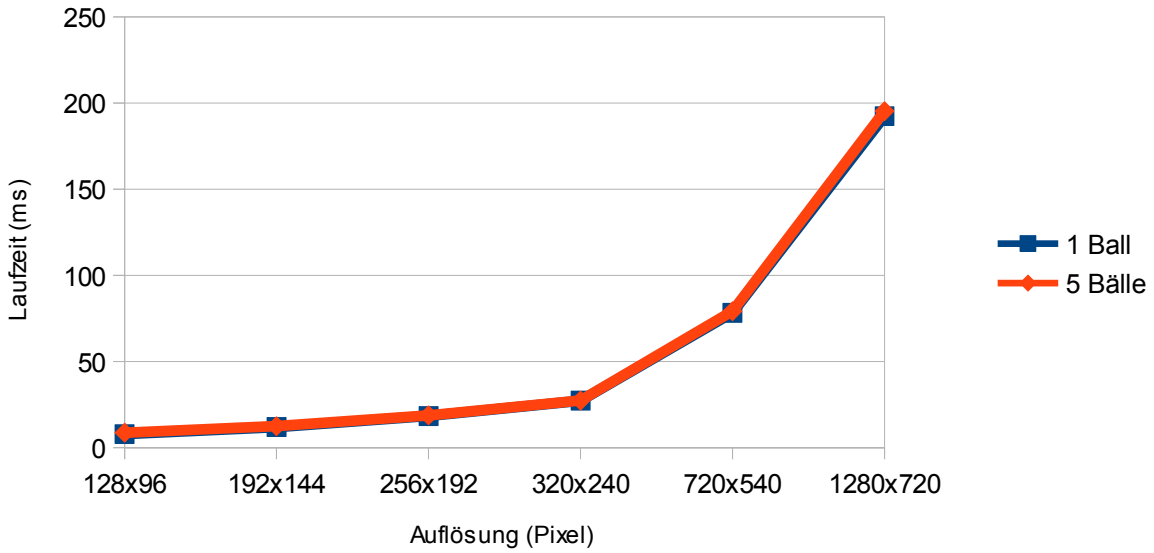

Abbildung 61: Einfluss der Auflösung auf die Laufzeit unter Verwendung der OpenCV-Implementierung

8.4.2. Einfluss der Anzahl der Radien auf die Laufzeit

Verwendete Implementierung	Anzahl Bälle	Anzahl Radien	Laufzeiten der Einzelmessungen (ms)										Mittelwert (ms)
			1	2	3	4	5	6	7	8	9	10	
eigene Impl.	1	1	47	32	31	31	47	16	31	31	32	31	32,9
		3	47	46	47	46	47	46	63	47	63	47	49,9
		5	63	63	62	47	62	63	46	62	47	62	57,7
		7	64	74	73	70	78	73	59	65	75	66	69,7
		9	78	62	78	94	63	93	94	78	78	78	79,6
eigene Impl.	2	1	63	47	46	47	46	47	47	31	47	47	46,8
		3	63	62	78	63	62	62	63	45	62	63	62,3
		5	78	78	78	94	63	79	93	78	94	78	81,3
		7	91	98	99	106	98	100	113	106	100	107	101,8
		9	125	140	125	140	140	125	125	140	141	125	132,6
OpenCV	1	11	16	16	15	16	16	31	16	16	16	16	17,4
		21	16	16	31	15	16	16	16	15	16	31	18,8
		31	16	16	31	15	16	31	16	16	31	16	20,4
OpenCV	2	11	16	15	16	16	16	15	16	32	16	16	17,4
		21	16	32	16	31	16	16	15	16	16	15	18,9
		31	16	32	16	16	31	16	15	31	15	16	20,4

Tabelle 8: Übersicht über die Laufzeiten beider Implementierungen bei verschiedenen Radiusbereichen

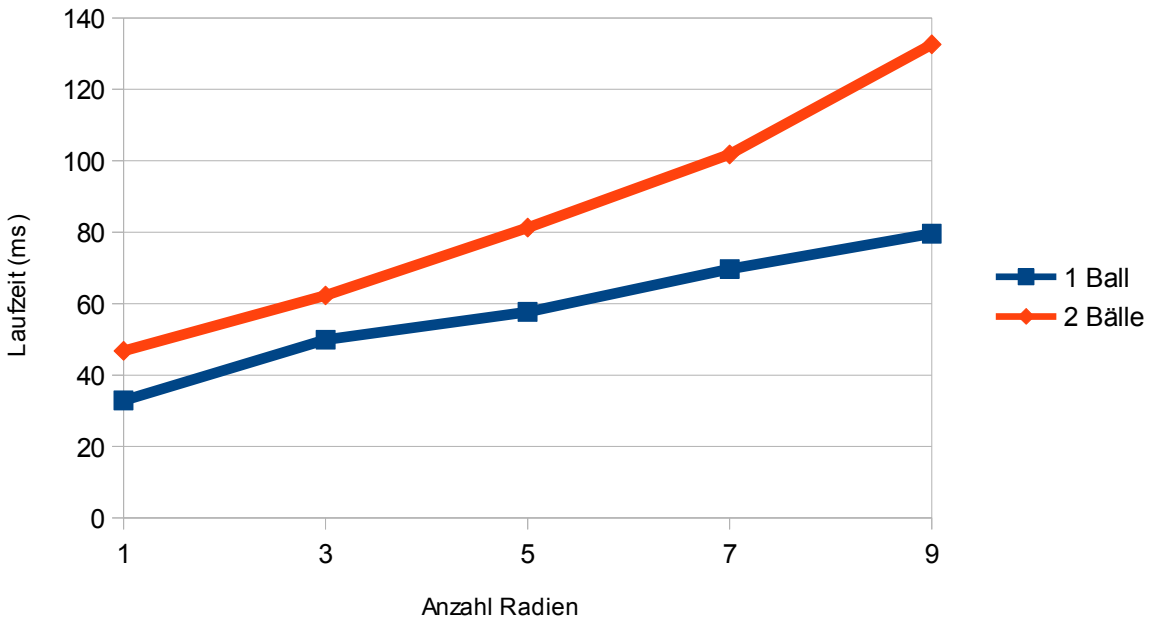

Abbildung 62: Einfluss der Anzahl der Radien auf die Laufzeit bei Verwendung der eigenen Implementierung

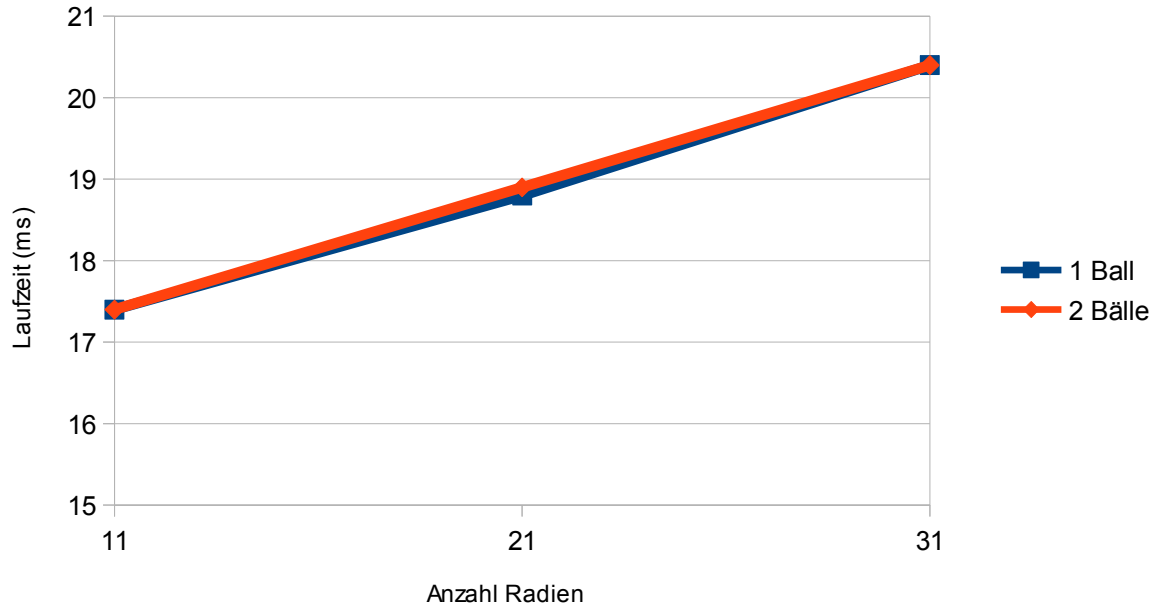

Abbildung 63: Einfluss der Anzahl der Radien auf die Laufzeit bei Verwendung der OpenCV-Implementierung

8.5. Simulation des Vorbeiflugs eines Quadrocopters

8.5.1. Geschwindigkeit von ca. 0,5m/s unter Verwendung der C270

Messung #	Erstdetektion bei Pixel	Entspricht %
1	246	3,91
2	240	6,25
3	244	4,69
4	244	4,69
5	239	6,64
6	245	4,3
7	245	4,3
8	243	5,08
9	246	3,91
10	240	6,25
Durchschnitt:	**243,2**	**5**

Tabelle 9: Ort der Erstdetektion bei einer Geschwindigkeit von 0,5m/s unter Verwendung der eigenen Implementierung

Messung #	1	2	3	4	5	**Mittelwert**
Anzahl Detektionen	27	26	26	28	25	**26,4**

Tabelle 10: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der eigenen Implementierung.

Messung #	Erstdetektion bei Pixel	Entspricht %
1	244	4,69
2	247	3,52
3	248	3,13
4	247	3,52
5	249	2,73
6	254	0,78
7	245	4,3
8	249	2,73
9	247	3,52
10	249	2,73
Durchschnitt:	**247,9**	**3,16**

Tabelle 11: Ort der Erstdetektion bei einer Geschwindigkeit von 0,5m/s unter Verwendung der OpenCV-Implementierung

Messung #	1	2	3	4	5	Mittelwert
Anzahl Detektionen	39	39	37	41	36	**38,4**

Tabelle 12: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der OpenCV-Implementierung

8.5.2. Geschwindigkeit von 1m/s unter Verwendung der C270

Messung #	Erstdetektion bei Pixel	Entspricht %
1	235	8,2
2	244	4,69
3	237	7,42
4	246	3,91
5	236	7,81
6	232	9,38
7	243	5,08
8	239	6,64
9	241	5,86
10	238	7,03
Durchschnitt:	**239,1**	**6,6**

Tabelle 13: Ort der Erstdetektion bei einer Geschwindigkeit von 1m/s unter Verwendung der eigenen Implementierung

Messung #	1	2	3	4	5	Mittelwert
Anzahl Detektionen	11	12	10	10	12	**11**

Tabelle 14: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der eigenen Implementierung

Messung #	Erstdetektion bei Pixel	Entspricht %
1	245	4,3
2	253	1,17
3	247	3,52
4	237	7,42
5	242	5,47
6	248	3,13
7	235	8,2
8	242	5,47
9	247	3,52
10	240	6,25
Durchschnitt:	**243,6**	**4,84**

Tabelle 15: Ort der Erstdetektion bei einer Geschwindigkeit von 1m/s unter Verwendung der OpenCV-Implementierung

Messung #	1	2	3	4	5	Mittelwert
Anzahl Detektionen	17	21	15	18	22	**18,6**

Tabelle 16: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der OpenCV-Implementierung

8.5.3. Vorbeiflug mit einer Geschwindigkeit von 1m/s unter Verwendung der C200

Messung #	Erstdetektion bei Pixel	Entspricht %
1	243	5,08
2	232	9,38
3	247	3,52
4	245	4,3
5	233	8,98
6	244	4,69
7	222	13,28
8	244	4,69
9	246	3,91
10	245	4,3
Durchschnitt:	**240,1**	**6,21**

Tabelle 17: Ort der Erstdetektion bei einer Geschwindigkeit von 1m/s unter Verwendung der eigenen Implementierung

Messung #	1	2	3	4	5	Mittelwert
Anzahl Detektionen	15	12	14	3	13	**13,4**

Tabelle 18: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der eigenen Implementierung

Messung #	Erstdetektion bei Pixel	Entspricht %
1	248	3,13
2	238	7,03
3	244	4,69
4	245	4,3
5	245	4,3
6	245	4,3
7	249	2,73
8	240	6,25
9	248	3,13
10	234	8,59
Durchschnitt:	**243,6**	**4,84**

Tabelle 19: Ort der Erstdetektion bei einer Geschwindigkeit von 1m/s unter Verwendung der OpenCV-Implementierung

Messung #	1	2	3	4	5	Mittelwert
Anzahl Detektionen	19	13	12	12	14	**14**

Tabelle 20: Anzahl der Detektionen bei einem Durchlauf des Aufnahmebereichs der Kamera unter Verwendung der OpenCV-Implementierung

8.6. Echtzeitflug im Quadrocopter

Implementierung	Anzahl Bälle	davon erkannt	entspricht Prozent	Fehler negativ	Fehler positiv	Ursache Licht	Ursache Höhe	Ursache Geschw.
eigene Impl.	34	28	82,35	6	0	3	2	1
eigene Impl.	18	15	83,33	3	2	3	0	0
OpenCV	46	41	89,13	5	0	2	2	1
OpenCV	44	35	79,55	9	0	8	1	0