Studienarbeit
zur
Still Image Segmentation

Studienarbeit
von
Susanne Krabbe
aus
Greifswald

vorgelegt am
Lehrstuhl für Praktische Informatik IV
Prof. Dr. Effelsberg
Fakultät für Mathematik und Informatik
Universität Mannheim

Dezember 2000

Betreuerin : Dipl.-Math. oec. Claudia Schremmer

1. Motivation für die „Still Image Segmentation“

Sollen digitale Bilder gespeichert oder über ein Netzwerk versendet werden, so werden sie oft komprimiert, um die vorhandene Datenmenge zu reduzieren.

Dabei versuchen alle Kompressionsverfahren das Originalbild bei möglichst hoher Kompressionsrate in möglichst guter Qualität zu erhalten. Was „gute Qualität“ ist, hängt jedoch vom optischen System des Betrachters ab. Da das normalerweise der Mensch ist, wird man die Qualität von komprimierten Bildern entsprechend dem visuellen Wahrnehmungssystem des Menschen beurteilen.

Um ein objektives Qualitätsmaß zu erhalten, das die Wahrnehmung eines menschlichen Betrachters nachempfindet, muß man beachten, daß die gleiche Störung, wenn sie in verschiedenen Bereichen eines Bildes auftritt, von einem Menschen hinsichtlich der Qualität des Bildes ganz unterschiedlich bewertet wird.

Die „Still Image Segmentation“ zerlegt ein Bild in die drei Bereiche Kanten, Texturen und Hintergrund.

Kanten kann man als sprunghafte Grauwertänderungen beschreiben, die im Bild die Konturen darstellen und Objekte trennen. Texturen sind viele, zum Teil nach einem Grundmuster wiederkehrende Helligkeitswechsel, durch die eine Zusammengehörigkeit innerhalb eines Objektes gekennzeichnet wird. Der Hintergrund – auch Flächen genannt - setzt sich aus Bereichen zusammen, in denen sich die Grauwerte nur geringfügig ändern und kontinuierliche Übergänge zwischen den Grauwerten zu finden sind. Der Hintergrund kann auch einfach als der Teil des Bildes betrachtet werden, der übrig bleibt, wenn vom Originalbild die Kanten und Texturen abgezogen werden.

In dieser Studienarbeit soll die Segmentierung von Bildern dargestellt werden, indem verschiedene Kanten- und Texturalgorithmen in Java implementiert werden.

2. Bildverarbeitungsalgorithmen für die „Still Image Segmentation“

Digitale Rasterbilder bestehen aus einer Menge von diskreten Werten, den Pixeln. Da hier nur Grauwertbilder betrachtet werden, können die einzelnen Pixel nur einen Grauwert zwischen 0 und 255 annehmen, wobei 0 schwarz und 255 weiß entspricht.

2.1. Faltung

Viele Bildverarbeitungsalgorithmen arbeiten nach dem Prinzip der Faltung (*engl*. convolution). Sie schieben eine Filtermaske über die einzelnen Pixel des Bildes und berechnen den neuen Wert für das Pixel, das sich unter der Mitte der Maske befindet, indem jedes Pixel im Originalbild, das im Bereich der Filtermaske liegt, mit dem entsprechenden Gewicht aus der Filtermaske multipliziert wird. Die erhaltenen Produkte addiert man und weist die Summe dem Pixel in der Mitte zu. Die Algorithmen in dieser Studienarbeit verwenden quadratische Filtermasken für die Faltung.

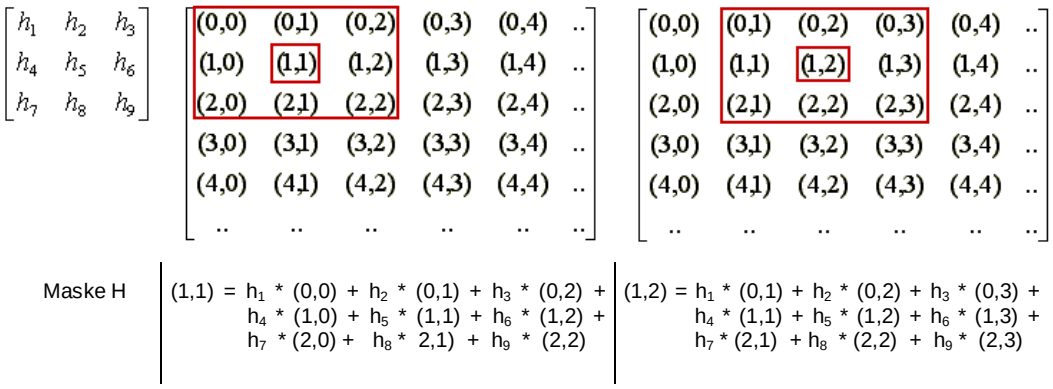


Abbildung 1: Faltung eines Bildes mit einer Maske H

2.2. Thresholding – Schwellwertbehandlung

Durch eine Schwellwertbehandlung kann man ein Bild in verschiedene Bereiche teilen. Dazu legt man einen Schwellwert oder auch Threshold (engl.) fest, mit dem man jedes Pixel des Bildes vergleicht. Allen Pixeln mit einem Grauwert größer gleich dem Threshold weist man den gleichen festen Grauwert zu und allen anderen Pixel – deren Grauwerte kleiner als der Schwellwert sein müssen – einen anderen.

Thresholding wird unter anderem bei der Kantenerkennung eingesetzt. Nachdem ein Kantenoperator auf ein Bild angewendet wurde, wird jedes Pixel

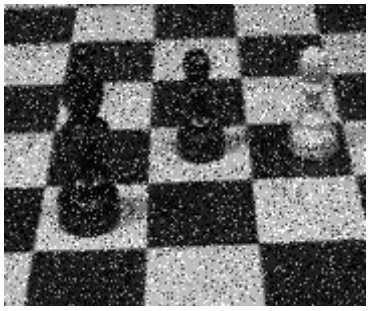
mit dem Threshold verglichen. Pixel mit Grauwerten größer gleich dem Schwellwert werden als Kanten gekennzeichnet, indem man ihnen die Farbe weiß gibt. Alle Pixel, die keine Kanten sind, also mit Grauwerten kleiner als dem Schwellwert, werden schwarz dargestellt.

Diese Art der Schwellwertbehandlung wird von fast allen Kantenalgorithmen verwendet. Der Canny-Kantendetektor benutzt ein anderes Thresholdingverfahren (siehe 2.4.8 Canny-Kantendetektor).

Auch die Texturalgorithmen verwenden Thresholding. Bei ihnen wird durch den Schwellwert ein Bild in Bereiche mit unterschiedlichen Texturen aufgeteilt.

2.3. Bildglättung

Viele Bilder enthalten Rauschen, das z. B. bei der Aufnahme der Bilder entsteht. Diese zufälligen Grauwertänderungen, die in keiner Beziehung zu den Grauwerten ihrer Nachbarpixel stehen, können ein Bild so stören, daß wichtige Details nicht mehr erkennbar sind. Um Rauschen zu unterdrücken, werden verschiedene Filter verwendet, z. B. der Median-Filter und der Gauß-Filter.



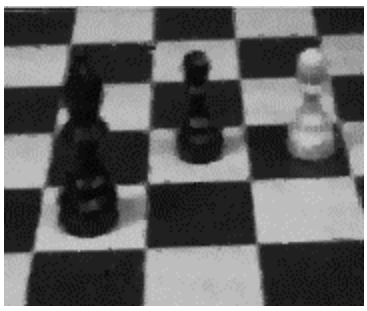


Abbildung 2:

(a) Bild mit Rauschen

(b) Bild nach Anwendung des Medianfilters

2.3.1. Gauß-Filter

Für die Durchführung der Glättung mittels der Gaußfunktion gibt es mehrere Wege. Man kann zum Beispiel das Bild mit einer Filtermaske falten, deren Werte direkt mit der Gaußfunktion berechnet wurden oder man benutzt eine Näherung.

Für die erste Variante braucht man die Gaußfunktion:

$$G(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\left(\frac{x^2}{2\sigma^2}\right)}$$

σ ist die Standardabweichung und bestimmt die Breite der Filtermaske – je größer σ ist, um so größer ist die Maske und um so stärker wird das Bild geglättet. Da die Gaußfunktion nie den Wert Null annimmt, würde die

Filtermaske unendlich groß werden. In der Praxis gewinnt man vernünftig große Masken, indem man die Gewichte auf Null setzt, wenn die Gaußfunktion einen bestimmten Wert unterschreitet. Die Koeffizientensumme der Maske muß noch auf Eins normiert werden, damit die Grauwertskala des Bildes nicht verändert wird, d.h., damit das Bild durch das Filtern nicht insgesamt dunkler oder heller wird. Die Glättung mit der Gauß-Funktion ist separabel, d.h., eine Glättung im Zweidimensionalen mit einem zweidimensionalen Gaußkern (d.h. einer Matrix) entspricht der Glättung eines Bildes mit einer eindimensionalen Filtermaske (d.h. einem Vektor) erst in x-Richtung und dann in y-Richtung. Das bedeutet, daß man mit dem oben berechneten Filter erst entlang den Spalten glättet und das Ergebnis dann entlang den Reihen bearbeitet. Wobei die Reihenfolge, ob erst Spalten oder Reihen genommen werden, egal ist.

Neben der direkten Berechnung kann man den Filter für die Gauß-Glättung auch durch Näherung erhalten, denn nach dem Grenzwertsatz von Moivre-Laplace approximiert die Binomialverteilung die Gauß-Funktion. Die Filterelemente entsprechen Binomialkoeffizienten und können daher aus dem Pascalschen Dreieck abgelesen werden.

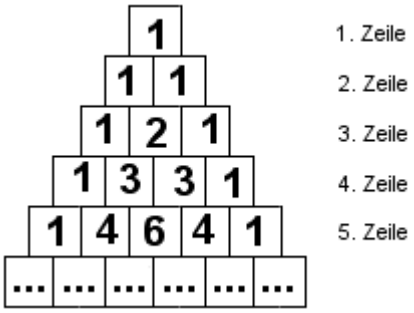


Abbildung 3: Ausschnitt aus dem Pascalschen Dreieck

Sucht man z. B. eine 3x3 Maske, benutzt man die Elemente in Zeile drei des Dreiecks. Der Filter besteht aus $\frac{1}{4}[1 \quad 2 \quad 1]$, das entspricht einer Standardabweichung von $\sigma = \sqrt{1/2}$. Der Faktor vor dem Vektor normiert die Summe der Vektorelemente auf Eins.

Für einen 5x5 Pixel großen Filter nimmt man die Elemente aus Zeile fünf und erhält $\frac{1}{16}[1 \quad 4 \quad 16 \quad 4 \quad 1]$, die zugehörige Standardabweichung ist $\sigma = 1$.

Aufgrund der Separabilität kann man diese beiden eindimensionalen Vektoren statt einer zweidimensionalen Maske verwenden, indem mit ihnen nacheinander horizontal und vertikal das Bild faltet.

2.3.2. Median-Filter

Beim Median-Filter wird das Bild nicht mit einem Filter gefaltet, sondern die n Pixel des Bildes, die unter der Filtermaske liegen, werden entsprechend ihrem Grauwert sortiert (der Median gehört zu den Rangordnungsfiltern):

$$\underset{Minimum}{Grauwert(0)} \le Grauwert(1) \le Grauwert(2) \le ... \le Grauwert(n-2) \le \underset{Maximum}{Grauwert(n-1)}$$

Das Ergebnis des Median-Filters ist der Grauwert, der sich bei ungeraden n an der Stelle $\frac{n+1}{2}$ befindet. Besteht die Filtermaske aus einer geraden Anzahl von Elementen, so ergibt sich der Median als das arithmetische Mittel der Grauwerte, die sich an den Positionen $\frac{n}{2}$ und $\frac{n}{2}+1$ befinden. Um diese Durchschnittsbildung zu vermeiden, verwendet man normalerweise eine Maske mit ungerader Anzahl von Elementen.

Für die Maske sind verschiedene Formen vorstellbar:

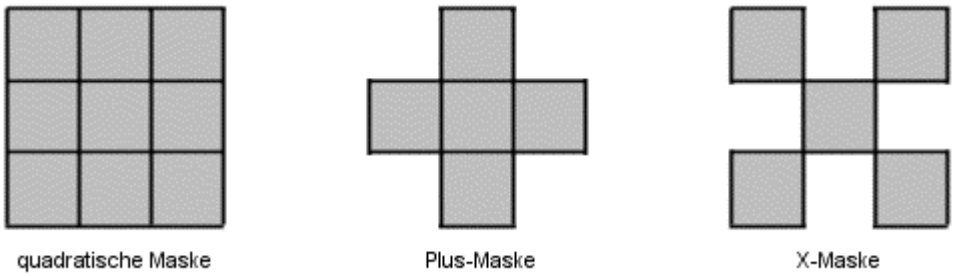


Abbildung 4: Formen für die Maske des Median-Filters

In dieser Implementation wird für die Bestimmung des Medians eine quadratische Maske benutzt, mit drei Pixeln oder fünf Pixeln als Seitenlänge.

2.4. Kantenerkennung

Kanten stellen die Umrandung von Objekten dar, sie grenzen Gegenstände von ihrem Hintergrund ab und zeigen die Grenze zwischen sich überlappenden Gegenständen in einem Bild.

Wenn man sich ein Bild als eine Funktion g(x, y) vorstellt, wobei jedem Pixelpaar (x, y) der sich dort befindende Grauwert zugeordnet ist, kann man Kanten als Sprung in g(x, y) definieren. Das bedeutet, daß eine Kante dort auftritt, wo sich Grauwerte abrupt ändern. Ein einfaches Beispiel ist ein Bild, das aus zwei vertikalen Streifen (Grauwert 20 und 200) besteht. Legt man horizontal einen Schnitt durch das Bild und betrachtet die Grauwerte in Abhängigkeit von der x-Richtung, erhält man folgende Darstellung.

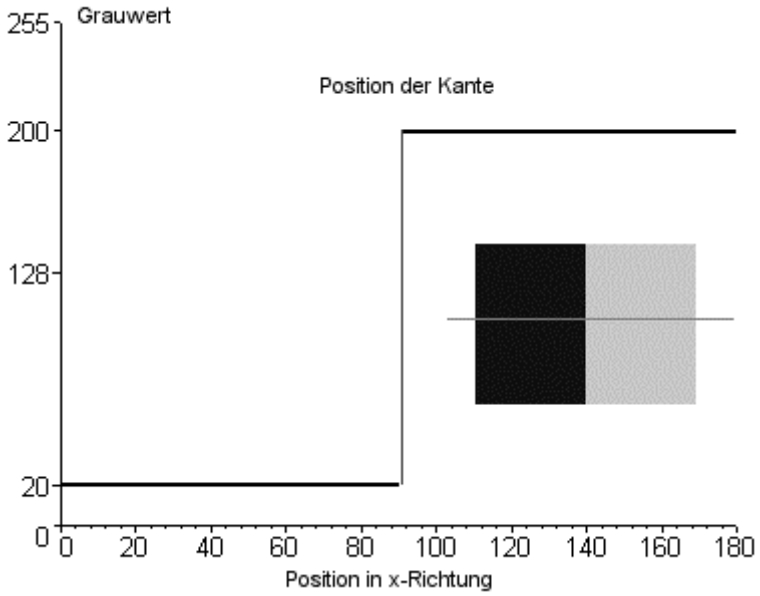


Abbildung 5: Beispiel für eine Kante[1]

Da Kanten starke Änderungen in der Grauwertfunktion eines Bildes darstellen, kann man Operatoren, die darauf reagieren, als Kantendetektoren verwenden. Ein Beispiel sind Ableitungsoperatoren.

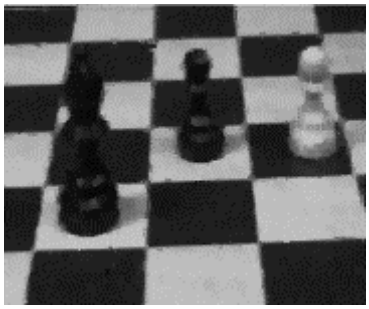
2.4.1. Ableitungsoperator – 1. Ableitung

Der Gradient grad g(x) = $\left(\frac{\partial g}{\partial x_1},...,\frac{\partial g}{\partial x_n}\right)$ ist definiert als Vektor der partiellen Ableitungen einer Funktion g(x) im n-dimensionalen Fall.

Für ein zweidimensionales Bild besteht der Gradient aus der partiellen Ableitung nach x und der partiellen Ableitung nach y, d.h. den Ableitungen in horizontaler und vertikaler Richtung. Wenn man für jedes Pixel die Ableitung in x- und y-Richtung bestimmt hat, erhält man den Ergebniswert für ein Pixel, indem man den Betrag des Gradienten in dem Punkt berechnet. Der Betrag des Gradienten ist als $|grad\ g(x,y)| := \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2}$ definiert.

Da ein Bild keine kontinuierliche, sondern eine diskrete Funktion ist, können die Ableitungen nur näherungsweise berechnet werden. Als Näherung für die erste Ableitung verwendet man die Filtermaske $[1 \quad 0 \quad -1]$, mit der das Bild in horizontaler und vertikaler Richtung gefaltet wird.

[1] nach Parker, James R.: Algorithms for image processing and computer vision, 1997, S. 4



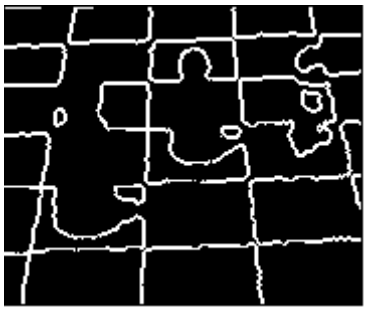


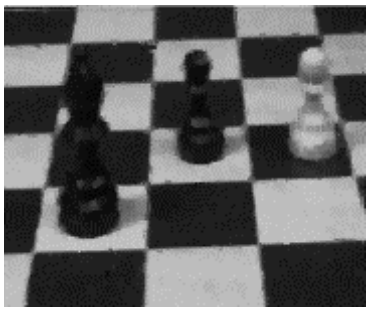
Abbildung 6:

(a) mit Median geglättetes Bild

(b) Kantenbild für den 1.Ableitungsoperator Threshold = 128

2.4.2. Ableitungsoperator – 2. Ableitung

Kanten werden mit der zweiten Ableitung nach dem gleichen Prinzip erkannt wie beim ersten Ableitungsoperator. Die Filtermaske ist $[-1 \quad 2 \quad -1]$. Sie wird wieder in x- und in y-Richtung auf das Originalbild angewendet.



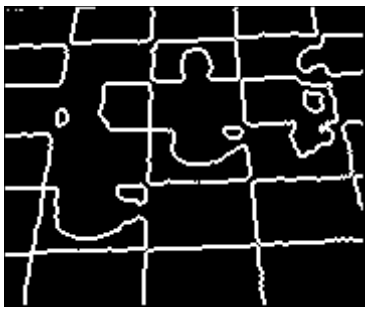


Abbildung 7:

(a) mit Median geglättetes Bild

(b) Kantenbild für den 2.Ableitungsoperator Threshold = 128

2.4.3. Roberts-Kantendetektor

Roberts-Kantendetektor benutzt die folgenden Differenzfilter, um den Gradienten zu berechnen:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Der Gradient wird in horizontaler Richtung bestimmt. Man erhält für jedes Pixel zwei Werte, aus denen man den Betrag des Gradienten berechnet. Diesen Wert weist man dem Pixel zu, der sich unter der linken oberen Ecke der Filtermaske befindet.

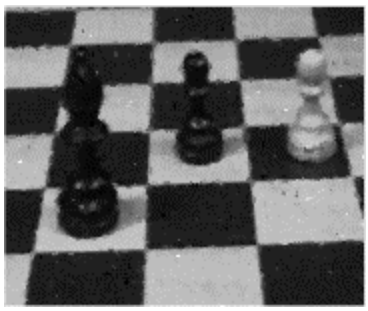


Abbildung 8:

(a) mit Median geglättetes Bild

(b) Kantenbild des Roberts-Operators
Threshold = 128

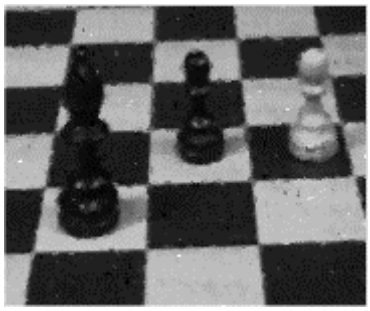
2.4.4. Prewitt-Kantendetektor

Die Filtermasken des Prewitt-Kantendetektors sind:

$$P_x = \frac{1}{3}\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}\begin{bmatrix}1 & 0 & -1\end{bmatrix} \qquad P_y = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\ 0\\ -1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

Die Filtermasken P_x und P_y kann man in zwei Vektoren zerlegen. P_x besteht aus einer Glättung in vertikaler Richtung und der ersten Ableitung in horizontaler Richtung, in P_y sind Ableitung und Glättung getauscht. Da Ableitungsoperatoren nicht nur Kanten erkennen, sondern auch Rauschen, das wie Kanten eine abrupte Grauwertänderung ist, wird das Bild geglättet und dadurch das Rauschen verringert. Die Glättung wird hier realisiert, in dem über drei Werte der Durchschnitt berechnet wird.

Für ein Pixel im Bild erhält man durch die Anwendung von P_x und P_y zwei Werte. Diese Werte kann man als Elemente des Gradientenvektors auffassen und aus ihnen dessen Betrag berechnen, der dann dem Pixel im Ergebnisbild zugewiesen wird.



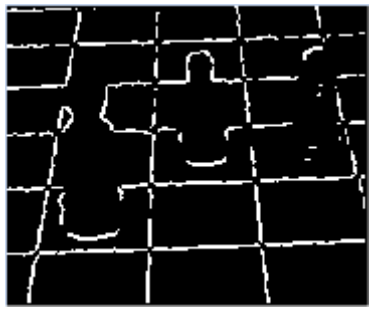


Abbildung 9:

(a) mit Median geglättetes Bild

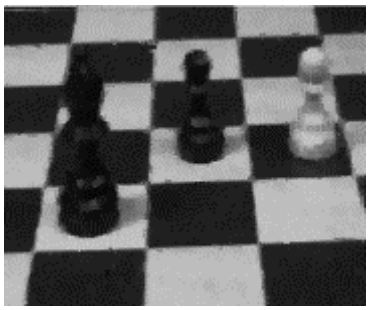
(b) Kantenbild des Prewitt-Operators
Threshold = 128

2.4.5. Sobel-Kantendetektor

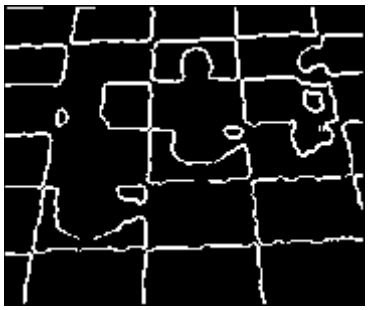
Der Sobel-Kantendetektor verwendet folgende Filtermasken:

$$S_x = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \qquad S_y = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Auch die Sobelfiltermasken kann man in zwei Vektoren zerlegen. So entspricht die horizontale Sobelfiltermaske S_x einer Glättung in y-Richtung und der ersten Ableitung in x-Richtung, für die vertikale Sobelmaske S_y sind Glättung und Ableitung vertauscht. Die Glättung wird verwendet, um im Bild enthaltenes Rauschen zu vermindern. Die Glättungsmaske ist eine Näherung des Gauß-Filters.



(a) mit Median geglättetes Bild



(b) Kantenbild des Sobel-Operators Threshold = 128

Abbildung 10:

2.4.6. Robinson-Kantendetektor

Dieser Kantendetektor faltet ein Bild mit den folgenden acht Filtermasken:

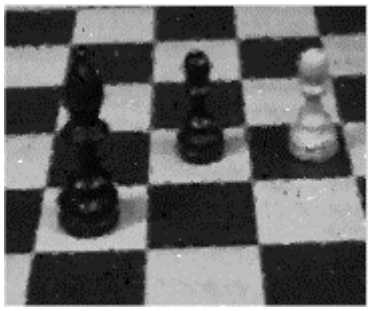
$$R_0 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad R_1 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad R_2 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad R_3 = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

$$R_4 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad R_5 = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \quad R_6 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad R_7 = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Man erhält für jedes Pixel im Bild acht verschiedene Werte. Aus diesen acht Werten kann man auf verschiedene Weise das Ergebnis bestimmen. Es kann das Maximum als Ergebnis gewählt werden oder das Mittel der acht Werte. Vor dem Mitteln kann man die Werte auch noch verschieden gewichten.

In dieser Implementation wird der größte Wert als Ergebnis benutzt.
Jede der acht Robinsonmasken modelliert eine von acht Kantenrichtungen. Mit welcher Matrix welche Kantenrichtung versucht wird nachzubilden, ist in der folgenden Tabelle zusammengefaßt:

Filtermaske	Helligkeitszunahme	Kantenrichtung
R_0	von links nach rechts	270°
R_1	von links unten nach rechts oben	] 270°, 360° [
R_2	von rechts nach links	90°
R_3	von rechts oben nach links unten	] 90°, 180° [
R_4	von oben nach unten	180°
R_5	von rechts unten nach links oben	] 0°, 90° [
R_6	von unten nach oben	0°
R_7	von links oben nach rechts unten	] 180°, 270° [



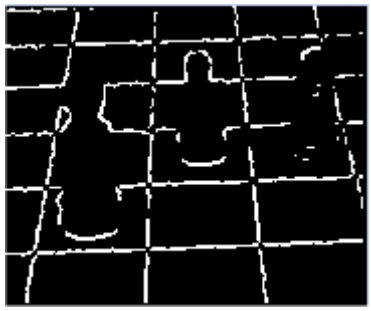


Abbildung 11:

(a) mit Median geglättetes Bild

(b) Kantenbild des Robinson-Operators
Threshold = 128

2.4.7. Kirsch-Kantendetektor

Der Kirsch-Kantendetektor versucht die Art der Grauwertänderungen, wie sie sich in der Nähe von Kanten mit unterschiedlichen Ausrichtungen finden, nachzubilden.
Die Masken des Kirschoperators sind:

$$K_0 = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad K_1 = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad K_2 = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad K_3 = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

$$K_4 = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad K_5 = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad K_6 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad K_7 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

Jede dieser Masken stellt eine von acht Kantenrichtungen dar. Welche Matrix welche Kantenrichtung nachbildet, ist in der folgenden Tabelle zu sehen:

Filtermaske	Helligkeitszunahme	Kantenrichtung
K_0	von links nach rechts	270°
K_1	von links unten nach rechts oben	] 270°, 360° [
K_2	von unten nach oben	0°
K_3	von rechts unten nach links oben	] 0°, 90° [
K_4	von rechts nach links	90°
K_5	von rechts oben nach links unten	] 90°, 180° [
K_6	von oben nach unten	180°
K_7	von links oben nach rechts unten	] 180°, 270° [

Für die Kantenerkennung wird ein Bild für jedes Pixel mit allen acht Filtermasken gefaltet. Man erhält acht Zwischenbilder. Aus diesen acht Bildern wird für jede Pixelposition das Maximum bestimmt. Das Zwischenbild, aus dem das Maximum stammt, bestimmt die Hauptrichtung der Kante. Befindet sich der größte Wert zum Beispiel im Bild, das durch Faltung mit der Maske K_0 entstanden ist, impliziert das eine vertikale Kante. Als Ergebnis wird jedem Pixel die maximale Filterantwort aller Masken zugewiesen.

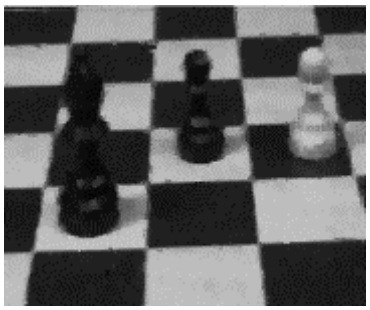


Abbildung 12:

(a) mit Median geglättetes Bild

(b) Kantenbild des Kirsch-Operators Threshold = 128

2.4.8. Canny-Kantendetektor

Der Canny-Kantendetektor kombiniert eine Glättung durch Gaußfunktion mit Kantenerkennung.

Das Bild wird als erstes mit einem Gauß-Filter geglättet. Die Stärke der Glättung hängt von der Standardabweichung σ ab, die ein Eingabeparameter des Canny-Kantendetektor ist. Für die Glättung wird ein eindimensionaler Filter verwendet, mit dem das Originalbild einmal entlang den Spalten und entlang den Reihen geglättet wird, so das man als Zwischenergebnis zwei Bilder – einmal das Original geglättet in x-Richtung und einmal geglättet in y-Richtung - erhält.

Für die Kantenerkennung wird die erste Ableitung der Gaußfunktion verwendet:

$$G(x)=e^{-\frac{x^2}{2\sigma^2}} \qquad G'(x)=\left(-\frac{x}{\sigma^2}\right)e^{-\left(\frac{x^2}{2\sigma^2}\right)}$$

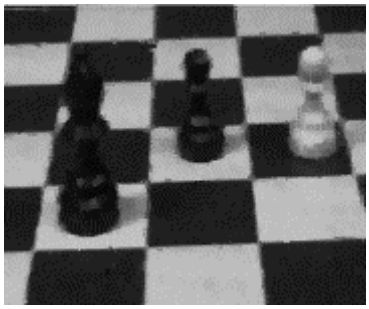
Abbildung 13: Gaußfunktion und erste Ableitung

Aus der ersten Ableitung wird eine Filtermaske berechnet, mit der dann die beiden Zwischenbilder behandelt werden. Man erhält für jedes Pixel zwei Werte, die man als Vektor auffassen kann. Für jeden Vektor berechnet man seinen Betrag und weist diesen dem Zielpixel zu. Die Werte im Betragsbild sind für Kantenpixel groß und kleiner, falls es sich nicht um Kanten handelt.

Im nächsten Schritt, der *nonmaximum suppression*, werden alle Pixel, deren Gradientenbeträge keine lokalen Maxima sind, „entfernt“ (auf Null gesetzt), denn der Betrag des Gradienten sollte für ein Kantenpixel größer sein als für die Pixel, die neben der Kante liegen. Damit werden alle Kanten auf ein Pixel verdünnt. Man betrachtet jedes Pixel und nimmt für es an, daß durch es eine Kante läuft. Da der Gradient senkrecht auf einer Kante steht, vergleicht man für dieses Pixel den Betrag des Gradienten mit den Beträgen der Pixel, die sich in Richtung des Gradienten befinden. Wenn es sich wirklich um ein Kantenpixel handeln sollte, müßten die Pixel in Richtung des Gradienten kleinere Beträge haben. Haben sie das nicht, kann das betrachtete Pixel nicht zu einer Kante gehören und wird als Nichtkantenpixel markiert.

Als letztes muß das Bild mit einem Threshold behandelt werden, um zu entscheiden, welche Bildpunkte als Kanten angezeigt werden. Beim Canny-Algorithmus wird dafür *Hysteresis-Thresholding* verwendet. Dafür braucht man zwei Thresholds, einen oberen (T_{high}) und einen unteren (T_{low}) (Parameter für den Canny-Algorithmus). Jedes Pixel im Bild mit einem Wert größer als T_{high} wird als Kantenpixel gekennzeichnet. Dann werden alle Pixel, die von diesem

Kantenpixel aus erreichbar sind und einen Wert größer als T_{low} haben, ebenfalls als Kantenpixel markiert. Das Markieren aller Kantenpixel kann rekursiv geschehen oder indem man in mehreren Durchgängen das Bild untersucht.



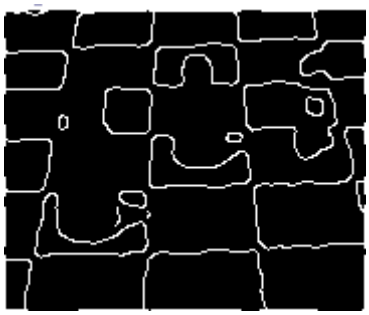


Abbildung 14:

(a) mit Median geglättetes Bild (b) Kantenbild des Canny-Operators

2.5. Texturerkennung

Texturen beschreiben das wiederholte Auftreten eines bestimmten Musters oder mehrerer Muster in einem Bildbereich.

Hier werden Texturen betrachtet, nicht um die Texturen selbst zu charakterisieren, sondern um damit ein Bild zu zerlegen. Das heißt, man versucht das Bild in Regionen mit unterschiedlichen Texturen zu teilen.

2.5.1. Co-Okkurenz-Matrizen

Grey level co-occurence Matrizen (GLCM) enthalten Informationen über die Position von Pixel, die ähnliche Grauwerte haben. Die Idee ist, das Bild abzusuchen und dabei zu zählen, wie viele Pixel, deren Grauwerte sich um Δz unterscheiden, einen festen Abstand *d* zueinander haben.

Für jede der vier Richtungen (horizontal, vertikal und beide Diagonalen) wird eine GLCM aufgestellt, d. h., einmal werden die Pixel gezählt, die in x-Richtung einen Abstand von d haben (horizontal), für die Vertikale wird der Abstand d in y-Richtung betrachtet, etc.

Für ein Grauwertbild ist die GLCM eine Matrix mit 256 Zeilen und 256 Spalten.

Sei I ein Bild mit 256 unterschiedlichen Grauwerten (0...255), der Abstand *d*=2 und das Bild werde in horizontaler Richtung betrachtet und sei MO die dazu berechnete *Grey level co-occurence* Matrix. Dann bezeichnet MO[i][j] die Anzahl der Pixel P1 und P2 in I, für die gilt, daß Grauwert(P1) = i und Grauwert(P2) = j ist und P2 den Abstand d zu P1 hat, d.h., die Grauwerte der beiden Pixel sind die Indizes der GLCM. Die Einträge in der GLCM können normalisiert werden, indem durch die Gesamtpixelanzahl geteilt wird, man erhält relative Wahrscheinlichkeiten. Die GLCM ist symmetrisch.

Wenn Bereiche in Bildern gleiche Co-Okkurenz-Matrizen haben, kann man annehmen, daß sie die gleiche Textur besitzen. Normalerweise werden aber nicht die Matrizen direkt benutzt, sondern aus ihnen statistische Kennzahlen berechnet und diese für die Charakterisierung von Texturen verwendet.

Da die Co-Okkurenz-Matrizen aber sehr rechenaufwendig sind, benutzt man als Näherung Summen- und Differenzhistogramme.

2.5.2. Summen- und Differenzhistogramme

Ein Summenhistogramm ist das Histogramm der Summen der Grauwerte aller Pixel, die sich in einem bestimmten Abstand in horizontaler, vertikaler oder diagonaler Richtung befinden, d. h., man addiert zum Grauwert des Pixel an der Position (x, y) den Grauwert des Pixels, der sich bei $(x + d_x, y + d_y)$ befindet. Diese Summe ist dann der Index für das Summenhistogramm und gibt das Element an, das inkrementiert werden muß. Für ein Bild mit 256 Grauwerten hat das Summenhistogramm einen Indexbereich von 0 bis 510, denn die Summe der kleinsten möglichen Grauwerte ist 0 + 0 = 0 und der größten Grauwerte 255 + 255 = 510.

Der Indexbereich des Differenzdiagramms geht von –255 (für 0 – 255) bis +255 (255 – 0). Im Differenzdiagramm werden die Grauwerte zweier Pixel, die einen festen Abstand voneinander haben, subtrahiert und das entsprechende Histogrammelement wird um eins erhöht, d.h., hier wird vom Grauwert des Pixels (x, y) der Grauwert des Pixels $(x + d_x, y + d_y)$ abgezogen.

Sowohl die Elemente der Summen- als auch der Differenzdiagramme normalisiert man, um aus den absoluten Häufigkeiten relative zu erhalten.

Die Summen- und Differenzhistogramme werden wie die *Grey level co-occurence* Matrizen für alle vier Richtungen berechnet (vertikal, horizontal und beide Diagonalen).

Aus den Histogrammen berechnet man, um die Textur zu beschreiben, einige statistische Größen. Dazu wird über das Bild ein quadratisches Fenster geschoben und aus allen Pixeln, die sich unter diesem Fenster befinden, die beiden Histogramme berechnet. Mit Hilfe der Histogramme kann man die gesuchten Kennzahlen berechnen, die entsprechend skaliert, dem Pixel in der Mitte des Fensters zugewiesen werden.

D sei im folgenden das Differenzdiagramm und S das Summendiagramm.

Erwartungswert (Mean)

Der Erwartungswert ist die Summe der mit ihren relativen Häufigkeiten gewichteten Summen der Grauwerte im Fenster.

$$\mu = \frac{1}{2}\sum_i i \cdot S(i)$$

Standardabweichung (Standard deviation)

$$\sqrt{\sum_i S(i) \cdot (i-\mu)^2}$$

Maximale Wahrscheinlichkeit (Maximum Probability)

Das ist der größte Eintrag im Summenhistogramm. Er gibt an, welcher Grauwertübergang am häufigsten im Bild auftritt.

Kontrast (Contrast)

$$\sum_j j^2 \cdot D(j)$$

Homogenität (Homogeneity)

$$\sum_j \frac{1}{1+j^2} \cdot D(j)$$

Entropie (Entropy)

$$-\sum_i S(i) \cdot \log(S(i)) - \sum_j D(j) \cdot \log(D(j))$$

Energie (Energy)

$$\sum_i S(i)^2 \cdot \sum_j D(j)^2$$

2.5.3. Law's Texture Energy Measures

Der folgende Ansatz zur Texturerkennung geht auf Laws (1980) zurück. Laws stellte ein paar Faltungsmatrizen bereit, speziell um die Energie in einer Textur zu berechnen. Die Matrizen entstehen durch Multiplikation der folgenden drei Vektoren mit sich selbst und den jeweils anderen beiden:

$E5 = (-1, -2, 0, 2, 1)$
$L5 = (1, 4, 6, 4, 1)$
$R5 = (1, -4, 6, -4, 1).$

L5, E5 und R5 sind durch Faltung dieser drei Vektoren entstanden:

$L3=(1,2,1)$

$E3=(-1,0,1)$

$S3=(-1,2,-1)$.

Wobei man L3 als eine Maske zur Durchschnittsbildung, E3 als erste Ableitung und S3 als zweite Ableitung interpretieren kann.

Man faltet das Bild mit einer der Masken, die man aus den fünfstelligen Vektoren erhält. Und berechnet dann, indem man ein Fenster über das Bild bewegt, die Energie, die sich für dieses Fenster ergibt. Die Energie ist hier definiert als:

$$E = \sum_x \sum_y |C(x,y)|,$$

C ist das mit einer der Masken gefaltete Bild.

2.5.4. Fractal Dimension

Fraktale sind Objekte, die einen bestimmten Grad von Selbstähnlichkeit aufweisen. Eine Gerade zum Beispiel ist sich in hohem Maße selbstähnlich. Man kann ein Segment dieser Geraden in vier gleich lange Abschnitte zerteilen und jeder dieser Abschnitte ergibt, wenn man ihn um den Faktor vier vergrößert, wieder das Originalstück der Geraden.

Im allgemeinen kann man ein Geradenstück in N selbstähnliche Abschnitte teilen, die mit einem Vergrößerungsfaktor N wieder das Original darstellen. Ein Quadrat müßte man in N^2 kleine selbstähnliche Quadrate teilen, damit man den Vergrößerungsfaktor N anwenden kann, um das Originalquadrat zu erhalten. Einen Würfel zerlegt man in N^3 selbstähnliche Würfel, die mit N auf Originalgröße gebracht werden können. *Fractal Dimension* ist in diesem Zusammenhang der Exponent der Zahl selbstähnlicher Stücke mit Vergrößerungsfaktor N, in die eine Figur zerlegt werden kann. Für die Gerade ist die *Fractal Dimension* also 1, für das Quadrat 2 und für den Würfel 3.

Die *Fractal Dimension* eines selbstähnlichen Gegenstands kann daher folgendermaßen definiert werden:

$$fractal\ dimension = \frac{\log(Anzahl\ der\ selbstähnlichen\ Objekte)}{\log(Vergrößerungsfaktor)}.$$

Wenn man log(Anzahl der selbstähnlichen Objekte) in Abhängigkeit von log(Vergrößerungsfaktor) zeichnet, erhält man eine Gerade, deren Steigung die *Fractal Dimension* sein sollte.

Da man Texturen als Muster definieren kann, die wiederholt und in unterschiedlichen Größen in einem Bild auftauchen, ist es möglich, mit *Fractal*

Dimension Texturen zu charakterisieren. Weil in „natürlichen“ Bildern Texturen aber selten aus einfachen Grundformen bestehen, kann man die *Fractal Dimension* für sie nur annähern, z. B. mit dem Hurst-Koeffizienten. Dazu betrachtet man einen 7x7 Pixel großen Bereich in einem Bild. In diesem Bereich werden die Pixel entsprechend ihres Abstands vom Pixel in der Mitte gekennzeichnet. Da es acht unterschiedliche Abstände gibt, gibt es auch acht verschieden gekennzeichnete Pixelgruppen. Innerhalb jeder Gruppe wird die größte Differenz zwischen den Grauwerten gesucht. Dann versucht man, die Beziehung der maximalen Differenzen zu den Abständen mit einer Geraden anzunähern. Die Steigung dieser Geraden ist der Hurst-Koeffizient und ersetzt das Pixel in der Mitte der 7x7 Pixelregion.
Als Beispiel[2] betrachte man den folgenden 7x7 Pixel großen Ausschnitt eines Bildes:

85	70	86	92	60	102	202
91	81	98	113	86	119	189
96	86	102	107	74	107	194
101	91	113	107	83	118	198
99	68	107	107	76	107	194
107	94	93	115	83	115	198
94	98	98	107	81	115	194

und die folgende Abbildung, in der die Pixel nach ihrem Abstand unterschiedlich markiert sind.

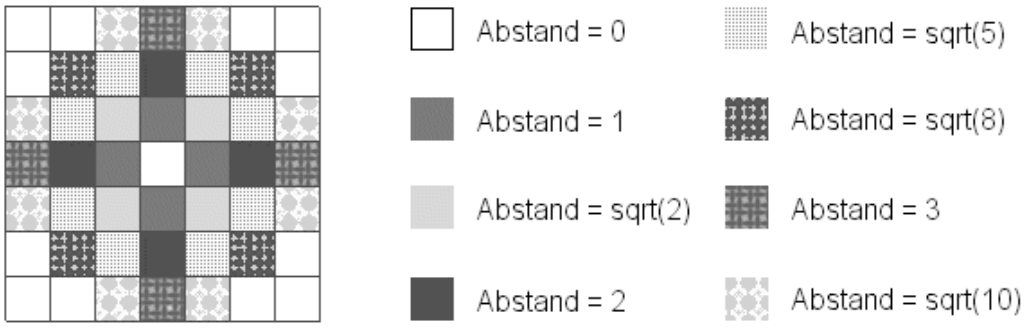


Abbildung 15: 7x7 Pixel großer Bildausschnitt

Als erstes muß man für jede Pixelgruppe die maximale Grauwertdifferenz berechnen. Untersucht man die Pixel mit Abstand 1 vom mittleren Pixel, hat man als größten Grauwert 113 und als kleinsten 83, d. h., die maximale Differenz ist 30. In der Pixelgruppe mit Abstand $\sqrt{2}$ beträgt die maximale

Differenz 39 = 113 – 74 und für Abstand 2 ist sie 44 = 118 – 74. Die Grauwertdifferenz wird also nicht mit dem Maximum und Minimum einer Pixelgruppe berechnet, sondern Maximum und Minimum können auch aus anderen Gruppen stammen, d.h., sie sind der bis dato größte beziehungsweise kleinste Grauwert.
Die folgende Tabelle führt alle berechneten Daten für das Beispiel auf.

Abstand	1	$\sqrt{2}$	2	$\sqrt{5}$	$\sqrt{8}$	3	$\sqrt{10}$
maximale Grauwertdifferenz	30	39	44	50	51	130	138
Ln(Abstand)	0,000	0,347	0,693	0,805	1,040	1,099	1,151
Ln(maximale Grauwertdifferenz)	3,401	3,664	3,784	3,912	3,932	4,868	4,927

Da es sich bei der *Fractal Dimension* um eine log-log-Beziehung handelt, müssen vom Abstand und den Grauwertdifferenzen die logarithmischen Werte berechnet werden. Dann nähert man diesen Werten mit Hilfe der Methode der kleinsten Quadrate eine Gerade an. In diesem Fall hat sie die Gleichung:

$$y = 1{,}145\,x + 3{,}229\,.$$

Die Steigung der Geraden, a = 1,145, ist der Hurst-Koeffizient und wird dem Pixel in der Mitte des 7x7 Pixel großen Bereichs zugewiesen.

2.6. Berechnung des Hintergrundes

Der Hintergrund ist die Differenz aus dem Originalbild und Kanten- und Texturbild, d.h., als Hintergrund wird all das angesehen, was weder Kante noch Textur ist. Da sowohl die Kanten- als auch die Texturbilder mit einem Threshold behandelt werden, kann man alles, was in ihnen weiß ist (zum Beispiel die Kanten), im Hintergrundbild schwarz darstellen.

[2] Beispiel vgl. Parker, James R.: Algorithms for image processing and computer vision, 1997, S. 172

3. Implementation

3.1. Implementation

Im Sinne von Java ist ein Applet eine Javaanwendung, die in eine Html-Seite eingebunden wird, in einem Browser ausgeführt wird und von der Klasse JApplet abgeleitet ist. In dieser Implementation gibt es auch so ein Applet, das aber nur beschränkte Funktionalität aufweist. Es stellt nur einen Button bereit, mit dem man einen Frame aufrufen kann, d. h. ein Fenster, welches eine Subklasse von JFrame ist und die eigentliche Funktionalität für die Segmentierung bereitstellt. Normalerweise werden Subklassen von JFrame verwendet um eigenständige Anwendungen zu entwickeln, d. h., Programme, die nicht innerhalb eines Browsers ausgeführt werden. Da hier der Frame aber von einem Applet aus gestartet wird und somit ein Teil des Applets ist, wird im folgenden auch für ihn allein die Bezeichnung Applet verwendet.

3.1.1. Entwicklungsumgebung

Das Applet wurde mit dem JDK1.2 und dem JDK1.3 auf einem PC unter Windows 98 entwickelt und mit dem Java™ Plug-in 1.3 für den Internet Explorer 5.0 und den Netscape Navigator 4.7 getestet.
Die Benutzeroberfläche wurde mit Hilfe der Swing-Klassen erstellt. Damit das Applet möglichst vielen auch nicht deutschsprachigen Benutzern verständlich ist, wurde für Bezeichnungen in der Benutzeroberfläche die englische Sprache gewählt.

3.1.2. Implementation der Algorithmen

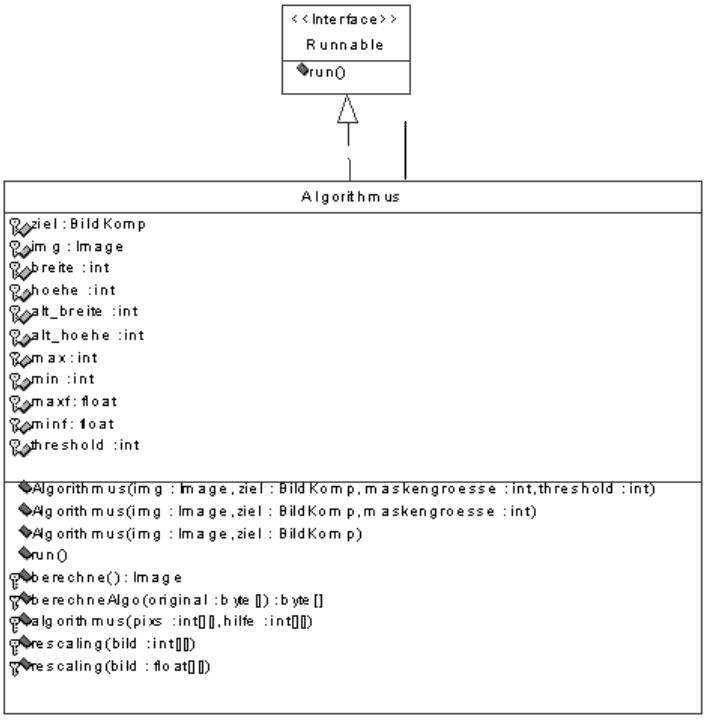


Abbildung 16: die Superklasse Algorithmus

Es soll hier nicht auf die Erstellung der Oberfläche in Java eingegangen werden, sondern nur die Implementierung der Bildverarbeitungsalgorithmen besprochen werden.

Da die meisten Bildverarbeitungsalgorithmen nach dem gleichen Prinzip arbeiten, z. B. benutzen alle nur ein Eingabebild und erstellen daraus nur ein Ausgabebild und für fast alle müssen die gleichen Arbeitsschritte vorgenommen werden, um z.B. Daten aus den Eingabebildern zu holen und in die Ausgabebilder zu schreiben, liegt es nahe, für die Bildverarbeitungsalgorithmen eine Superklasse anzulegen. Diese Klasse sammelt dann die allen Algorithmen gleichen Eigenschaften und stellt die Funktionalität bereit, die von allen Algorithmen gebraucht wird.

In diesem Fall übernimmt diese Aufgabe die Klasse Algorithmus. Die Attribute und Operationen dieser Klasse sind als protected definiert, damit ihre Subklassen die Attribute und Operationen erben. Die Klasse Algorithmus enthält als Attribut das BildKomp-Objekt ziel. Die Klasse BildKomp wird benutzt, um Bilder darzustellen. Sie benutzt dazu die Fähigkeit von JLabel, Bilder als Icons anzuzeigen. Im gesamten Applet gibt es fünf BildKomp-Objekte, eins für das Originalbild, eins, um das geglättete Bild anzuzeigen und jeweils eins für das Kanten-, Textur- und Hintergrundbild. In Algorithmus wird mit ziel das BildKomp-Objekt bezeichnet, in dem das Ergebnisbild angezeigt werden soll. Das Image-

Objekt img enthält das Eingabebild. In den beiden Attributen breite und hoehe wird die Größe des Bildes gespeichert. In alt_breite und alt_hoehe stehen die Ausmaße des Bildes, nachdem es vergrößert wurde (siehe 3.1.3 Randbehandlung), um eine problemlose Anwendung von Filtermasken zu gewährleisten. Die Attribute max, min, maxf und minf werden für die Funktionen rescaling gebraucht. Mit diesen Parametern wird der Wertebereich des Bildes angegeben. Im Attribut threshold ist der Schwellwert gespeichert, mit dem später das Bild gethresholdet werden soll.

Die beiden Funktionen rescaling nehmen eine lineare Transformation vor – einmal für eine float-Matrix und einmal für eine int-Matrix. Sie bilden die Grauwerte des Bildes aus dem Bereich [min, max] bzw. [minf, maxf] auf [0, 255] ab.

Die Funktion berechne holt die Daten aus dem Originalbild img und sorgt dafür, daß aus den berechneten Daten wieder ein Bild wird, das als Rückgabewert zurückgegeben wird. berechne ruft die Funktion berechneAlgo auf.

Die Operation berechneAlgo wandelt die Daten, die sie von berechne erhält, in handlichere Formate um (z.B. Vektoren in Matrizen, mit Matrizen läßt sich intuitiver umgehen, da die beiden Matrizenindizes der Breite und Höhe des Bildes entsprechen), startet die eigentliche Berechnung, reskaliert das Ergebnis und läßt dieses Ergebnis thresholden.

Die Methode algorithmus wird dazu verwendet, die eigentliche Logik des Bildverarbeitungsalgorithmus zu implementieren. Hier werden z.B. die Faltungen mit den Filtermasken durchgeführt.

Die Klasse Algorithmus implementiert das Runnable-Interface. Damit ist es möglich die einzelnen Algorithmen als eigene Threads laufen zu lassen. Dazu werden die Algorithmusobjekte der Funktion SwingUtilities.invokeLater als Parameter übergeben. Da in Swing Updates nicht mehr synchronisiert werden, wird diese Funktion verwendet, wenn man Swing-Komponenten von Threads updaten lassen möchte. Die Methode SwingUtilities.invokeLater übergibt die run-Methode der Algorithmen an den Ereignis-Dispatcher. Dieser führt run asynchron aus. In der run-Methode der Klasse Algorithmus wird die Funktion berechne aufgerufen und das Bild, das diese Funktion liefert, der BildKomp ziel zugewiesen. ziel zeigt dann ein geändertes Bild an.

Anmerkung:

Da Java-Bytecode interpretiert wird, sind Java-Programme im Gegensatz zu z.B. C-Programmen sehr viel langsamer. Das macht sich bei vielen Algorithmen

– z.B. den Texturalgorithmen – bemerkbar durch längere Wartezeiten, bis die GUI wieder reagiert.

3.1.3. Randbehandlung

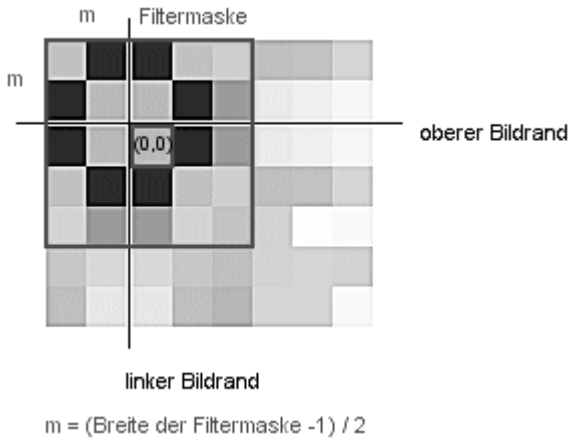


Abbildung 17: Kopieren von m Pixelreihen am linken und oberen Bildrand

Viele Bildverarbeitungsalgorithmen arbeiten mit Faltungsmasken (es werden nur quadratische, ungerade große Masken betrachtet), bei denen für das Pixel, das unter der Mitte der Maske liegt, ein Wert berechnet wird. Dadurch ist das Ergebnisbild der Faltung kleiner als das Original, denn für die Randpixel kann man keine Berechnung durchführen. Wenn sie im Zentrum der Faltungsmaske liegen, ist diese nämlich nicht vollständig ausgefüllt. Für das Pixel (0,0) in der oberen Abbildung, über das eine 5x5 Pixel große Maske gelegt wurde, ist z.B. der linke und der obere Teil der Maske leer.

Um diese leeren Bereiche der Faltungsmasken zu füllen, gibt es verschiedene Strategien. Man könnte sie z. B. mit Nullen auffüllen oder aber, so wie es hier implementiert ist, den Rand des Bildes jeweils außen an das Bild gespiegelt kopieren. Wieviele Pixel zu kopieren sind, hängt von der Breite der Maske ab. Ist sie fünf Pixel breit, müssen am Rand zwei Pixel hinzugefügt werden, denn das Pixel in der Mitte der Maske braucht zwei linke und zwei rechte Nachbarn. Verallgemeinernd kann gesagt werden, daß an jeder Seite des Bildes m = (Maskenbreite – 1) / 2 Pixelreihen kopiert werden müssen, damit auch für die äußeren Pixel die Masken gefüllt sind.

3.1.4. Threshold-Werte für Canny

Der Canny-Algorithmus benutzt Hysteresis-Thresholding. Dafür werden zwei Schwellwerte (T_{high} und T_{low}) gebraucht. In der Beschreibung des Algorithmus ist nur vorgesehen, daß der untere Schwellwert kleiner ist als der obere. Da im Applet aber der Benutzer diese beiden Werte bestimmen kann, muß man bei

der Implementation beachten, daß auch andere Kombinationen der Schwellwerte auftreten können.

Wenn der untere Threshold T_{low} kleiner als der obere Threshold T_{high} ist, dann wird das normale Hysteresis-Thresholding angewandt. Es werden erst alle Pixel als Kanten markiert, die einen Wert größer gleich T_{high} haben. Dann werden alle Pixel, die man von diesen schon markierten Pixel erreichen kann und die einen Wert größer gleich T_{low} haben, ebenfalls als Kantenpixel gekennzeichnet.

Wenn $T_{low} > T_{high}$ ist, werden zwei neue Schwellwerte berechnet. Sie werden berechnet, indem das Histogramm des Bildes ausgewertet wird, welches die Beträge aus der Faltung mit der ersten Ableitung der Gaußfunktion enthält. Das heißt, es wird das Histogramm des Bildes mit den potentiellen Kanten ausgewertet. Man sucht den oberen Schwellwert, indem man festlegt, daß ein bestimmter Prozentsatz p an Pixel im Bild zu den Kanten gehören soll. Hier wurde p = 90% - (Anteil schwarzer Pixel am Bild) angenommen. Der Anteil der schwarzen Pixel wurde abgezogen, da sonst die Thresholds oft bei Null liegen, denn im Gradientenbetragsbild sind hauptsächlich schwarze Pixel und im Verhältnis dazu wenig helle Pixel vorhanden. Mit Hilfe dieses Prozentsatzes p und dem Histogramm bestimmt man dann den Grauwert, für den gilt, daß p% Pixel einen größeren Grauwert haben. T_{high} wird auf diesen Grauwert gesetzt. T_{low} ist T_{high} geteilt durch zwei.

Gilt für die Schwellwerte, daß $T_{low} = T_{high} = 0$ ist, dann ist das Kantenbild total weiß. Denn für diese Thresholds wird jedes Pixel als Kantenpixel interpretiert wird, da jeder mögliche Grauwert eines Pixels mindestens >= 0 ist.

Die Kombination $T_{low} = 0$ und $T_{high} > 0$ entspricht $T_{low} = T_{high} = 0$, denn von jedem Kantenpixel mit einem Grauwert >= T_{high} erreicht man nur Pixel mit Grauwert >= $T_{low} = 0$, da es nur Grauwerte gibt, die mindestens >= 0 sind. Das Kantenbild ist also weiß.

Wenn $T_{low} = T_{high} <> 0$ gilt, dann wird das Kantenbild nur mit diesem einem Schwellwert behandelt. Nur Pixel mit Grauwerten größer gleich dem Threshold sind dann Kantenpixel.

3.2. Einbindung des Applets in die Html-Seite

3.2.1. Systemvoraussetzungen

Da dieses Applet mit den Swing-Klassen arbeitet, braucht man neben einem Browser noch das Java™ Plug-in 1.3 als Erweiterung zu dem Internet Browser. Das Plug-in kann man unter http://java.sun.com/products/plugin/index.html für Linux, Solaris und Microsoft Windows downloaden.

3.2.2. Html-Seite des Applets

Da das Applet das Java™ Plug-in 1.3 zur Ausführung benötigt und nicht die Virtual Machine, die in den Browsern integriert ist, muß das Applet mit speziellen Befehlen in eine Html-Seite integriert werden (<object ...></object>-Tags für den Internet Explorer, <embed..></embed>-Tags für den Netscape Navigator). Um das Erstellen von Html-Seiten, die das Java™ Plug-in 1.3 verwenden, zu automatisieren und weniger fehleranfällig zu machen, gibt es von Sun den Java Plug-in 1.3 HTML Converter unter http://java.sun.com/products/plugin/index.html zum Downloaden. Dieses Programm ist eine Java-Anwendung, das aus einer Html-Seite, in der man das Applet mit den üblichen Tags <applet>...</applet> eingebunden hat, eine neue plug-in-fähige Html-Seite generiert. Dabei werden die neuen Tags in die Html-Seite eingefügt und die <applet>...</applet>-Tags auskommentiert.

Hinweis:
Schaut man sich das Applet mit der konvertierten Html-Seite im Appletviewer an, werden zwei Instanzen des Appletviewers gestartet. Der Appletviewer interessiert sich in einer Html-Seite nur für die <applet>...</applet>-Tags bzw. dem <object>- oder <embed>-Tag. Da er in der konvertierten Html-Seite neben den neuen Tags auch die alten, eigentlich auskommentierten Tags findet, werden zwei Instanzen gestartet. Das heißt, der Appletviewer ignoriert im Gegensatz zu Browsern Html-Kommentare. Entfernt man den Bereich mit den <applet>...</applet>-Tags, ist das Problem behoben. Interessant ist, daß der Appletviewer, wenn man ihm eine Html-Seite präsentiert, in der das Applet für den Internet Explorer und für den Netscape Navigator eingebunden ist, er nur eine Instanz startet.

Auszug aus einer konvertierten Html-Seite, die das SegmentationApplet einbindet und zwar als erstes für den Internet Explorer und dann für den Netscape Navigator:

```
<!--"CONVERTED_APPLET"-->
<!-- CONVERTER VERSION 1.3 -->
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 200 HEIGHT = 50
codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-
win32.cab#Version=1,3,0,0">
<PARAM NAME = CODE VALUE = "SegmentationApplet" >
<PARAM NAME = "type" VALUE="application/x-java-
applet;version=1.3">
<PARAM NAME = "scriptable" VALUE="false">
<PARAM NAME = IMAGE0  VALUE  ="bridge.jpg">
<PARAM NAME = IMAGE1  VALUE  ="building.jpg">
<PARAM NAME = IMAGE2  VALUE  ="cat.jpg">
<PARAM NAME = IMAGE3  VALUE  ="cells2.jpg">
<PARAM NAME = IMAGE4  VALUE  ="chess.jpg">
<PARAM NAME = IMAGE5  VALUE  ="egg.jpg">
<PARAM NAME = IMAGE6  VALUE  ="flowers.jpg">
<PARAM NAME = IMAGE7  VALUE  ="interior.jpg">
<PARAM NAME = IMAGE8  VALUE  ="parrot.jpg">
<PARAM NAME = IMAGE9  VALUE  ="ship.jpg">
<PARAM NAME = IMAGE10 VALUE  ="taxi.jpg">
<PARAM NAME = IMAGE11 VALUE  ="bloodcells.gif">
<PARAM NAME = IMAGE12 VALUE  ="castle.gif">
<PARAM NAME = IMAGE13 VALUE  ="cat.gif">
<PARAM NAME = IMAGE14 VALUE  ="cells.gif">
<PARAM NAME = IMAGE15 VALUE  ="chessnoise.gif">
<PARAM NAME = IMAGE16 VALUE  ="face.gif">
<PARAM NAME = IMAGE17 VALUE  ="fruit.gif">
<PARAM NAME = IMAGE18 VALUE  ="house.gif">
<PARAM NAME = IMAGE19 VALUE  ="lena.gif">
<PARAM NAME = IMAGE20 VALUE  ="matches.gif">
<PARAM NAME = IMAGE21 VALUE  ="penguin.gif">
<PARAM NAME = IMAGE22 VALUE  ="windows.gif">
<PARAM NAME = IMAGEDIR VALUE  ="img">
<PARAM NAME = COLOR VALUE  = "ffffff">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.3"
CODE = "SegmentationApplet" WIDTH = 200 HEIGHT = 50
IMAGE0  = "bridge.jpg"
IMAGE1  = "building.jpg"
IMAGE2  = "cat.jpg"
IMAGE3  = "cells2.jpg"
IMAGE4  = "chess.jpg"
IMAGE5  = "egg.jpg"
IMAGE6  = "flowers.jpg"
IMAGE7  = "interior.jpg"
IMAGE8  = "parrot.jpg"
IMAGE9  = "ship.jpg"
IMAGE10 = "taxi.jpg"
IMAGE11 = "bloodcells.gif"
IMAGE12 = "castle.gif"
IMAGE13 = "cat.gif"
IMAGE14 = "cells.gif"
IMAGE15 = "chessnoise.gif"
IMAGE16 = "face.gif"
IMAGE17 = "fruit.gif"
IMAGE18 = "house.gif"
IMAGE19 = "lena.gif"
```

```
IMAGE20 = "matches.gif"
IMAGE21 = "penguin.gif"
IMAGE22 = "windows.gif"
IMAGEDIR = "img"
COLOR =  "ffffff"
scriptable=false
pluginspage="http://java.sun.com/products/plugin/1.3/plugin-
install.html">
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>
<!--"END_CONVERTED_APPLET"-->
```

3.2.3. Parameter des Applets

Vor der Konvertierung der Html-Seite des Applets, wird es mit folgenden Befehlen in die Seite eingebunden:

```
<APPLET CODE="SegmentationApplet" WIDTH = 200 HEIGHT = 50>
<PARAM NAME = IMAGE0  VALUE  ="bridge.jpg">
<PARAM NAME = IMAGE1  VALUE  ="building.jpg">
<PARAM NAME = IMAGE2  VALUE  ="cat.jpg">
<PARAM NAME = IMAGE3  VALUE  ="cells2.jpg">
<PARAM NAME = IMAGE4  VALUE  ="chess.jpg">
<PARAM NAME = IMAGE5  VALUE  ="egg.jpg">
<PARAM NAME = IMAGE6  VALUE  ="flowers.jpg">
<PARAM NAME = IMAGE7  VALUE  ="interior.jpg">
<PARAM NAME = IMAGE8  VALUE  ="parrot.jpg">
<PARAM NAME = IMAGE9  VALUE  ="ship.jpg">
<PARAM NAME = IMAGE10 VALUE  ="taxi.jpg">
<PARAM NAME = IMAGE11 VALUE  ="bloodcells.gif">
<PARAM NAME = IMAGE12 VALUE  ="castle.gif">
<PARAM NAME = IMAGE13 VALUE  ="cat.gif">
<PARAM NAME = IMAGE14 VALUE  ="cells.gif">
<PARAM NAME = IMAGE15 VALUE  ="chessnoise.gif">
<PARAM NAME = IMAGE16 VALUE  ="face.gif">
<PARAM NAME = IMAGE17 VALUE  ="fruit.gif">
<PARAM NAME = IMAGE18 VALUE  ="house.gif">
<PARAM NAME = IMAGE19 VALUE  ="lena.gif">
<PARAM NAME = IMAGE20 VALUE  ="matches.gif">
<PARAM NAME = IMAGE21 VALUE  ="penguin.gif">
<PARAM NAME = IMAGE22 VALUE  ="windows.gif">
<PARAM NAME = IMAGEDIR VALUE  ="img">
<PARAM NAME = COLOR VALUE  = "ffffff">
</APPLET>
```

„SegmentationApplet“ ist die von JApplet abgeleitete Klasse, die ausgeführt wird, wenn man die Html-Seite lädt. Sie stellt einen Button direkt in der Html-Seite dar. Um diesen Button ordentlich darzustellen, reicht eine Breite von 200 Pixel und eine Höhe von 50 Pixel. Als nächstes werden die Namen der Bilddateien angegeben, die das Applet verwenden soll. Der Name des Parameters setzt sich „IMAGE“ und einem Zähler für die Anzahl der Bilder zusammen, in diesem Beispiel wurden dreiundzwanzig Bilder dem Applet übergeben, d.h., die Parameternamen beginnen bei „IMAGE0“ und enden bei „IMAGE22“. Als Wert bekommen diese Parameter den Namen der Bilddatei, es

muß auf Groß- und Kleinschreibung geachtet werden. Dann gibt es noch den Parameter „IMAGEDIR“, dort kann man das Verzeichnis benennen, in dem sich die Bilder befinden. Das Verzeichnis ist relativ zum Standort der SegmentationApplet.class, d.h. der JApplet-Klasse, anzugeben. In diesem Beispiel sind die Bilder in einem Unterverzeichnis „img“ zu dem Verzeichnis der JApplet-Klasse. Wenn sich die Bilder im gleichen Verzeichnis wie die Datei SegmentationApplet.class befinden, wird bei „IMAGEDIR“ nichts („“) eingetragen. Als letzten Parameter kann man die Farbe des Applets („COLOR“) bestimmen. Diese Farbe wird nur als Hintergrundfarbe für den Button verwendet, der in der Html-Seite dargestellt wird, d.h., man kann damit die Hintergrundfarbe des Applets an die Hintergrundfarbe der Html-Seite angleichen. Unter „Value“ wird der Farbwert als RGB-Kombination in Hexadezimaldarstellung angegeben. Im Beispiel „ffffff“ entspricht in Dezimaldarstellung einem RGB-Wert, bei dem jeder Anteil den Wert 255 hat, was die Farbe weiß ist. Man kann auch sagen, daß die Farben genauso benannt werden müssen, wie sie in Html-Seiten benannt werden, nur, daß das Kreuz „#“ am Anfang weggelassen wird. Falls hier keine Farbe bezeichnet wird, nimmt das Applet weiß als Hintergrundfarbe an.

3.2.4. Anforderungen an die Bilder

Das Applet verwendet ausschließlich Grauwertbilder. Es wird sowohl das JPG- als auch das GIF-Format vom Applet erkannt. Die Bilder dürfen maximal 180 Pixel breit und 150 Pixel hoch sein. Sind sie größer, werden sie nicht angezeigt.

3.3. Probleme

Das Java™ Development Kit Version 1.2 hat Probleme, wenn es Html-Seiten anzeigen soll, die Tabellen enthalten. Wenn man das Segmentation-Applet mit dem Appletviewer dieser JDK-Version ausführt und dann versucht, auf eine der Hilfeseiten zuzugreifen (über ? → Algorithms → Edge Detection zum Beispiel), erhält man folgende Fehlermeldung:

```
java.lang.Error: Can't build a tr, BranchElement(tr) 7,8
   at
javax.swing.text.html.HTMLEditorKit$HTMLFactory.create(HTMLEditorKit.java:705)
   at javax.swing.text.CompositeView.loadChildren(Compiled Code)
   at javax.swing.text.CompositeView.setParent(CompositeView.java:153)
   at javax.swing.text.CompositeView.replace(Compiled Code)
   ....
```

und die Html-Seite wird nicht angezeigt. Mit dem Appletviewer der Java™ Development Kit Version 1.3 und dem Java™ Plug-in 1.3 tritt dieser Fehler nicht auf. Die Seiten werden korrekt dargestellt.

Wenn man versucht, das Applet mit dem Netscape Navigator 4.7 unter Windows 98 anzuschauen, kann es in einigen Fällen zu Fehler kommen, d.h. das Applet wird nicht angezeigt. In der Statuszeile des Navigators erscheint folgende Meldung „Applet SegmentationApplet notinited“ und in der Java-Konsole wird folgende Ausnahme ausgegeben:

```
java.security.AccessControlException: access denied (java.lang.RuntimePermission
modifyThreadGroup)
    at java.security.AccessControlContext.checkPermission(Unknown Source)
    at java.security.AccessController.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkPermission(Unknown Source)
    ...
```

Dieser Fehler tritt auf, wenn man die Seite mit dem Applet über den Menüpunkt „Datei → Seite öffnen...“ oder per Doppelklick aus dem Windows Explorer öffnen will.

4. Das Applet

4.1. Aufbau der Benutzeroberfläche

Nachdem der Benutzer über den Button „Load Segmentation Applet“ in der Html-Seite das Fenster „Still Image Segmentation“ aufgerufen hat, erscheint die Benutzeroberfläche des Applets.

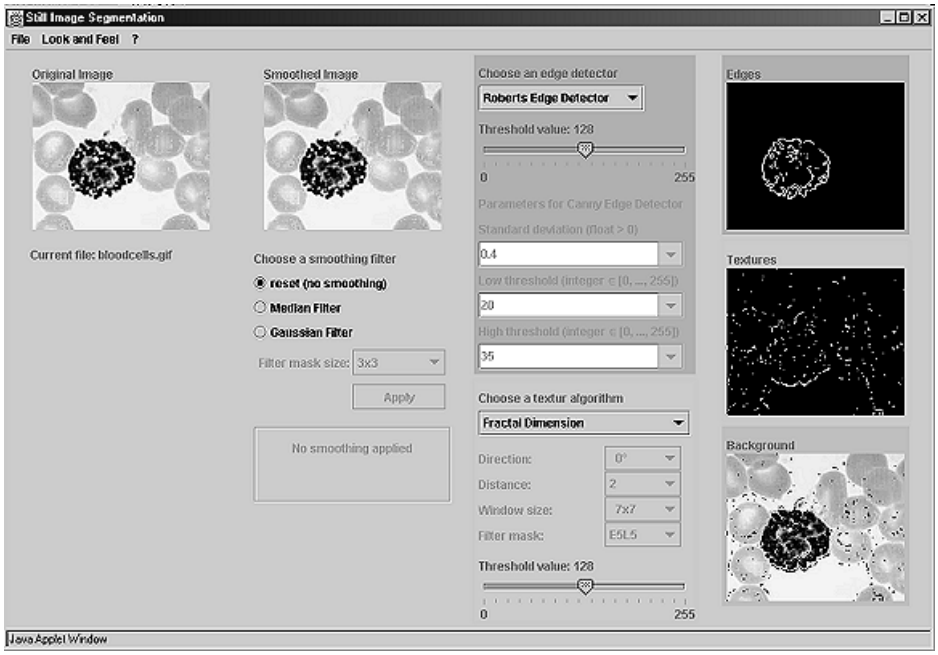


Abbildung 18: Anfangszustand der Benutzeroberfläche

Das Fenster „Still Image Segmentation“ kann in vier Bereiche geteilt werden, dabei wurde das Layout so gestaltet, das es dem Ablauf der Segmentation folgt. Links wird das unbearbeitete Bild und sein Name (z. B. „Current file: bloodcells.gif“) angezeigt. Rechts daneben werden Möglichkeiten angeboten, die Bilder zu glätten. Daran schließt sich der Bereich an, in dem man im oberen Teil Kantenalgorithmen und im unteren Teil Texturalgorithmen auswählen kann. Im letzten Abschnitt des Fensters werden drei Bilder untereinander angezeigt, es sind - von oben nach unten – das Kantenbild, das Texturbild und das Hintergrundbild.

4.1.1. Glättung

Im zweiten Bereich werden zwei Glättungsalgorithmen angeboten, mit denen Rauschen aus Bildern entfernt werden kann. Es gibt drei Optionen „reset (no smoothing)“, „Median Filter“ und „Gaussian Filter“. Beim Starten des Applets ist die reset-Option aktiv und die ComboBox „Filter mask size“, der Button „Apply“, sowie das Label („No smoothing applied“) unterhalb des Buttons sind nicht aktiv. Diese Komponenten werden erst aktiv, wenn einer der Glättungsalgorithmen ausgewählt wird. Dann kann man mittels der ComboBox „Filter mask size“ die Größe der Filtermaske verändern. Zur Auswahl stehen eine 3x3 Pixel und eine 5x5 Pixel große Maske. Damit das Bild geglättet wird, muß explizit der Button „Apply“ gedrückt werden. Wie oft welche Glättung angewendet und mit welcher Maskengröße als letztes geglättet wurde, wird in dem eingerahmten Label angezeigt. Will man die Glättung aufheben, kann mit der reset-Option das Bild in seinen Urzustand zurückversetzt werden. Unter „Smoothed Image“ wird dann das Originalbild dargestellt.

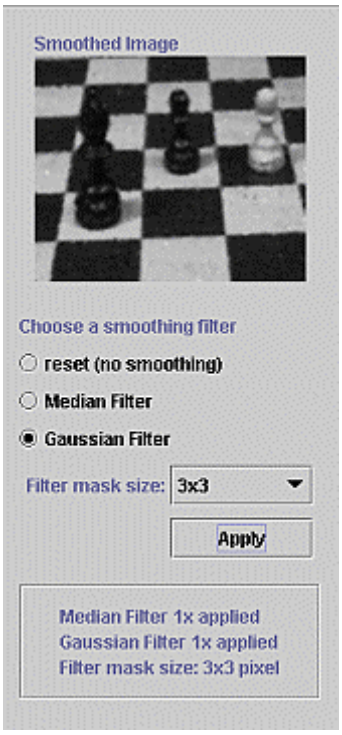


Abbildung 19:
Glättungsalgorithmen

4.1.2. Algorithmen

In den beiden rechten Teilen des Applets werden Kanten- und Texturalgorithmen zur Auswahl gestellt und es werden das Kantenbild, das Texturbild und das Hintergrundbild angezeigt. Die Bildbereiche sind farblich verschieden hinterlegt, ebenso wie die Bereiche der Algorithmen unterschiedliche Hintergründe haben. Die Bereiche mit gleichen Farben gehören logisch zusammen, d.h., blau für die Kantenalgorithmen und das Kantenbild und hellblau für die Texturalgorithmen und das Texturbild. Es wird damit versucht zu verdeutlichen, welches Bild sich ändert, wenn man bestimmte Parameter eingibt. Mit dem Hintergrundbild sind keine Komponenten farblich verbunden, da die Berechnung des Hintergrundbildes nicht vom Benutzer beeinflußt werden kann.

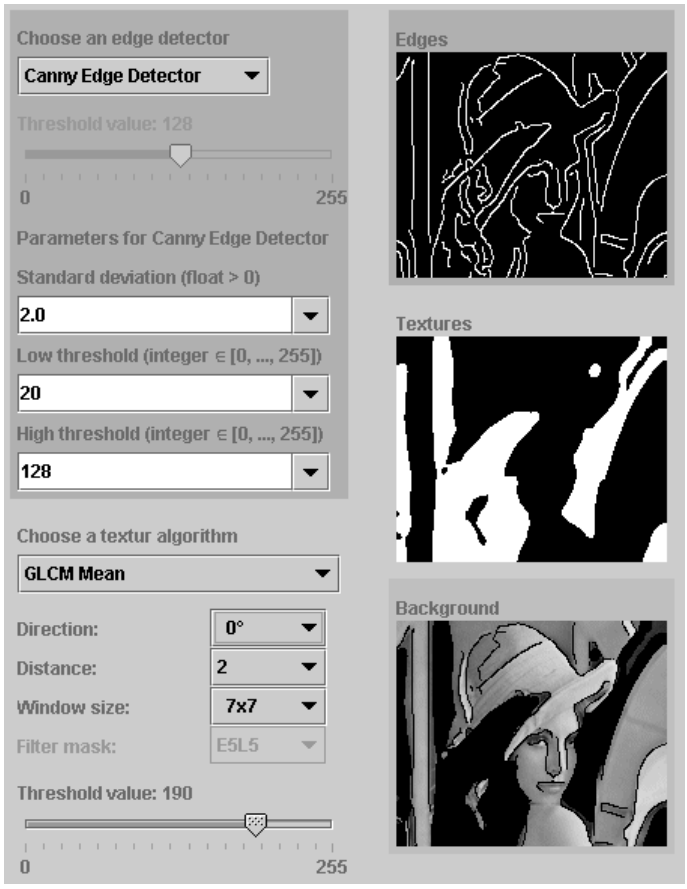


Abbildung 20: Kanten- und Texturalgorithmen

In der ComboBox „Choose an edge detector“ kann man aus verschiedenen Kantendetektoren einen auswählen. Hat man sich für einen Algorithmus entschieden, wird sofort das Kantenbild, in dem Kanten als weiße Linien dargestellt werden, berechnet und unter „Edges“ angezeigt. Will man die Anzahl der angezeigten Kanten beeinflußen, kann man mit dem Slider „Threshold value“ Schwellwerte im Bereich von 0 bis 255 einstellen. Wenn man einen Schwellwert bestimmt hat, wird das Kantenbild erneut berechnet.

Für einen einzigen Kantenalgorithmus wird dieser Slider nicht gebraucht, nämlich für den Canny-Algorithmus. Für ihn sind mehrere Eingabeparameter nötig, die man mit den drei ComboBoxen unter „Parameters for Canny Edge

Detector“ angeben kann. Bei diesen drei ComboBoxen kann der Benutzer nicht nur aus vorgegebenen Werten aussuchen, sondern auch selbst Parameter eingeben. Mit der „Standard deviation...“-ComboBox wird die Standardabweichung für die Glättung mit einem Gauß-Filter, d.h. die Stärke der Glättung angegeben. Je größer die Standardabweichung, desto stärker wird auch geglättet. Aufgrund der Definition der Gauß-Funktion dürfen für die Standardabweichung nur reelle Werte größer Null eingegeben werden. Falls nicht, wird der Benutzer mit einer Fehlermeldung (Division durch Null) auf seinen Eingabefehler aufmerksam gemacht. In den beiden ComboBoxen „Low threshold ...“ und „High threshold ...“ werden die Werte spezifiziert, mit denen der Canny-Algorithmus sein Thresholding durchführen soll. Die Schwellwerte müssen ganze Zahlen aus dem Bereich 0 bis 255 sein.

Beim Aufrufen des Applets ist Roberts-Kantendetektor ausgewählt und der Threshold 128 eingestellt.

Im unteren, hellblauen Teil des Algorithmusbereiches kann man verschiedene Texturalgorithmen auswählen und ihre Parameter eingeben. Als Default ist der Algorithmus Fractal Dimension unter „Choose a textur algorithm“ eingestellt. Für ihn kann man keinen Parameter außer dem Threshold ändern.

Für die Algorithmen, deren Namen mit „GLCM“ anfangen, sind drei ComboBoxen – „Direction“, „Distance“ und „Window size“ – aktiv. Unter „Direction“ kann man 0°, 45°, 90° oder 135° auswählen. Damit wird die Richtung festgelegt, in welche die *Grey level co-occurence* Matrizen die Pixeldistanzen berechnen. Mit „Distance“ kann man genau die Pixeldistanzen bestimmen, entweder 1, 2, 4 oder 6 Pixel. Mit „Window size“ wird die Größe des Fensters beeinflußt, das zur Berechnung der statistischen Maße verwendet wird. Das Fenster kann 7x7, 9x9, 13x13 oder 21x21 Pixel groß sein. Als Standardwerte sind für „Direction“ 0°, für „Distance“ 2 und für „Window size“ 7x7 vorgegeben. Außerdem kann man über den Slider „Threshold value“ einen Schwellwert festlegen. Dieser Threshold wird benutzt, um das Texturbild in verschiedene Regionen zu zerlegen. Damit kann entschieden werden, wohin Teile des Bildes gehören.

Als letztes gibt es bei den Texturalgorithmen noch „Law's Texture Energy Measures”. Für diesen Algorithmus sind die unteren beiden ComboBoxen aktiv, d.h. “Window size” und “Filter mask”. Unter “Window size” stellt man, wie schon bei den GLCM-Algorithmen, die Größe des Fensters für die Berechnung der Energie ein. Mit der ComboBox “Filter mask” sucht sich der Benutzer eine von

Laws Filtermasken aus, mit denen das Bild gefaltet werden soll. Und es ist wieder möglich einen Threshold für das Texturbild einzustellen.

Hinweis:
Die Texturalgorithmen sind alle erheblich langsamer als die Kantenalgorithmen, am schnellsten ist Fractal Dimension, am langsamsten sind die GLCM-Algorithmen. Man muß hier eine Antwortzeit von einigen Sekunden einkalkulieren.

4.1.3. Menüs

Abbildung 21: Menüpunkt "File"

Neben diesen vier Hauptteilen stellt das Applet noch ein Menü bereit.
Unter dem Menüpunkt „File“ kann man mittels „Quit“ das „Still Image Segmentation“-Fenster schließen. Wählt man „Open...“ so öffnet sich der Dialog „Open“, in dem man verschiedene Bilder auswählen kann. In der Liste des Dialogs sind die Dateinamen der Bilder angegeben. Um ein neues Bild zu öffnen, kann man entweder auf einen Namen in der Liste doppelklicken oder man klickt den Namen nur in der Liste an oder man gibt ihn bei „File name“

Abbildung 22: Dialog "Open"

ein und bestätigt dann mit „Return“ bzw. drückt den Button „Open“. Sollte die Eingabe falsch oder das Bild nicht vorhanden sein, wird eine Fehlermeldung angezeigt. Mit der ComboBox „File type“ kann ausgewählt werden, ob GIF- oder JPG-Bilder in der Liste angeboten werden sollen. Wird „Cancel“ betätigt, so wird der Dialog geschlossen, ohne daß ein neues Bild angezeigt wird.

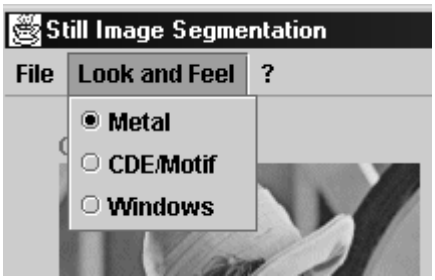


Abbildung 23: Menüpunkt "Look and Feel"

Im Menü „Look and Feel“ kann das Erscheinungsbild des Applets verschiedenen Look and Feels angepaßt werden. Zur Auswahl stehen nur die Look and Feels, die auf dem System des Benutzers vorhanden sind.

Unter dem Menüpunkt „?“ kann man eine

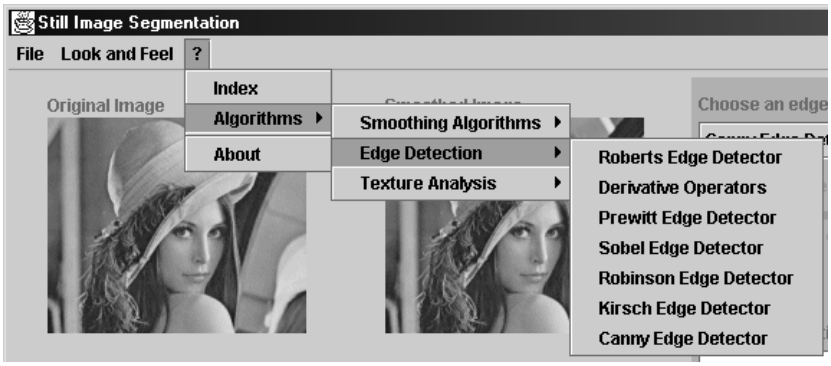


Abbildung 24: Menüpunkt "?"

kurze Beschreibung zu jedem Algorithmus erhalten. Dazu geht man in den Menüpunkt ?→Algorithms und dann in die entsprechende Kategorie, zu welcher der Algorithmus gehört. Einen Überblick über alle Algorithmen gibt der Punkt „Index“. Will man sich die Beschreibung zu einem der Algorithmen anzeigen lassen, so öffnet sich ein neues Fenster namens „Help“. In diesem Fenster werden Html-Dateien dargestellt. Man kann in diesen Html-Dateien mit Hilfe der Links navigieren oder indem man sich für jeden Algorithmus den entsprechenden Punkt aus dem Menü des Applets aussucht. Das Hilfefenster läßt sich über den „Close window“-Button schließen.

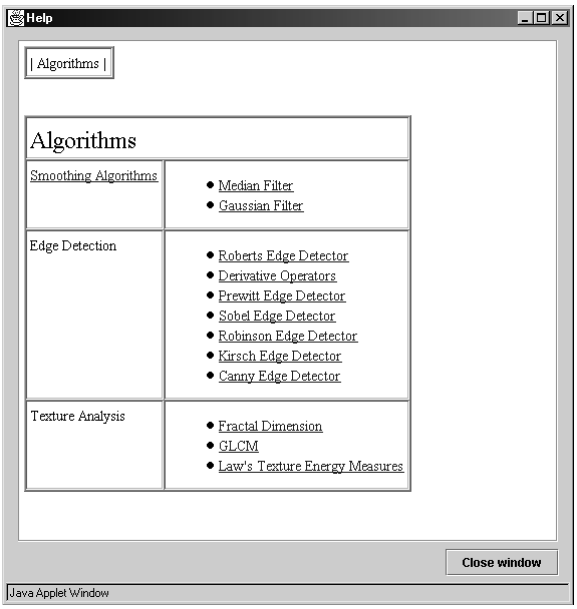


Abbildung 25: eine Hilfeseite, hier der "Index"

Literaturverzeichnis

- Parker, James R.: Algorithms for image processing and computer vision, 1997
- Weeks, Arthur R.: Fundamentals of electronic image processing, 1996
- Schader, Martin; Schmidt-Thieme, Lars: Java™ Eine Einführung, 1999

Adressen im Inernet

- Image Processing Fundamentals (http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Contents.html)
- CVonline: Image Transformations and Filters (http://www.dai.ed.ac.uk/CVonline/transf.htm)
- Java Developer Connection (http://developer.java.sun.com/)
- Bilder:http://www.dai.ed.ac.uk/HIPR2/typelib.htm