HUMBOLDT-UNIVERSITÄT ZU BERLIN

Institut für Informatik
Arbeitsgruppe für künstliche Intelligenz

Kantenbasierte Wiedererkennung einfarbiger Objektflächen

Eine Ausarbeitung von André Stephan
Betreuer: Daniel Göhring

26. März 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Zusammenfassung

In dieser Arbeit wird die Objekterkennung und -wiedererkennung thematisiert. Es werden dafür allgemeine Grundlagen der *Computer-Vision* behandelt und ihre Nutzbarkeit abgewogen. Als Ergebnis wird ein modulares System erzeugt, das es ermöglicht, beliebig geformte, monotone Flächen zu erkennen und später wiederzuerkennen. Dabei handelt es sich um eine schnelle, farb- und formunabhängige Objekterkennung, gestützt auf den Ergebnissen des SUSAN-Kantendetektors. So könnten die Anwendungsbereiche beispielsweise in der Selbstlokalisierung oder Kartenerstellung in einer dem Roboter unbekannten Umwelt liegen.

1 Einleitung

Die Fähigkeit eines autonomen Agenten seine Umwelt wahrnehmen zu können ist die Grundvoraussetzung für sein Handeln. Speziell die visuelle Wahrnehmung eines dreidimensional ausgeprägten Roboters einer ihm unbekannten Umwelt, stellt eine besondere Herausforderung dar.
Selbst in einer sehr stark eingeschränkten und definierten Umwelt wie dem Roboterfußball, kann diese ungewollt von der Definition abweichen und dieses in einem Nichterkennen von Objekten resultieren. So ändern sich die Farbeigenschaften von Objekten mitunter extrem, wenn zum Beispiel Intensitätsschwankungen des Tageslichtes, nicht homogene Lichtverhältnisse oder Verschattungen durch die Spieler existieren. Andererseits ist eine dreidimensionale Kartografierung eines allgemeinen Innenraumes auf das Erkennen neuer Objekte angewiesen, die sich darüber hinaus nicht immer durch feste Objektformen (wie Kreise, Dreiecke oder Rechtecke) genau beschreiben lassen. Dazu kommt, dass je nach Lage und Neigung der Roboterkamera relativ zu den erkannten Objekten, können diese im Kamerabild translatiert, in ihrer Größe verändert, verzerrt oder rotiert sein. Um jedoch erfolgreich eine Karte erstellen zu können, muss der Roboter diese Objekte trotzdem wiedererkennen können. Da ein Roboter nur über begrenzte Ressourcen verfügt, sollte jeder Algorithmus natürlich so sparsam wie möglich mit den Ressourcen umgehen.
Diese Arbeit beschäftigt sich mit einer syntaktischen [7] Erkennung und Wiedererkennung monoton gefärbter Objektflächen. Es handelt sich dabei um eine kameragestütze Wiedererkennung, wobei ein farb- und formunabhängiger Ansatz verfolgt wird, der zuvor gesehene Objekte wiedererkennen soll, trotz Translation, Rotation, Größenänderung und Verzerrung. Ein weiteres Ziel ist es einen vollständigen Abarbeitungsschritt in deutlich unter einer Sekunde zu erzielen.

1.1 Verfahrensweise

Es werden zunächst einige der klassischen Ansätze des computergestützten Sehens aufgeführt und beschrieben. Diese werden auf ihre Tauglichkeit bezüglich der Zielsetzung überprüft und abgewogen. Mit diesem Wissen kann dann modular jede Teilfunktionalität genauer konzipiert werden, aus denen der komplexe Algorithmus aufgebaut sein wird. Nach einer Übersicht des Algorithmus werden dann die erzielten Ergebnisse besprochen und Aussichten für mögliche Erweiterungen dieser Arbeit gegeben. Im Anhang wird auf die verwendete

Hardware und Software der Arbeitsumgebung eingegangen und die verwendete Kamerakonfigurationen aufgelistet, mit der das Arbeitsziel erreicht wurde.

2 Grundlagen

Im Folgenden wird ein kurzer Überblick über bekannte Ansätze innerhalb der *object recognition* gegeben. Dieser Prozess ist im Allgemeinen vielschichtig und kann in die folgenden Schwerpunkte aufgegliedert werden: Farbraumreduzierung, Objekterkennung und Objektidentifikation. Im Anschluss wird eine Auswahl von Modulen vorgestellt, die für die weitere Arbeit relevant sind.

2.1 Farbraumreduzierung

Alle klassischen Ansätze, speziell die Verfahren zur Kantenfindung wie der Prewit-, Laplace- oder Sobel-Operator, haben die gemeinsame Eigenschaft nur auf eindimensionalen Farbräumen (üblicherweise mit 256 Werten) zu arbeiten. Dies ist typischerweise ein Grauwertbild, wie es im Y-Kanal des YUV-Farbraumes verwendet wird. Genauso könnte man aber auch die einzelnen Kanäle aus einem dreidimensionalen Farbraum als Ausgangsbasis nehmen, diese einzeln verarbeiten und die gewonnenen Informationen wieder zusammen führen. Der Y-Kanal hat aber eine speziell an das menschliche Helligkeits- und Farbsehen angepasste Farbverteilung der Rot-, Grün- und Blaufarbwerte und ist deswegen den anderen vorzuziehen.

2.2 Objekterkennung

Bei der Objekterkennung handelt es sich um das Auffinden von farblich zusammengehörenden Teilen (die Objekte) im zweidimensionalen Bild. Es haben sich dafür zwei große Ansätze entwickelt. Einerseits die Kantenerkennung (*edge detection*) und andererseits die Bereichssegmentierung (*region segmentation*). Bei der Kantenerkennung werden die Objektgrenzen (Kantenpunkte) detektiert und bei der Bereichssegmentierung werden ganze Bereiche innerhalb eines Objektes festgestellt, in denen die Farbpixel über eine gewisse Ähnlichkeit verfügen.

2.2.1 Kantenerkennung

Eine klassische Kantenerkennung durchläuft drei Ebenen: Rauschfilterung, Gradientenbestimmung und Vervollständigung der Kantenzüge.

Vor einer Kantenextraktion werden typischer Weise **Rauschfilter** auf das Bild

angewandt, die mögliches Rauschen im Bild[1] unterdrücken. Bei der Rauschunterdrückung durch einen Mittelwertfilter wird der Farbwert jedes Pixels an die Pixelfarbwerte seiner Umgebung angepasst, wobei alle Pixelfarbwerte in der Umgebung gleich gewichtet werden. Dafür wird über jeden Pixel des Bildes eine so genannte Maske geschoben. Die Maske ist eine Art Matrix, deren Einträge mit den Farbwerten der Bildpixel verknüpft werden. Dabei wird jeweils der Mittelpunkt der Maske auf den aktuellen Pixel im Bild gelegt und die benachbarten Pixel werden mit den entsprechenden Einträgen der Matrix verknüpft. Der Mittelwertfilter bezieht alle Farbwerte innerhalb des Bereichs der Maske gleichmäßig mit ein (siehe Abbildung 1 a). Somit glättet der Mittelwertfilter extreme Frequenzen linear. Ein weiterer Filter ist der Gaußfilter. Die zweidimensionale Gaußfunktion ist wie folgt definiert:

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\left(-\frac{(x-x_0)^2}{2\sigma_x^2} - \frac{(y-y_0)^2}{2\sigma_y^2}\right)} \tag{1}$$

Im Gegensatz zum Mittelwertfilter werden hier nicht alle Pixelfarbwerte gleich gewertet, sondern jetzt werden die Pixelfarbwerte anti proportional zu ihrer Entfernung zum zentralen Pixel gewichtet (siehe Abbildung 1 b und c). Somit glättet der Gaußfilter extreme Frequenzen nicht linear. Eine weitere Filterart bietet der Medianfilter. Im Gegensatz zu einem Gauß- oder Mittelwertfilter eliminiert der Medianfilter extreme Frequenzen, in dem er jeden Pixelfarbwert durch den Median der Pixelfarbwerte in seiner Umgebung ersetzt.

Kommen wir nun zur **Gradientenbestimmung**. Kanten sind Intensitätsänderungen (Gradient) des Farbwerts (eindimensionaler Wert) im Bild. Manche Änderungen des Farbwerts sind sprunghaft und andere kontinuierlich. Es gibt letztendlich 4 verschiedene Arten von Kanten (siehe Abbildung 2) in einem Bild: *step, ramp, spike* und *roof* Kanten[4]. Intensitätsänderungen können durch Differentialquotienten der Farbwerte bestimmt werden. Für eine stetige Funktion f ist die Ableitung bestimmt durch:

$$f' = \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} \tag{2}$$

Da wir aber diskrete Farbwerte besitzen, wird aus Δx minimal 1 und es gilt dann:

$$f' = \frac{\Delta y}{\Delta x} = f(x) - f(x-1) \tag{3}$$

[1]Bildrauschen wird unter anderem durch Schwankungen im Sensorsignal der Kamera verursacht, was eine Verfälschung des Bildes zur Folge hat. In den für Kameras typischen CCD-Sensoren (*Charge-coupled Device* was in etwa „ladungsgekoppeltes Bauteil bedeutet) wird Rauschen durch zu schwache Beleuchtung des Sensors verursacht.

(a)			(b)			(c)		
1	1	1	0,578	0,953	0,578	0,029	0,213	0,029
1	1	1	0,953	1,571	0,953	0,213	1,571	0,213
1	1	1	0,578	0,953	0,578	0,029	0,213	0,029
0,111	0,111	0,111	0,075	0,123	0,075	0,011	0,084	0,011
0,111	0,111	0,111	0,123	0,204	0,123	0,084	0,619	0,084
0,111	0,111	0,111	0,075	0,123	0,075	0,011	0,084	0,011

Abb. 1: *Verschiedene Rauschfilter mit* 3×3 *Maskengröße im Überblick, oben: die einzelnen Pixelwertigkeiten, unten: entsprechende prozentuale Anteile, (a): Mittelwertfilter, (b): Gaußfilter mit* $\sigma = 1$ *berechnet für x- und y-Werten aus [-1, 0, 1], (c): Gaußfilter mit* $\sigma = 1$ *berechnet für x- und y-Werten aus [-2, 0, 2], Quelle: eigene Darstellung*

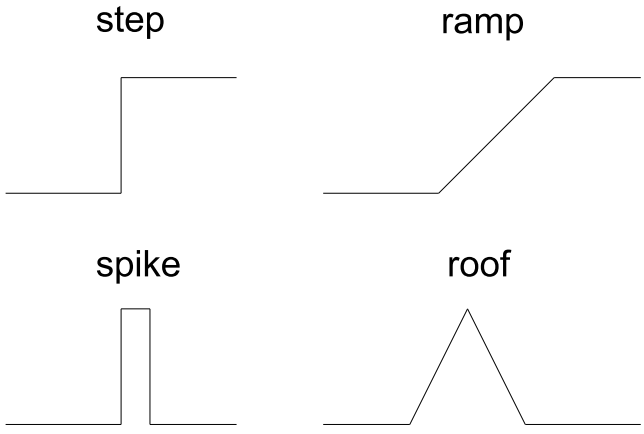


Abb. 2: *Verschiedene Kantenformen aufgelistet, Quelle: [4]*

Die Bestimmung der ersten Ableitung in horizontaler Richtung lässt sich durch verschiedene Masken (siehe Abbildung 3) implementieren.[4] Masken die die

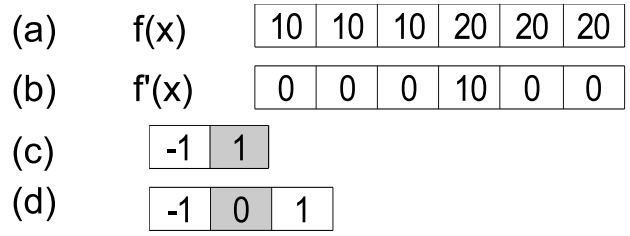


Abb. 3: *Differentialquotient mit Masken, (a): horizontale Farbwerte im Bild, (b): erste Ableitung mit Maske, (c) und (d): Masken, Quelle: [4]*

Differentialquotienten im Bild berechnen werden auch Differenzoperatoren[1] genannt. Weit verbreitete richtungsabhängige Differenzoperatoren für die Bestimmung in x- und y-Richtung sind: der Prewit- und Sobel-Operator (siehe Abbildung 4).[4] Die klassischen Beispiele für richtungsunabhängige Differenz-

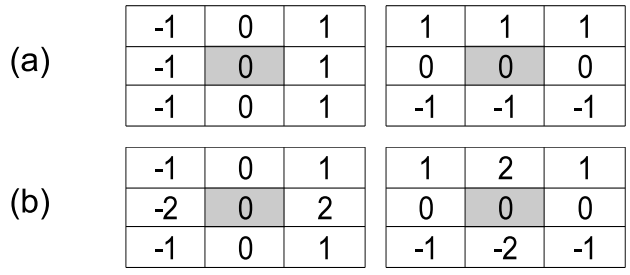


Abb. 4: *Richtungsabhängige Differenzoperatoren, (a): Prewit-Operator, (b): Sobel-Operator, Quelle: [4]*

operatoren sind der Laplace-Operator und eine Kombination mit dem Gaußfilter zur Rauschunterdrückung der Laplace-Gauß-Operator (siehe Abbildung 5). Ausschließlich mit einem richtungsabhängigen Differenzoperator lässt sich zu jedem Pixel im Bild der Gradientvektor bestimmen. Der Gradientvektor ist ein Tupel (f_x, f_y), wobei f_x die Ableitung in x-Richtung und f_y die Ableitung in y-Richtung ist. Jeder Gradient besitzt eine Stärke M (*magnitude*), die dem Differenzquotienten zum Farbwert der Nachbarpixel entspricht, und eine Richtung θ die in die Richtung des Differenzquotienten zeigt. Es gilt dafür:[4]

$$M = \sqrt{f_x^2 + f_y^2} \tag{4}$$

(a)

0	-1	0
-1	4	-1
0	-1	0

(b)

0	1	2	1	0
1	0	-2	0	1
2	-2	-8	-2	2
1	0	-2	0	1
0	1	2	1	0

Abb. 5: *Richtungsunabhängige Differenzoperatoren, (a): Laplace-Operator, (b): Laplace-Gauß-Operator, Quelle: [4]*

$$\theta = \arctan \frac{f_y}{f_x} \tag{5}$$

Bei richtungsunabhängigen Differenzoperatoren wird nur eine Maske für das Bild benötigt, was den Aufwand halbiert. Jedoch kann kein Gradientvektor bestimmt werden. Theoretisch reicht der Gradient an sich aus, um eine Kante zu detektieren. Man sucht einfach nach Gradienten im Bild die einen gewissen vorab definierten Grenzwert überschreiten. Jedoch weisen manche Kanten (roof und ramp) einen kontinuierlichen Verlauf des Gradienten über mehrere Pixel auf. Mit der obigen Methode würden also die *roof* und *ramp* Kanten sehr breit detektiert werden. Zur Behebung dieses Problems kann man die Gradientenrichtung verwenden. Dafür werden alle Gradienten in Richtung θ untersucht. Dabei wird θ auf die Möglichen 8 Nachbarrichtungen diskretisiert. Es wird anschließend auf Maximalität in Bezug auf den Vorgänger und Nachfolger des zu untersuchenden Gradienten getestet, es gilt dafür:[4]

$$M(x,y) = \begin{cases} M(x,y), & M(x,y) > M(x',y') \text{ und } M(x,y) > M(x'',y'') \\ 0, & \text{sonst} \end{cases} \tag{6}$$

Dabei ist $M(x',y')$ der Vorgänger und $M(x'',y'')$ der Nachfolger des Kantenpunktes (x,y) in Richtung θ. Dieses Vorgehen nennt man *non-maximum suppression.*

Um die Kantenpunkte nun zu **Kantenzüge** zu verbinden, hat sich der Ansatz des *surface fitting* entwickelt. Hier wird ausgehend von einem Kantenpunkt nach nächstliegenden Kantenpunkten gesucht, die zum gleichen Kantenzug gehören könnten. Im Canny-Kanten-Detektor wird zur Vervollständigung von Kantenzügen eine Hysteresefunktion eingesetzt, die bei einem hohen Gradienten den Anfangspunkt eines Kantenzuges detektiert und ab dort werden

rekursiv alle Nachbarn betrachtet und diese zum Kantenzug hinzufügt, wenn ihr Gradient größer gleich einem niederen Gradientengrenzwert ist.[4]

Einen nicht so klassischen Ansatz verfolgt der **SUSAN-Kantendetektor**[5]. Hier wird eine Maske über das Bild geschoben und die Anzahl der Pixel bestimmt, die eine ähnliche Farbe wie die des zentralen Pixels (Nucleus) besitzen. Dieser Bereich der Maske wird USAN (*Univalue Segment Assimilating Nucleus*) genannt (siehe Abbildung 6). Je kleiner dieser Bereich ist, umso eher handelt es sich bei dem Nucleus um einen Kantenpunkt, deswegen auch (*Smallset Univalue Segment Assimilating Nucleus*).
Der SUSAN-Algorithmus verwendet im Gegensatz zu den klassischen Ansätzen

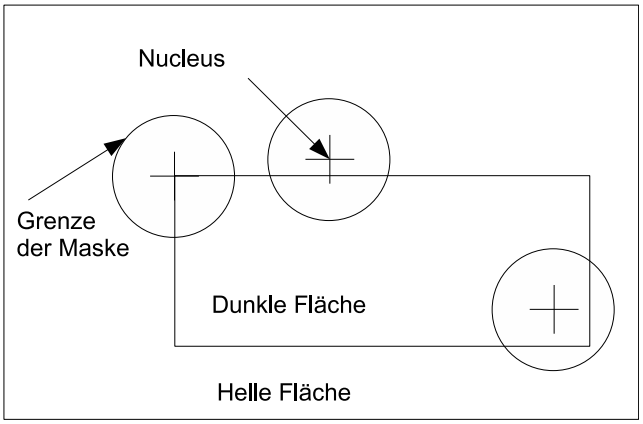


Abb. 6: *Kreismasken auf einem Bild mit einem rechteckigen Objekt, Quelle: [5]*

eine zweidimensionale kreisrunde Maske. Die Standardgröße beträgt dabei 37 Pixel mit einem 3.4 Pixelradius, es kann aber auch eine 3×3 Maske verwendet werden. Die Farbintensität I jedes Pixels $\vec{r}$ wird mit dem Nucleus $\vec{r_0}$ verglichen und das Ergebnis berechnet sich wie folgt:

$$c(\vec{r}, \vec{r_0}) = e^{-\left(\frac{I(\vec{r})-I(\vec{r_0})}{t}\right)^6} \tag{7}$$

Der Funktionsverlauf ist in Abbildung 7 zu sehen. Dies wird für alle Pixel zu $n(\vec{r_0})$ aufsummiert.

$$n(\vec{r_0}) = \sum_{\vec{r}} c(\vec{r}, \vec{r_0}) \tag{8}$$

Damit berechnet sich die Größe der USAN in Pixel. Als nächstes wird n mit einem festen Grenzwert $g = 3n_{max}/4$ verglichen. Dabei ist n_{max} die Anzahl

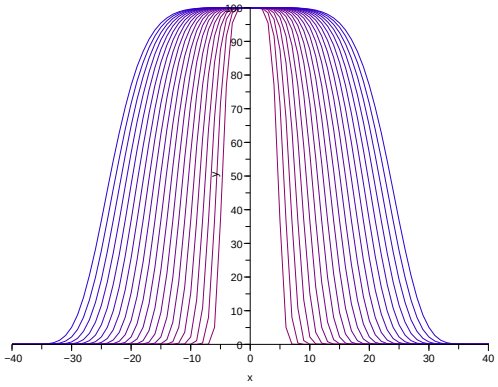


Abb. 7: *Vergleichsfunktion $c(\vec{r}, \vec{r_0})$ auf y-Achse und Farbintensitätsunterschied auf der x-Achse, von blau nach rot wird $t = 5$ bis $t = 25$ erhöht, je größer t umso stärker muss der Intensitätssprung sein, um einen weiteren Punkt der USAN zu identifizieren, Quelle: eigene Darstellung*

aller Pixel in der Maske. Die Stärke eines Kantenpunktes R ist somit:

$$R(\vec{r_0} = \begin{cases} g - n(\vec{r_0}), & n(\vec{r_0}) < g \\ 0, & \text{andernfalls} \end{cases} \tag{9}$$

Anschließend wird die Kantenrichtung bestimmt. Die Kantenrichtung ist rechtwinklig zu dem Vektor zwischen dem Schwerpunkt der USAN und dem Nucleus, wobei für den Schwerpunkt gilt:

$$\bar{\vec{r}}(\vec{r_0}) = \frac{\sum_{\vec{r}} c(\vec{r}, \vec{r_0})}{\sum_{\vec{r}} c(\vec{r}, \vec{r_0})} \tag{10}$$

Falls Schwerpunkt und Nucleus innerhalb des gleichen Pixels liegen, wird das Flächenträgheitsmoment der USAN zur Richtungsbestimmung verwendet. Mit Hilfe der Kantenrichtung kann nun *non-maximum suppression* angewandt werden um die Kante auszudünnen.

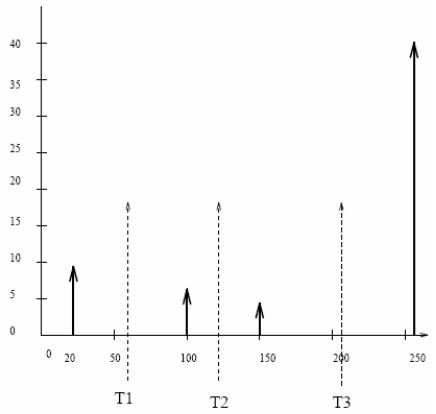
2.2.2 Bereichssegmentierung

Hier haben sich 2 Varianten entwickelt: *seed segmentation*und *region growing*.

Bei der **seed segmentation** betrachtet man zu erst das Histogramm der (eindimensionalen) Farbwerte im Bild. Histogramme sind Diagramme, die die Häufigkeit jedes Farbwerts im Bild anzeigen. Farblich zusammenhängende Bildbereiche resultieren in jeweils in einem Ausschlag (*peak*) im Histogramm (siehe Abbildung 8). Durch Bildrauschen können diese Ausschläge nicht identifizierbar

255	255	255	255	255	255	255	20
255	255	255	100	100	255	20	20
255	255	255	100	100	255	20	20
255	255	255	100	100	255	20	20
255	255	255	255	255	255	20	20
255	255	255	255	255	255	255	255
150	150	255	255	255	255	255	255
150	150	255	255	255	255	255	255

(a)



(b)

Abb. 8: *(b): Ausschläge im Histogramm, (a): deuten zusammenhängende Bereiche im Bild (a) an, Quelle: [4]*

sein. Dann wird beispielsweise der *peakiness test*[4] durchgeführt, um nur die höchsten und schärfsten Ausschläge im Histogramm identifizieren zu können. Es gilt dafür[4]:

$$\text{Peakiness} = \left(1 - \frac{V_a + V_b}{2P}\right) \times \left(1 - \frac{N}{W \times P}\right) \tag{11}$$

Dabei ist W die Weite des Ausschlags, P die Höhe, V_a, V_b sind die tiefsten Talpunkte links und rechts des peaks und N ist die Anzahl der Pixel im Bild, die vom peak erfasst werden. Anschließend kann zwischen je zwei benachbarten Ausschlägen ein Grenzwert (*seed*) im Tal gewählt werden (siehe Abbildung 8: T_1 bis T_3), um beide Bereiche von einander abgrenzen zu können. Zum Schluss wird pixelweise das Bild durchlaufen, um zusammenhängende Farbbereiche im Bild zu finden, die zwischen zwei der gewählten Farbgrenzen liegen.

Das **region growing** hingegen verwendet keine Informationen aus dem Histogramm. Beim *split and merge*[4] Algorithmus wird bei jedem Programmdurchlauf das Bild in vier Teile (zum Beispiel Quadrate) geteilt, wenn der zu teilende

Bereich nicht in der vorab definierten Eigenschaft konsistent ist. Nach jedem Teilen werden alle die benachbarten Unterquadranten zusammengefasst, die in der gleichen Eigenschaft übereinstimmen (siehe Abbildung 9). Bei anderen

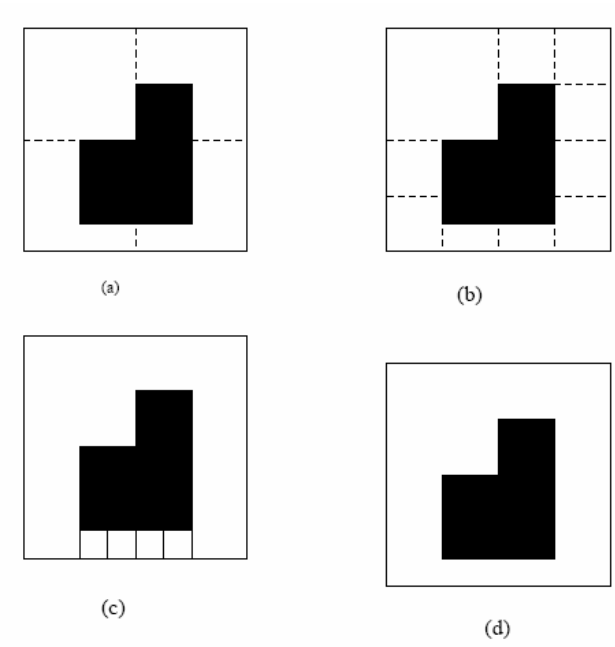


Abb. 9: *Split-And-Merge-Algorithmus von (a) bis (d) bis ein Bereich mit einer gemeinsamen Eigenschaft gefunden wurde, Quelle: [4]*

Ansätzen wird von einer ausgehenden Bereichswahl ausgegangen (zum Beispiel von einer *seed segmentation*). Dabei ist zu beachten, dass je nach Wahl der ausgehenden Bereiche das Endergebnis sich ändern kann. Dort werden dann all die benachbarten Regionen zusammengefasst, bei denen die Grenze zweier benachbarter Regionen schwach ist. Ein Vertreter hierfür ist der *phagocyte* Algorithmus.[4] Für diese Stärke S eines Grenzpunktes zwischen zwei Punkten A, B aus zwei benachbarten Regionen gilt[4]:

$$S(A, B) = |f(x_1, y_1)f(x_2, y_2)| \tag{12}$$

Für die Schwäche W eines Grenzpunktes zwischen zwei Punkten A, B aus zwei benachbarten Regionen gilt[4]:

$$W(A, B) = \begin{cases} 1, & S(A, B) < T_1 \\ 0, & \text{andernfalls} \end{cases} \tag{13}$$

Mit T_1 ist ein Grenzwert. Für die Schwäche W einer ganzen Grenze zwischen zwei benachbarten Regionen gilt[4]:

$$W(\text{Grenze}) = \sum_{\forall A,B} W(A,B) \tag{14}$$

Es werden nur dann zwei benachbarte Regionen verschmolzen, wenn gilt[4]:

1. $\frac{W(\text{Grenze})}{min(P_1,P_2)} > T_2, 0 \leq T_2 \leq 1$, mit P_i ist Umfang von Region i
2. $\frac{W(\text{Grenze})}{\text{Anzahl der Punkte an der Grenze}} > T_3, 0 < T_3 < 1$

2.2.3 Vergleich

Trotz der Tatsache, dass wir von Farbbereichen Kantenpunkte extrahieren können und von Kanten auf Bereiche schließen können, liefern beider Ansätze Unterschiedliche Ergebnisse. Das liegt an den folgenden unterschiedlichen Eigenschaften: [4]

1. Kantenzüge eines Kantendetektion müssen nicht geschlossen sein, wohingegen Bereiche abgeschlossen sind.
2. Bereichssegmentierung ist global und Kantenextraktion geschieht lokal.
3. Die Positionen von Bereichsgrenzen eines Bildes können variieren, die von Kantenpunkten nicht.

2.3 Objektidentifikation

Die Objektidentifikation hat zur Aufgabe besondere Merkmale eines Objektes zu bestimmen, um in einem neuen Bild nach dieser Kombination von Eigenschaften (*features*) zu suchen. Beim Auffinden dieser Kombination, hat man das Objekt wiedererkannt. Um den klassischen Ansatz der Objektidentifikation und -wiedererkennung im Bild zu verstehen, benötigt man das Wissen über die Gauß- und Laplace-Pyramide.
Für beide Pyramiden gilt der gleiche Aufbau. Jede Ebene besitzt eine quadratische Auflösung. Jede um eins höhere Ebene in der Pyramide ist genau in jeder Dimension um die Hälfte kleiner (die niedrigste Ebene besitzt die höchste Auflösung). Damit besitzt jede Pyramide maximal binär logarithmisch (von der maximalen Auflösung) viele Ebenen (siehe Abbildung 10)

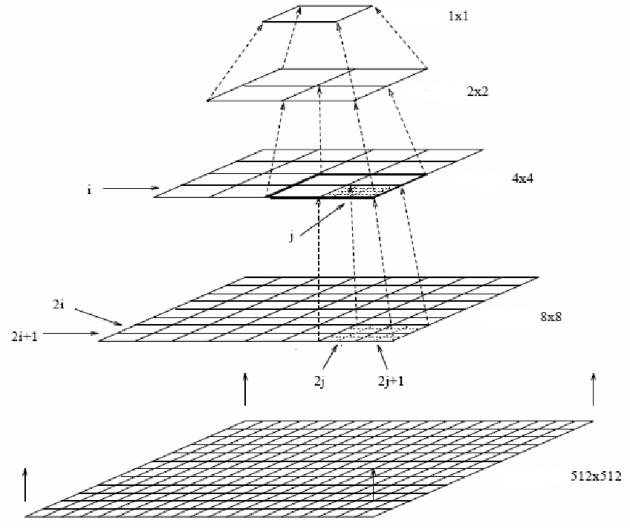


Abb. 10: *Allgemeiner Pyramidenaufbau, Quelle: [4]*

Die **Gauß-Pyramide** besitzt auf der untersten Stufe g_0 das zu bearbeitende Bild. Dabei ist zu beachten, dass eine quadratische Ausgangsauflösung des Bildes zu wählen bzw. aus dem Bild auszuschneiden etc. Jede Stufe g_l berechnet sich aus der um eins kleineren Stufe g_{l-1} mit Hilfe einer 5×5 Gauß-Maske. Es gilt:[4]

$$g_l(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{l-1}(2i+m, 2j+n) \tag{15}$$

Dabei ist $w(m,n)$ der Wert in der Gauß-Maske an Position (m,n). Die **Laplace-Pyramide** berechnet sich aus der Differenz aus dem g_i Bild und dem expandierten g_{i-1} Bild der Gaußpyramide. Dabei berechnet sich das expandierte Bild $g_{k,l}$ aus:[3]

$$g_{k,l}(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{k,l-1} \frac{(2i+\frac{m}{2}, 2j+n)}{2} \tag{16}$$

Ein Beispiel für die Pyramiden ist in Abbildung 11 gegeben. Somit repräsentiert jeder Pixel einer Laplace-Pyramide das Ergebnis der Differenz zweier Gauß-Funktionen angewandt auf das original Bild. Man beachte, dass diese Differenz äquivalent zur Anwendung des Laplace-Gauß-Operators auf das Bild ist.[**?**]
Die Methode *Distinctive Image Features from Scale-Invariant Keypoints*[2] wird

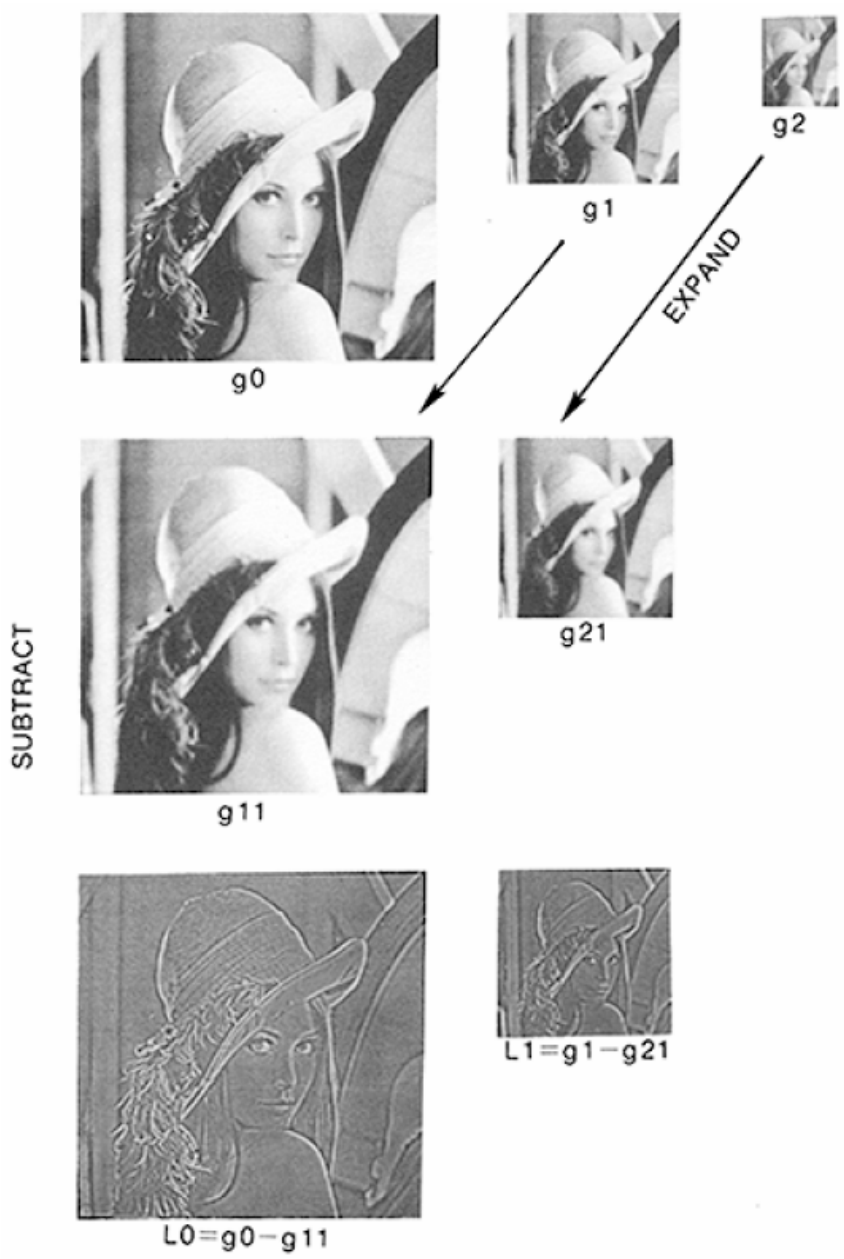


Abb. 11: *Ein Beispiel der Gauß- und Laplace-Pyramide, Quelle: [3]*

zur Objektwiedererkennung verwendet und basiert auf der Gauß-Pyramide. Jede Ebene der Pyramide wird mehrmals mit dem Gaußfilter prozessiert und jedes Ergebnis in einem Bild der Ebene festgehalten (Oktave genannt). Dann wird jeweils die Differenz zweier solcher benachbarter Bilder innerhalb der selben Oktave gebildet (siehe Abbildung 12). Als Ergebnis erhält man dann die Differenz *Difference of Gausian (DOG)*. Ein ganz ähnliches Ergebnis zum DOG

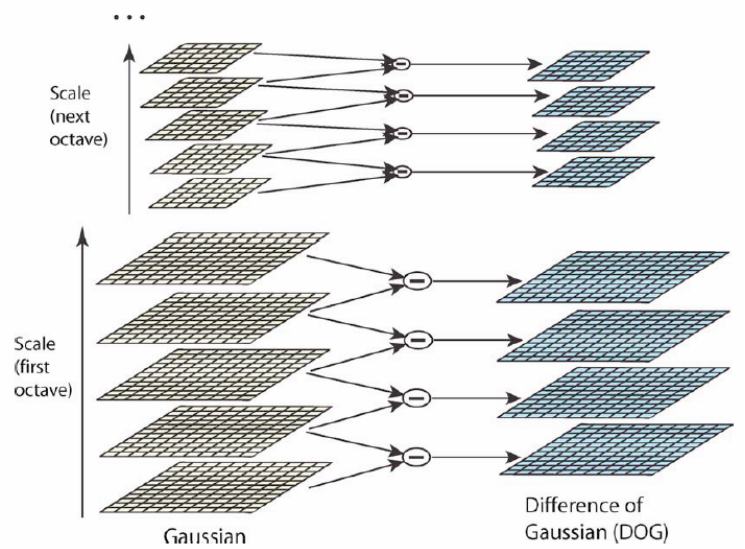


Abb. 12: *Difference of Gaussian, Quelle: [2]*

liefert die Laplace-Pyramide. In dieser wird auf jeder Ebene ein Kantenbild erzeugt, das umso feinere Kanten enthält, je höher die Auflösung der Ebene ist. Anschließend werden in jeder Oktave des DOG die lokalen Extrempunkte (extreme Kantenpunkte) gesucht (siehe Abbildung 13). Diese haben nämlich die Eigenschaft unabhängig von Bildauflösung und Rotation zu sein und gute Robustheit bei Verzerrung zu liefern. Das Ziel ist es solche Kantenpunkte (*Features*) einerseits im Bildausschnitt des zu identifizierenden Objektes und andererseits der dann zu untersuchenden Szene zu extrahieren. Es wird dann eine Datenbank an Suchobjekteigenschaften (Extrempunkte und Beziehungen untereinander) erstellt, die während der Objektsuche im Bild nach Treffern durchsucht wird. Dafür wird zu Beginn ein Kantenpunkt (mit seinen Eigenschaften wie Richtung und Gradientenstärke) aus der Suchszene in der Datenbank gesucht und dann iterativ der nächste Nachbar prozessiert.

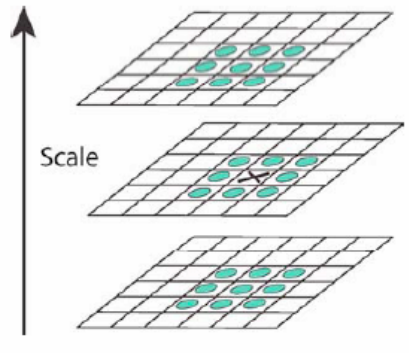


Abb. 13: *Lokale Extrempunkte werden in jeder Oktave für alle 26 Nachbarpixel gesucht, Quelle: [2]*

2.4 Auswahl

Um eine Auswahl der klassischen Methoden treffen zu können, spielt das Nebenziel, dass eine ganze Programmschleife in deutlich unter einer Sekunde abgearbeitet wird, eine große Rolle.
Da in den vorgestellten Methoden davon ausgegangen wird, dass das Eingabebild im eindimensionalen Format vorliegt, wird das RGB24 Bildformat in ein 256stufiges Bildformat umgewandelt, dass dem Y-Kanal entspricht.
Zwar liefert ein Bereichssegmentierungsalgorithmus eine ganze Objektkontur, jedoch ist nicht gewährleistet, das die Grenzen des Objektes unter (fast) gleichen Lichtverhältnissen immer gleich lokalisiert werden. Es könnte womöglich ein und das gleiche Objekt anders wahrgenommen werden. Deswegen wird im folgenden die Kantenerkennung verwendet. Bei den Kantendetektoren fällt die Wahl auf den **SUSAN-Algorithmus**. Hier wird jedoch wegen der Zeitanforderung von der Benutzung der SUSAN-Rauschfilterung abgesehen und die kleinste Maskengröße, nämlich die 3×3 Maske, verwendet. Weiterhin wird aus besagtem Grunde keine Kantenrichtung extrahiert und somit auch keine *non-maxima suppression* angewandt.
Trotz der viel versprechenden Ansätze bei der **Objektidentifikation** wird, aufgrund ihrer intensiven Rechenleistung zur Bestimmung und Verwendung der Pyramiden und des Fehlens der Kantenrichtungen, von keinem dieser Ansätze Gebrauch gemacht. Somit muss ein eigener intuitiver Algorithmus zur Objektidentifikation und -wiedererkennung konzipiert werden.

3 Implementation

In diesem Kapitel werden die einzelnen Module erstellt und analysiert. Als Resultat wird ein Programm erzeugt, dass diese Module implementiert und der Zielsetzung gerecht werden soll.

3.1 Farbraumreduktion

Im Folgenden werden verschiedene Ansätze bei der Transformation des Farbraumes besprochen und kurz analysiert.
Da alle weiteren Algorithmen auf eindimensionalen Farbwerten arbeiten, wird der dreidimensionale RGB Farbraum mit 256^3 Werten auf 256 Werte reduziert werden. Dabei sollten die Eigenschaften eines Kantendetektors (das nächst höhere Modul) berücksichtigt werden. So sollten die Kontraste der dreidimensionalen Objekte der realen Szene bestmöglich auf die 256 Farbwerte reduziert werden.
Die gängigste Variante den RGB-Raum zu transformieren, ist die Transformation in den Y-Kanal. Der Y-Kanal des YUV-Farbraumes gibt die Helligkeit an, der U-Kanal den Blauanteil und der V-Kanal den Rotanteil. Es gilt dafür:

$$Y := 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{17}$$

Jede statische Reduktion des Farbraumes auf einen kleineren Farbraum hat zur Folge, dass vorher noch visuell trennbare Farben, möglicherweise den gleichen Farbton zugewiesen bekommen (siehe Abbildung 14). Ein anderer Ansatz wäre

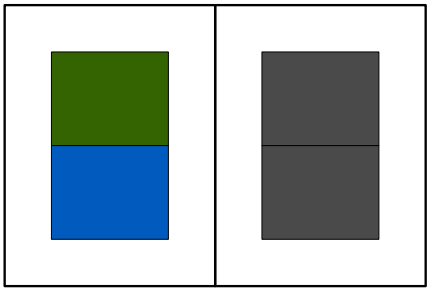


Abb. 14: *Informationsverlust durch Farbraumreduktion: linkes Objekt besitzt RGB-Farbtöne: [50, 100, 0] und [0, 90, 190] , Quelle: eigene Darstellung*

eine zufällige Verteilung der 3 Farbkanäle auf den „Zielkanal“ zu wählen um den lokalen Informationsverlust im Bild in gewissen Zeitfenstern zu vermeiden und

Kontraststellen zu erhalten. Weiterhin bestünde noch die Option einzeln die RGB-Kanäle weiter zu verwenden und eine Art *saliency map* auf den Kanälen zu erstellen. Jedoch müsste dann der Kantendetektor über alle 3 Kanalbilder laufen. Aus Performancegründen kann dies nicht weiterverfolgt werden.

3.2 SUSAN-Kantendetektor[5]

Es werden unter den Aspekten der Qualität die verschiedenen Einstellungen vom SUSAN-Kantendetektor getestet und die Ergebnis gegenübergestellt.
In dem von Steven Smith implementierten SUSAN-Algorithmus[6] existieren mehrere Optionen zur Feineinstellung. So kann man zwischen der Maskengröße von 37 Pixeln (kreisförmig angeordnet) und einer Maske der Größe 9 (3×3) wählen. Wenn man beide Ergebnisse mit einander vergleicht (siehe Abbildung 15 Bilder b und d) sieht man den nahe liegenden Unterschied. Die 3×3 Maske kann natürlich nur *step edges* bzw. die Flanken einer *spike edge* im Abstand eines Pixels feststellen. Wo hingegen die 37 Pixel-Maske Kanten im Radius von bis zu 3 Pixeln feststellen kann. Im Vergleich zur 3×3 Maske verursacht die 37 Pixelmaske einen berechneten 4 mal höheren Aufwand, der auf dem Testrechner bei der Programmausführung sich lediglich verdoppelt hat.
Bei Aktivierung der *non-maximum suppression* kann festgestellt werden, dass die erhaltenen Resultate (siehe Abbildung 15 Bilder c und e) selbst bei genauerem Betrachten nicht nennenswert sind und somit für die Systemintegration später nicht verwendet werden.

3.3 Objekterkennung

Nachdem Kanten von Objekten aus einem eindimensionalen Farbraum extrahiert werden konnten, müssen jetzt zusammenhängende Kantenzüge gefunden werden um abgeschlossene Objekte aus dem Bild erkennen zu können. So werden intuitive Ansätze besprochen und eine Auswahl daraus selektiert. Als nächstes soll aber das **Erkennen von Objekten** innerhalb dieser Arbeit näher definiert werden:

- Objekte im Bild werden die Bereiche definiert, die von einem vollständigem Kantenzug umgeben sind. Damit müssen die Objekte vollständig im Bildbereich der Kamera liegen.

- Objekte müssen somit keinen Formen gleichen wie zum Beispiel: Kreise, Rechtecke etc.

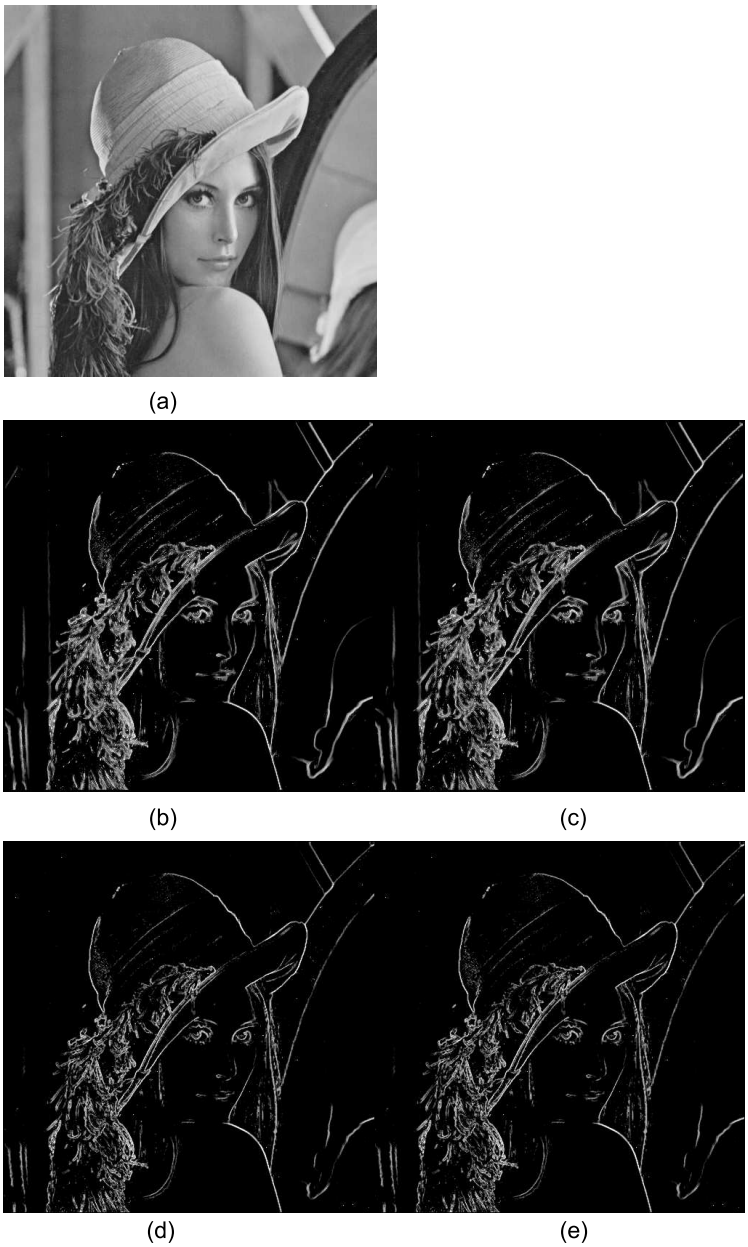


Abb. 15: *Gegenüberstellung der Optionen des SUSAN-Algorithmus, (a): Originalfoto, (b) und (c): 37 Pixelmaske mit und ohne non-maximum suppression, (d) und (e): analog nur* 3×3 *Pixelmaske, Quelle: eigene Darstellung*

Da die Objekterkennung dieser Arbeit formunabhängig sein soll, können Ansätze wie zum Beispiel die Houghtransformation nicht angewendet werden. Bei der Houghtransformation handelt es sich um ein Voting-Verfahren für festgelegte Formen, die im Bild gesucht werden. So werden per Mehrheitsfindung Formen wie zum Beispiel Linien beliebiger Steigung über alle Punkte gelegt. Danach werden die Linien übernommen, die sich die meisten Punkten teilen. Wie schon früher besprochen, erhalten wir vom Kantendetektor Kanten ohne Richtung. Somit kann die Kantenvervollständigung in Richtung der Kante hier nicht stattfinden. Im Folgenden wird festgelegt, dass nur die Objekte erkannt werden sollen, die sich durch einen vollständige Silhouette (vollständiger Kantenzug) im Bild hervorheben.
Um dieses Ziel zu erreichen könnten zwei Ansätze verfolgt werden. Die erste Option bestünde darin, entlang aller Kantenzüge nach einem sich schließenden Zyklus zu suchen. Dadurch könnten auch Objekte hierarchisch in ihre Unterobjekte (so denn vorhanden) untergliedert werden. Jedoch müsste man hierbei sicherstellen können, dass auch wirklich eine nicht leere Punktmenge (die keine Kante ist) umschlossen wird. Zweitens müsste dann der Kantenzug noch verdünnt werden um ihn auf die „echten“ Grenzen des Objektes zu reduzieren. Der andere und in dieser verwendete Ansatz besteht darin, dass man die zusammenhängenden Bereiche im Bild findet, die keine Kanten sind und vollständig von Kantenpunkten umschlossen sind. Darauf werden die Kantenpunkte als Objektgrenze genommen, an die das Objekt anstößt. Zwar müssen im Durchschnitt deutlich mehr Pixel im Bild prozessiert werden, als bei der ersten Variante, nur ist die Umsetzung äußerst trivial. Typischerweise verwendet man hierfür einen *floodfill* Algorithmus. Der Ablauf des Algorithmus besteht darin, alle Nachbarn eines Pixels der keine Kante ist einer Klasse zuzuordnen und das dann entweder rekursiv oder iterativ zu implementieren. Bei der rekursiven Methode müsste stets darauf geachtet werden, keinen *stack overflow* zu erzeugen. Somit fällt die Wahl auf die iterative Methode. Als Nachbarn gelten nur die Pixel, die über die 4 Himmelsrichtung (und nicht alle 8 Nachbarn) an den Pixel angrenzen. Der Rechenaufwand der rekursiven und der iterativen Lösung hat sich auf dem Testrechner als gleich herausgestellt, wenn nicht sogar der rekursive Algorithmus (trotz extra Behandlung der Vermeidung eines Stapel-Überlaufs) ein wenig schneller lief. Trotzdem wird im Weiteren die sichere iterative Lösung verwendet.
Innerhalb dieses Moduls werden somit die einzigen beschreibenden Eigenschaften (ausschließlich Kantenpunkte) der Objekte gesammelt. Es wird sich zeigen, dass allein mit Hilfe dieser Information die Zielsetzung erreicht werden kann.

3.4 Objektidentifizierung und -wiedererkennung

Nun da im vorherigen Modul Objektinformationen aus einem Bild extrahiert werden konnten, wenden wir uns der Identifizierung und Wiedererkennung der Objekte zu, um später ein ausgewähltes Objekt unter den anderen wiederfinden zu können.
So wurde schon bei den Grundlagen erwähnt, dass die Suchobjekte im Bild translatiert, rotiert, anders skaliert oder gar verzerrt sein können. Um Objekte trotz Translation im Bild wiedererkennen zu können müssen dem Objekt Eigenschaften entnommen werden, die es eindeutig identifizieren. So verwendet die in den Grundlage vorgestellte Methode Beziehungen zwischen Extrempunkten und ihren Eigenschaften des Objektes. Da in dieser Arbeit nicht die Kantenpunktstärke oder -richtung bekannt sind, wurde in dieser Arbeit der Schwerpunkt $\vec{s}$ aller Kantenpunkte $\vec{k_i}$ jedes Objektes berechnet:

$$\vec{s} = \frac{\sum_{i=1}^{n} \vec{k_i}}{n} \tag{18}$$

Mit dem Schwerpunkt kann einerseits ein Histogramm über alle Abstände der Kanten zum Schwerpunkt berechnet werden. Anschließend können dann zwei Histogramme zweier Objekte auf Gleichheit getestet. Damit wäre dann auch das Problem der Rotation gelöst. Denn dieses Histogramm ist unabhängig von Rotationen. Um dann noch das Suchobjekt trotz Größenveränderungen im Bild detektieren zu können, müsste jedoch das Histogramm des Suchobjekte in den verschiedensten Auflösungsstufen vorliegen. Diese müssten dann alle mit den im Bild detektierten Histogrammen verglichen werden. Die Umsetzung dieser Problematik würde jedoch intensivstes Berechnen voraussetzen und widerspricht somit der Zielsetzung der Erreichung einer Hochgeschwindigkeitskomponente. Deswegen wird im Folgenden die Idee der Verwendung von Histogrammen nicht berücksichtigt.
Es wurde in dieser Arbeit ein anderes Konzept gewählt. Für die **Translation** allein könnte man für jedes Objekt den durchschnittlichen Abstand r_d aller Kanten zum Schwerpunkt bestimmen. Dieser Abstand r_{mean} kann jedoch bei unterschiedlichen Objekten gleich groß sein (siehe Abbildung 16 gelber Radius). Deshalb wurde noch der kleinsten Abstand r_{min} und der größten Abstand r_{max} von allen Kanten zum Schwerpunkt betrachten (siehe Abbildung 16 grüner und roter Radius). Viel wichtiger sind r_{min} und r_{max} für das Wiederfinden eines Objektes im Bild trotz **Größenskalierung**. Denn das Verhältnis ist in jeder Größenskalierung im fast identisch (wegen Diskretisierung der Pixel auf bestimmte Auflösung der Kamera). Weitere rotationsunabhängige Verhältnisse, die andere Objekteigenschaften beschreiben sind: r_{min} zu r_{mean} und das

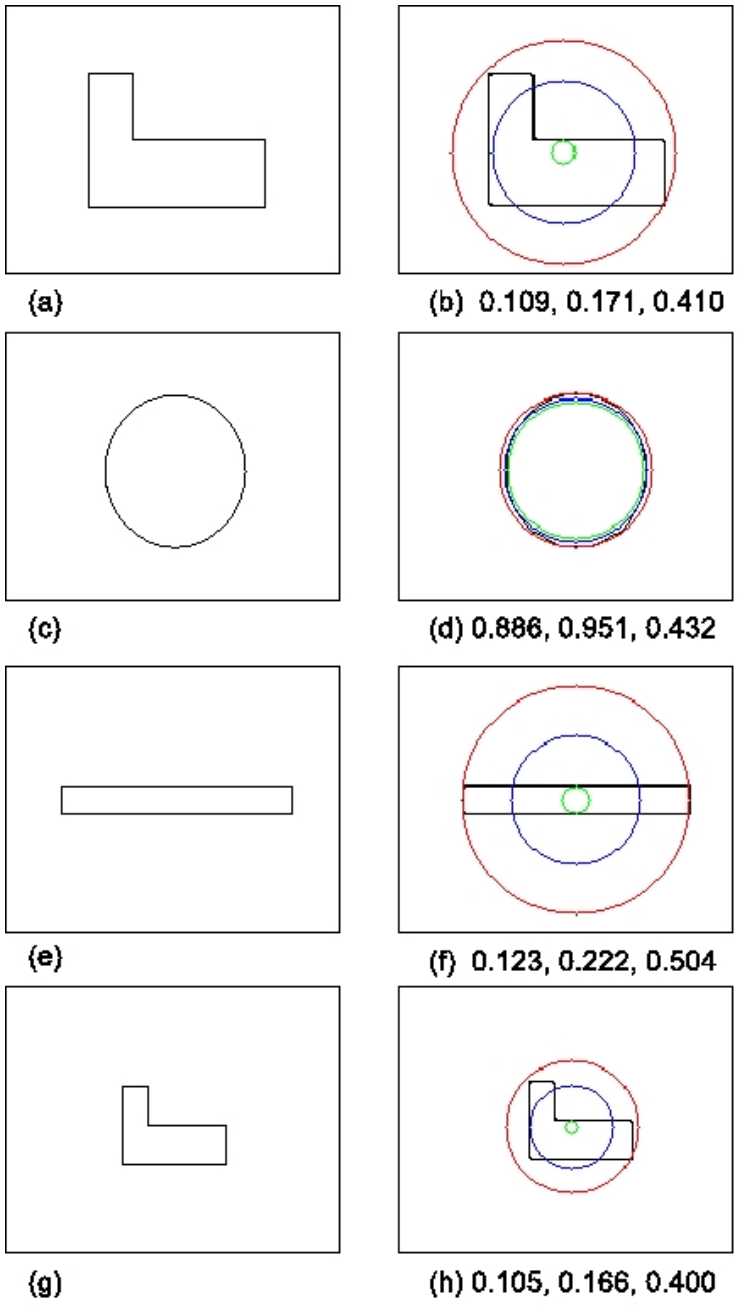


Abb. 16: *Objektidentifizierung anhand durchschnittlicher Radien im Bild, (a): original Bild, (b): grüner, blauer und roter Radius für kleinsten, mittleren und maximalen Radius aller Kanten zum Schwerpunkt, Angaben der Proportionen* $\frac{r_{min}}{r_{max}}$, $\frac{r_{min}}{r_{mean}}$, $\frac{\#(r_{edge_i}<r_{mean})}{all\ edges}$, *Quelle: eigene Darstellung*

Verhältnis aller Kantenpunkte mit einem Abstand zum Schwerpunkt $< r_{mean}$ zu allen Kantenpunkten. Die Proportionen in der Übersicht:

$$\frac{r_{min}}{r_{max}}, \frac{r_{min}}{r_{mean}}, \frac{\#(r_{edge_i} < r_{mean})}{\text{all edges}} \tag{19}$$

Dieser Proportionen sind der „Fingerprint“ eines jeden Objektes, durch diesen es identifiziert wird. Das Problem der Objektfindung trotz perspektivischer Verzerrung bleibt bestehen, solange nur Objektkanten aus einem einzigen Bild extrahiert werden. Wenn jedoch das Objekt von verschiedenen Richtungen aus (mit Verzerrung) aufgenommen wird, so dass viele Kantenzüge bei verschiedenen Verzerrungen gespeichert werden können, kann das Problem zumindest angegangen werden. Jedoch muss dann mit einer erhöhten False-Positive-Rate gerechnet werden.
Nachdem für ein speziell ausgewählte Suchobjekt im Bild die Identifikation durchgeführt wurde, kann anschließend jedes erkannte Objekt mit seinem Fingerprint mit dem des gesuchten Objektes verglichen werden. Innerhalb eines noch nicht festgelegten Grenzwertes werden dann nach Treffern gesucht.

3.5 Überblick

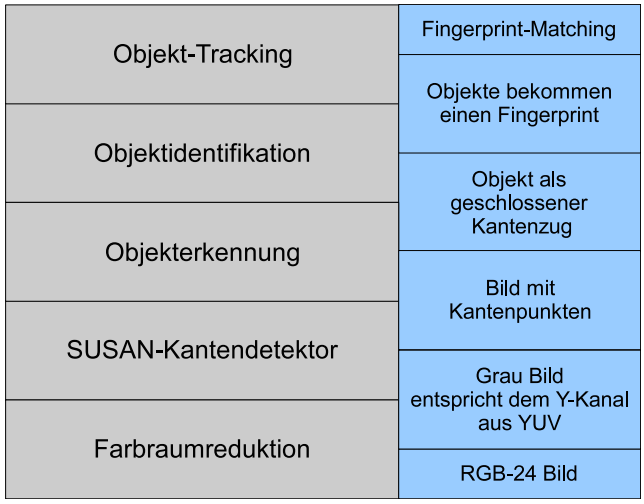


Abb. 17: *links: Modularer Aufbau des Programms von unten nach oben gelesen, rechts: Ein- und Ausgabe jeder Schicht, Quelle: eigene Darstellung*

4 Resultate

Mit der jetzigen Funktionalität (siehe Abbildung 17) ist es möglich Kantenobjekte zu erkennen, die aus der dreidimensionalen Szene stammen. Diese Objekte aus der Szene müssen monoton gefärbt sein und können eine beliebig geformte Kontur besitzen, so denn sich diese kontrastreich vom Hintergrund abhebt. Weiter wird in jedem neuen Bild nach schon bekannten Konturen gesucht, um die Suchobjekte wiederfinden zu können.

Um die Güte des Algorithmus bestimmen zu können, muss die Güte der beiden Hauptfunktionalitäten bestimmt werden. Diese sind einerseits die Erkennung und andererseits die Wiedererkennung der Objekte.
Die **Güte der Erkennung** von Objekten ist ausschließlich durch die Güte der SUSAN-Kantenerkennung definiert. Dabei ist die Erkennung von Kantenpunkten im Kamerabild abhängig von dem Schwellwert der Vergleichsfunktion (siehe Abbildung 7). Smith und Brady haben in ihrem Artikel „SUSAN-A New Approach to Low Level image Processing“ durch eine Vielzahl von Tests das Verhalten der SUSAN-Kantenerkennung sehr genau beschrieben und ausgewertet. Im Folgenden werden einige wenige dieser Ergebnisse präsentiert. Die verschiedenen Kantentypen in einem Bild und die erkannten USANs sind in Abbildung 18 zu erkennen. Die Kanten als Schwarz-Weiß-Schwarz-Übergang in einem Straßenbild und einem digital erstellten Bild sind in Abbildung 19 dargestellt.
Um die **Güte der Wiedererkennung** der Suchobjekte bestimmen zu können, muss die Wiedererkennungsrate des Suchobjektes zu erst bei statischer Kameraposition und Objektgröße und dann jeweils in Anwesenheit von Translation, Rotation, Größenänderung und Verzerrung im Kamerabild bestimmt werden.
Vorab wird der Ablauf der Wiedererkennung festgehalten und Eigenschaften definiert:

1. Die Phase des *Objekt-Scannens*: Hier wird mittels der Kamera das Objekt abgelichtet und alle dem Programm unbekannte Objektkonturen (Kantenzüge) und ihre Fingerprints werden dann im Speicher hinterlegt. Unbekannt sind die neuen gesehenen Fingerprints, die um einen gewissen Prozentsatz von allen im Speicher Befindlichen abweichen. Hier wird der Ansatz verfolgt, den Prozentsatz so zu wählen, dass aus einem festen Blickwinkel bei gleicher Objektgröße alle neuen gesehenen Fingerprints dem Fingerprint des zu erst detektierten Kantenzuges zugeordnet werden. Dies führt dazu, trotz anders detektierter Kantenpunkte (bei Rauschen) das Suchobjekt gefunden werden kann. Es ist hier darauf zu achten, dass

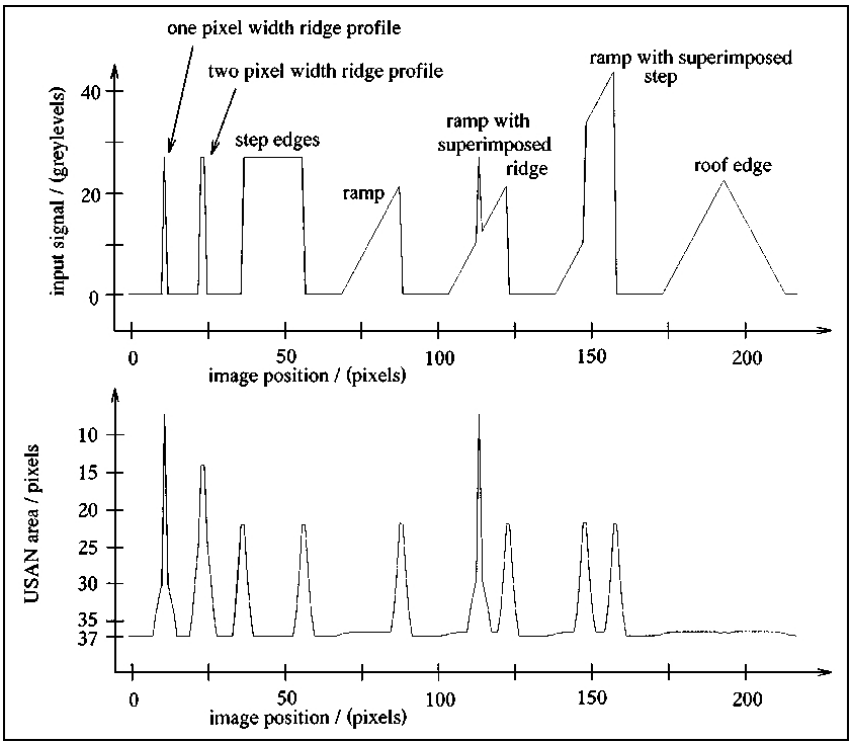


Abb. 18: *Die Erkennung der verschiedenen Kantentypen mit Hilfe des SUSAN-Kantendetektors, Quelle: [5]*

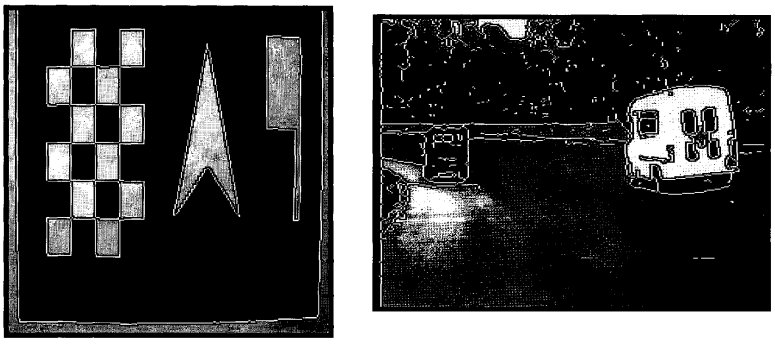
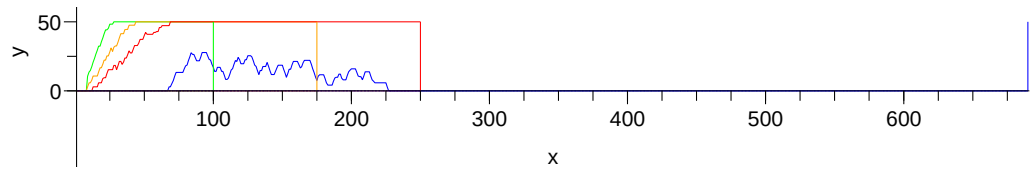


Abb. 19: *Links: Kanten im Straßenbild, Rechts: Kanten im digitalem Bild, Quelle: [5]*

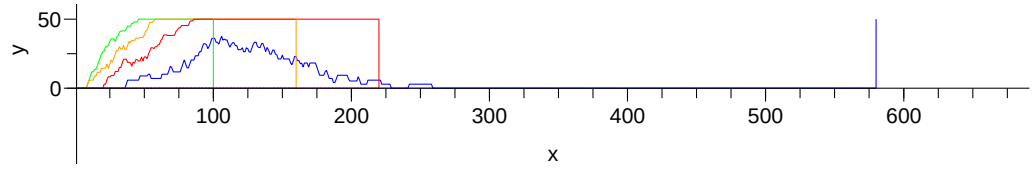
nur das zu scannende Objekt im Kamerabild enthalten ist, damit nur dessen Kantenzug vom Programm detektiert wird.

2. Die Phase der *Objektwiedererkennung*: Hier wird jeder gesehener Objektkantenzug mit seinem Fingerprint mit denen im Speicher Befindlichen verglichen und bei Übereinstimmung innerhalb des Prozentsatzes wird das Objekt als wiedererkannt markiert.

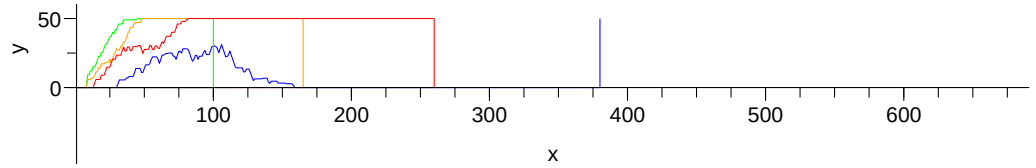
Bei unverändertem Blickwinkel und unveränderter Größe hat sich gezeigt, dass eine prozentuale Abweichung von zwei Prozent des ersten gemessenen Fingerprints des Objektes im Schnitt ausreicht, um in einem festen Blickwinkel auf dasselbe unveränderten Objekt innerhalb 300 aufeinander folgender Ablichtungen, immer wieder dieses wiedererkennen zu können, ohne ein weiteren Kantenzug mit seinem Fingerprint im Speicher zu hinterlegen. Die Wiedererkennung eines translatierten und oder rotierten Objektes ist durch den Ansatz der Erkennung eines Objektes durch seine Objektgrenzen gegeben und bedarf keiner weiteren Nachforschungen.
Um die Wiedererkennungsrate des Suchobjekts bei verschiedenen Größen bestimmen zu können, wurde der folgende Versuchsaufbau und -ablauf durchgeführt:
Es wurden 3 verschiedene schwarze Flächen (Rechteck, Ellipse und Dreieck) auf weißem Hintergrund auf einen LCD-Bildschirm projiziert. Die Kamera befand sich 20 cm entfernt vom Bildschirm auf mittlerer Bildschirmhöhe und -breite und wurde auf eine dem Monitor geeignete Einstellung geeicht (siehe Anhang: Kamera-Einstellungen). Diese drei Objekte wurden jeweils in 4 verschiedenen ausgehenden Größenstufen „gescannt“. Dabei wurde nicht der Abstand der Kamera zum Bildschirm variiert, sondern die Größe der Objekte auf dem Bildschirm. Anschließend wurden die Objekte in jeder der vier Stufen von der minimalen zur maximalen Größe skaliert, die vom System noch wahrgenommen wird. Dafür wurde das (skalierte) Objekt immer so zur Kamera ausgerichtet, dass der Schwerpunkt des Objektes im Mittelpunkt des Kamerabildes liegt. Somit wurde gewährleistet, dass der Blickwinkel auf dasselbe Objekt um maximal $1/320 \times 1/240$ (horizontaler Öffnungswinkel multipliziert mit vertikalem Öffnungswinkel der Kamera) abweicht. Währenddessen wurde die Rate der Wiedererkennung gemessen. Die Messung stützt sich auf 50 aufeinander folgende Bildschirmablichtungen (printscreens) bei denen die Anzahl der Wiedererkannten und die Anzahl nicht Wiedererkannten gezählt wurden. Es wurde festgestellt, dass die Ausgangsgröße beim Scannen des Objektes signifikanten Einfluss auf die Wiedererkennung ausübt (siehe Abbildung 20). So ist die Wiedererkennungsrate eines Objektes bei sich ändernden Größenverhältnissen



(a) Wiedererkennungsrate eines Rechtecks bei Größenänderung, Größe des Rechtecks in Scannen-Phase: grün: $18 \times 10cm^2$, orange: $11 \times 6,15cm^2$, rot: $7,8 \times 4,4cm^2$, blau: $2,6 \times 1,5cm^2$



(b) Wiedererkennungsrate einer Ellipse bei Größenänderung, Größe der Ellipse in Scannen-Phase: grün: $17 \times 10,8cm^2$, orange: $11 \times 7cm^2$, rot: $7,7 \times 4,9cm^2$, blau $2,9 \times 1,8cm^2$



(c) Wiedererkennungsrate eines Dreiecks bei Größenänderung, Größe des Dreiecks in Scannen-Phase: grün: $19 \times 11cm^2$, orange: $12 \times 6,8cm^2$, rot: $7,6 \times 4,1cm^2$, blau: $5 \times 2,7cm^2$

Abb. 20: *Wiedererkennung des „gescannten“ Objektes in verschiedener Größenskalierung bei gleichem Blickwinkel, x-Achse: Größenskalierung des Objektes in Prozent, y-Achse: Anzahl der Wiedererkennungen des Objektes bei insgesamt 50 Momentaufnahmen, jedes Diagramm repräsentiert einen ganzen Programmablauf, die Graphen in den Farben: grün, orange, rot und blau repräsentieren die tiefpassgefilterte (der Fenstergröße: 17) Wiedererkennungsrate der jeweils oben aufgeführten Objekten und ihren Größen, dabei handelt es sich jeweils um das selbe Objekt in der Scannen-Phase und Wiedererkennungs-Phase, die senkrechten Linien markieren die maximale Größenskalierung des Objektes die die Kamera noch ablichten kann, Quelle: eigene Darstellung*

umso größer, je größer das Objekt während des Scannen-Phase im Kamerabild erscheint. So wird ein um beispielsweise 50% kleiner erscheinendes Objekt am besten wiedererkannt, je größer das Suchobjekt in der Scannen-Phase ist. Dies liegt daran, dass der Fingerprint des Objektes auf dem durchschnittlichen Abstand aller Kantenpunkte zum Schwerpunkt basiert. Dies ist natürlich umso genauer, je mehr Kantenpunkte des Objektes erkannt werden. Analog dazu wirkt sich bei kleinerer Größe des Suchobjektes während der Scannen-Phase der Fehler von falsch erkannten Kantenpunkten stärker aus. Einerseits können Kantenpunkte durch Rauschen falsch detektiert werden und andererseits ist der Fehler aufgrund der Diskretisierung des Kamerabildes in Pixel bei kleineren Objekten größer als bei größeren Objekten im Kamerabild.
Die Wiedererkennungsrate trotz perspektivischer Verzerrungen insbesondere durch Abweichung vom Blickwinkel in der Scannen-Phase des Suchobjektes konnte nicht experimentell bestimmt werden. Aufgrund der festgelegten Eigenschaft, dass „gescannte“ Objektkonturen innerhalb einer zwei prozentigen Abweichung des Fingerprints wiedererkannt werden, gilt natürlich, dass nur dann verzerrte Objekte einem „gescannten“ Objektkantenzug zugeordnet werden, so denn die Verzerrung in einer maximal zwei prozentigen Abweichung des Fingerprints resultiert. Im Versuch hat sich dabei gezeigt, dass schon bei minimaler Abweichungen von Blickwinkel $1/320 \times 1/240$ das aus einer Kameraposition Suchobjekt kaum wiedererkannt wird.

Da der in dieser Arbeit vorgestellte Algorithmus eine Klassifikation von wiedererkannten und nicht wiedererkannten Objekten vornimmt, müssen auch die falsch positiv wiedererkannten Objekte betrachtet werden. Dazu wird üblicher Weise eine Referenzmenge von Objekten festgelegt. Wenn man alle Objekte dieser Menge scannt und anschließend diese Objekte einzeln jeweils nur einem Referenzobjekt zuweist, kann die Rate der falsch positiv wiedererkannten Objekte bezüglich der Referenzmenge bestimmt werden. Eine Verwendung einer ausreichend großen Menge an Referenzobjekten konnte im Rahmen dieser Arbeit nicht festgelegt werden. In dieser Arbeit wurde hingegen eine Wahl von realen Suchobjekten (Schablonen: Rechteck, Ellipse ,Dreieck) und eine Wahl von realen Testobjekten getroffen. Bei den Testobjekten handelt es sich um dreidimensionale Objekte im Raum der Versuchsdurchführung (siehe Abbildung 21). Hier wurde die Kamera in der Scannen-Phase 20cm über dem Suchobjekt fixiert. Es wurden weiße Schablonen auf grünem Hintergrund verwendet. Das Suchobjekt wurde aus nur einem festem Blickwinkel und einer festen Kamerahöhe „gescannt“ wodurch intern nur eine Referenzkontur mit ihrem Fingerprint für das Objekt im Speicher erstellt wurde. Die verwendete Kamera-Einstellung ist

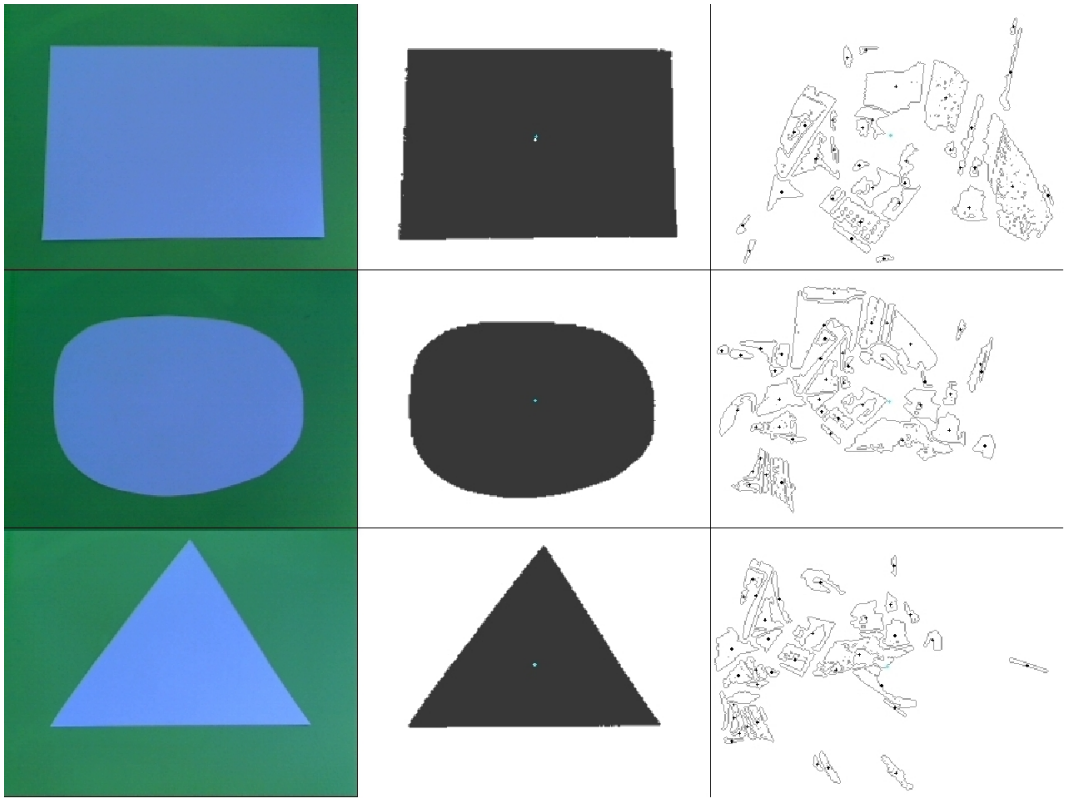


Abb. 21: *Richtig/Falsch positiv wiedererkannte Objekte mit fester Kameraposition während der Scannen-Phase, jede Zeile repräsentiert einen Programmdurchlauf, die erste Spalte zeigt das Kamerabild des weißen Schablonen-Objektes auf grünem Hintergrund bei fester Kameraposition während der Scannen-Phase, die letzten Spalten zeigen das Kamerabild mit den wiedererkannten Objekten (schwarz ausgefüllter erkannter Objektkantenzug mit weißem Kreuz des Objektschwerpunktes) und nicht wiedererkannten Objekten mit schwarzem Kantenzug und schwarzem Kreuz des Objektschwerpunktes während der Objektwiedererkennungs-Phase, Quelle: eigene Darstellung*

im Anhang zu finden. Während der Phase der Wiedererkennung wurde die Kamera auf beliebige Objekte im Innenraum der Versuchsdurchführung ausgerichtet. Dabei konnten keine falsch positiv wiedererkannten Objekte gefunden werden (siehe Abbildung 21).
Scannt man die Objekte jedoch aus mehreren Blickwinkel erhöht sich natürlich auch die falsch positiv Rate der wiedererkannten Objekte, neben dem Anstieg der Wiedererkennungsrate der Suchobjekte trotz Verzerrungen (siehe Abbildung 22).
Kommen wir nun zur Abschätzung der **Zeitkomponente** des Algorithmus. In Experimenten hat sich gezeigt, dass bei bis zu 50 im Speicher befindlichen Fingerprints, also aus 50 verschiedenen Blickwinkeln „gescannte“ Objektabbildungen (Kantenbilder des selben Objektes) kann der Algorithmus jedes dritte Bild (von 15 Bildern pro Sekunde) vollständig abarbeiten. Die Prozessorlast liegt dann bei durchschnittlich 80% im Debug-Modus. Echtzeitfähig ist das jetzige System jedoch nicht, trotz der durchschnittlich geringen Abarbeitungszeiten. Dies ist der Fall, da hier mit dynamisch wachsenden Listen gearbeitet wird, die in ihrer Größe nicht begrenzt sind. Wenn man aber in das System gewisse Grenzen einbaut, könnte sogar Echtzeitfähigkeit implementiert werden, wobei dann natürlich auch die Echtzeitfähigkeit des vorliegenden C++ Frameworks gewährleistet sein muss.

4.1 Überblick

Die wesentlichen Resultate im Überblick:

1. Die Detektion von Kantenpunkten innerhalb des Kamerabildes ist abhängig von dem Schwellwert der SUSAN-Vergleichsfunktion (siehe Abbildung 7) und einige Beispiele der SUSAN-Kantendetektion angewandt auf zwei Bilder sind in den Abbildungen: 18 und 19 zu sehen.

2. Eine Objektkontur wird als nicht wiedererkanntes Objekt klassifiziert, wenn der Fingerprint um mehr als 2% von allen im Speicher liegenden Fingerprints abweicht.

3. Das aus einer festen Kameraposition „gescannte“ Suchobjekt wird trotz beliebiger Translation und Rotation bei gleichem Blickwinkel zu 100% wiedererkannt.

4. Das aus einer festen Kameraposition „gescannte“ Objekt wird in Anwesenheit von Größenänderung bei gleichem Blickwinkel (wie in der Scannen-Phase) umso besser wiedererkannt je größer die Objektkontur des Such-

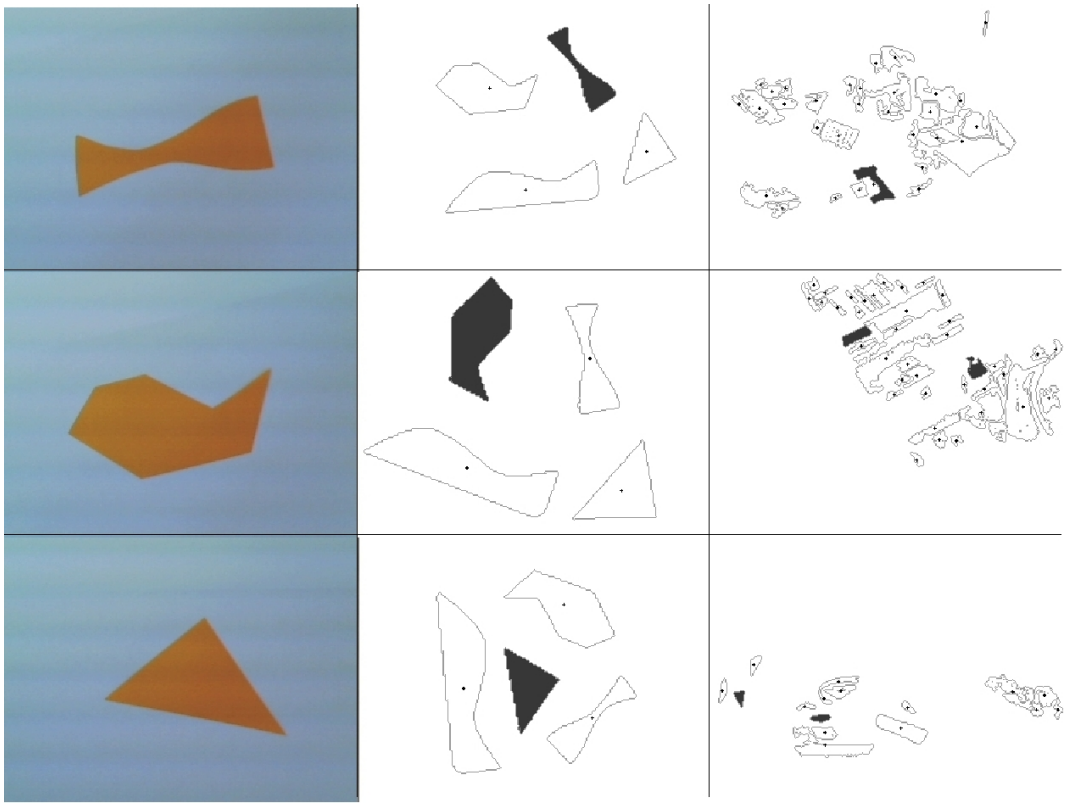


Abb. 22: *Richtig/Falsch positiv wiedererkannte Objekte mit variabler Kameraposition während der Scannen-Phase, jede Zeile repräsentiert einen Programmdurchlauf, die erste Spalte zeigt das Kamerabild des orangen Schablonen-Objektes auf weißem Hintergrund bei per Hand veränderter Kameraposition während der Scannen-Phase (ca. bis zu 5 Grad), die letzten Spalten zeigen das Kamerabild mit den wiedererkannten Objekten (schwarz ausgefüllter erkannter Objektkantenzug) und nicht wiedererkannten Objekten mit schwarzem Kantenzug und schwarzem Kreuz des Objektschwerpunktes während der Objektwiedererkennungs-Phase, Quelle: eigene Darstellung*

objektes während der Scannen-Phase im Kamerabild detektiert wurde (siehe Abbildung 20).

5. Das Suchobjekt mit nur einer Referenzkontur (Objekt wurde aus einer einzigen Kameraposition „gescannt“) wird in Anwesenheit von Verzerrungen (insbesondere perspektivische Verzerrungen bei Abweichung des Blickwinkels $> 1/320 \times 1/240$ vom horizontalen und vertikalen Öffnungswinkel der Kamera) kaum wiedererkannt.

6. Die Rate der falsch positiv wiedererkannten Objekte, ausgehend von einer Referenzkontur des Suchobjektes, im Innenraum der Versuchsanordnung ist 0% (siehe Abbildung 21).

7. Jedes dritte Bild von 15 pro Sekunde kann vollständig prozessiert werden, wenn maximal 50 Referenzkonturen (Suchobjekt wurde aus 50 Blickwinkeln „gescannt“) und ihre Fingerprints gespeichert wurden, die Prozessorlast liegt dann bei 80% im Debug-Modus.

5 Fazit

Es konnte ein farb- und formunabhängiger, schneller Algorithmus zum Erkennen und Suchen monotoner Flächen implementiert werden. Die Reaktionszeiten liegen bei deutlich unter einer Sekunde (im Schnitt bei 200ms) und ermöglichen viele Einsatzmöglichkeiten bei dynamischer Umwelt. Der Algorithmus setzt sich aus klassischen und intuitiven Methoden der Computer-Vision zusammen. Er zeigt gute Robustheit bei Translation, Rotation und Größenskalierung. Wenig bis beschränkt robust ist der Algorithmus bei Verzerrungen, da durch Verzerrung die einzige Information der Suchobjekte (ihre Kantenform) an Information verliert und sich damit die Falsch-Positiv-Rate erhöht. Aufgrund des SUSAN-Kantendetektor zeigt der Algorithmus sehr gute und konstante Resultate der Kantenfindung auf Grauwertbildern, selbst bei kleinster Maskengröße und in Abwesenheit von Vor- und Nachfilterung.
Es konnte somit ein Algorithmus für die farb- und formunabhängige und kamerabasierte Objektfindung erarbeitet werden, der sich an die zu suchende Objektform anpasst, ohne vordefinierte Vorgaben über das Aussehen des Objektes oder der Beschaffenheit der Umwelt zu benötigen.

5.1 Weitere Arbeiten

Leider arbeitet der verwendete Kantendetektor nur auf Graubildern, so dass durch diesen Informationsverlust Kontraste im ursprünglich dreidimensionalen Bild verloren gehen können. Deswegen wäre ein Kantendetektor der auf einem dreidimensionalen Farbraum operiert wünschenswerter. So könnte man einem Detektor, welcher beispielsweise Informationen über den euklidischen Abstand und den Winkel zweier Farbpunkte im Raum verwendet, nutzen.[8]
Wenn man die Beziehungen mehrerer gleichzeitig „gescannter“ Objektflächen mit in den Algorithmus einfließen ließe, kann man sogar ein dreidimensionales Objekt suchen. Dies würde voraussetzten, dass alle angrenzenden Flächen aus denen das Objekt zusammengesetzt ist, sich voneinander unterscheiden. Durch die Information mehrerer in Beziehung stehender Objektflächen, dürfte die Falsch-Positiv-Rate im Vergleich zu dieser Arbeit gesenkt werden, so denn auch mehrere Flächen des dreidimensionalen Objektes gleichzeitig detektiert werden können.

A Arbeitsumgebung

Im Rahmen dieser Arbeit wurde aus dem Arbeitsprojekt *HTH-2006* das C++ Framework *PCCognition* verwendet. Als Programmierumgebung diente *Microsoft Visual Studio 2005 Professional Edition Version 8* mit *Microsoft .NET Framework Version 2*. Innerhalb des Frameworks gelten die folgenden Konventionen für die Bildverarbeitung:

- Die Datenquelle ist ein Videostream mit einer Einzelbildrate von 15 Bilder pro Sekunde
- Das Farbspektrum des Einzelbildes liegt in RGB24 vor
- Die Ausgabegröße des Einzelbildes ist $320 \cdot 240$ Bildpunkte

Bei der in der Arbeit vorliegenden Kamera handelt es sich um die *Logitech QuickCam Pro 5000*. Um die Standard Geräte-Einstellungen zu ändern, kann das Programm *amcap8.exe* verwendet werden.

B Kamera-Einstellungen

Für Gütebestimmung bei Größenänderung der Suchobjekte abgebildet auf dem LCD-Monitor:

- Helligkeit: 5000 von 10000
- Kontrast: 10000 von 10000
- Sättigung: 0 von 10000
- Schärfe: 10000 von 10000
- Weißabgleich: 10000 von 10000
- Gegenlichtkompension: 2 von 2
- Belichtung 1/50 [s]
- Empfindlichkeit: 4000 von 10000

Für Messung der falsch positiv wiedererkannten Objekte im Innenraum bei durchschnittlicher Beleuchtung :

- Helligkeit: 5000 von 10000

- Kontrast: 1000 von 10000
- Sättigung: 1000 von 10000
- Schärfe: 2000 von 10000
- Weißabgleich: 0 von 10000
- Gegenlichtkompension: 1 von 2
- Belichtung 1/30 [s]
- Empfindlichkeit: 0 von 10000

Literatur

[1] Prof. Dr. Verena V. Hafner. Kognitive Robotik: Visuelle Verarbeitung. Humboldt Universität, 2008. Url: https://www2.informatik.hu-berlin.de/ki/lehre/ws0708/vlkogrob-skript/KR0708-Visuell2.pdf.

[2] D. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, volume 20, pages 91–110, 2003.

[3] J. Ogden, E. Adelson, J. Bergen, and P. Burt. PyramidBased Computer Graphics, 1985. Url: citeseer.ist.psu.edu/ogden85pyramidbased.html.

[4] Mubarak Shah. Fundamentals Of Computer Vision. Universität Zentral Florida, December 1997. Url: http://www.cs.ucf.edu/courses/cap6411/book.pdf.

[5] S. M. Smith and J. M. Brady. *SUSAN – A new approach to low level image processing.* Chertsey, Surrey, UK, 1995. Url: citeseer.ist.psu.edu/smith95susan.html.

[6] Stephen Smith. SUSAN Version 21. Universität Oxford, 1995-1999. Es handelt sich um den C-Code erhältlich unter: http://users.fmrib.ox.ac.uk/ steve/susan/susan2l.c.

[7] Paul Suetens, Pascal Fua, and Andrew J. Hanson. Computational strategies for object recognition. *ACM Comput. Surv.*, 24(1):5–62, 1992.

[8] Slawo Wesolkowski and Ed Jernigan. Color Edge Detection In RGB Using Jointly Euclidean Distance And Vector Angle. *Vision Interface*, 1999.