Formelsammlung Echtzeit Bildverarbeitung

Mario Felder

4. Juli 2014

Inhaltsverzeichnis

Kapitel 1

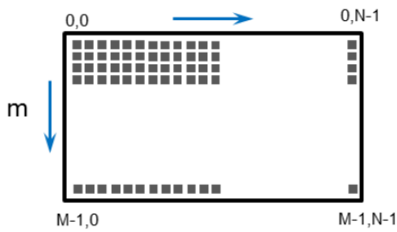
Einführung

Die Bilddaten können mathematisch als Matrix beschrieben werden:

$$I = [I_{m,n}]$$

mit $0 \leq m \leq M - 1$ und $0 \leq n \leq N - 1$

Achtung: MATLAB verwendet das Intervall $[1, N]$ bzw. $[1, M]$



1.1 Rasterung

Die Rasterung ist ein Mass für die Detailtreue eines Bildes. Bei gegebener CCD- bzw. CMOS-Sensorgrösse (M x N Pixel) wird die Auflösung durch die geometrische Abbildung bestimmt.

Dabei gelten folgende Zusammenhänge:

$$\frac{1}{g} + \frac{1}{b} = \frac{1}{f} \qquad \Leftrightarrow \qquad b = \frac{f \cdot g}{g - f}$$

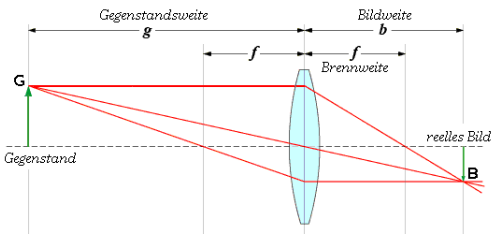
f: Brennweite
b: Bildweite
g: Gegenstandweite

Häufig ist $g \gg b$ und es kann somit $b = f$ gesetzt werden.

Weiter ergibt sich daraus:

$$\frac{B}{G} = \frac{b}{g}$$

B: Bildgrösse
G: Gegenstandsgrösse



Auflösung bei einem Sensor mit NxM Pixel Auflösung:

$$G_b = \frac{g \cdot W}{b \cdot N}$$
$$G_h = \frac{g \cdot H}{b \cdot M}$$

G_b : Auflösung in $\frac{mm}{Pixel}$ (Breite)
G_h : Auflösung in $\frac{mm}{Pixel}$ (Höhe)

W : Breite des Sensors $[mm]$
H : Höhe des Sensors $[mm]$
N : Horizontale Pixel
M : Vertikale Pixel

1.2 Nachbarschaftsrelationen und Abstand

Vierer-Nachbarschaft:

	$m-1, n$	
$m, n-1$	m, n	$m, n+1$
	$m+1, n$	

Achter-Nachbarschaft:

$m-1, n-1$	$m-1, n$	$m-1, n+1$
$m, n-1$	m, n	$m, n+1$
$m+1, n-1$	$m+1, n$	$m+1, n+1$

Euklidische Abstandsnorm:

$$d(I_{m,n}, I_{p,q}) = \sqrt{(m-p)^2 + (n-q)^2}$$

Maximum Abstandsorm:

$$d(I_{m,n}, I_{p,q}) = max\left(|m-p|, |n-q|\right)$$

1.3 Globale Charakterisierung von Bildern

1.3.1 Histogramm

Das Histogramm gibt die absolute oder relative $p_I(g)$ Häufigkeit aller Grauwerte $g \in [0, 255]$ eines Bildes an.

Relative Häufigkeit:

$$\begin{aligned} &0 \leq p_I(g) \leq 1 \qquad , \forall g \\ &\sum_g p_I(g) = 1 \end{aligned}$$

Kumulative Häufigkeit:

$$h_I(g) = \sum_{g' \leq g} p_I(g') \qquad\qquad , h_I(255) = 1$$

1.3.2 Mittelwert

$$\mu_I = \frac{1}{M \cdot N} \sum_{m,n} I_{m,n} = \sum_g g \cdot p_I(g)$$

1.3.3 Varianz

$$\sigma_I^2 = \frac{1}{M \cdot N} \sum_{m,n} (I_{m,n} - \mu_I)^2 = \sum_g (g - \mu_I)^2 \cdot p_I(g)$$

Lösung in MATLAB:

```
%read image
Image = imread('sample.png');

%using formula
[M, N] = size(Image);
mu = sum(Image(:))/(M*N)

sd = sum((double(Image(:)) - mu).^2)/(M*N);
sd = sqrt(sd)

```

```
%with MATLAB function
mu = mean2(Image)
sd = std2(Image)

%using the histogram
[Count, Value] = imhist(Image);
RelCount = Count/sum(Count);
mu = sum(Value .* RelCount)

sd = sum((Value - mu).^2 .* RelCount);
sd = sqrt(sd)
```

Lösung in C:

```
for(int m = 0; m < M; m++){
   for(int n = 0; n < N; n++){
      sum += Image[m][n];
   }
}
mu_I = sum/(M*N);

/**
*
* TODO
*
**/
```

Kapitel 2

Punktoperationen und Bildverknüpfungen

2.1 Transformationstabellen (LUT)

Jedem Grauwert $g \in G$ wird einen Grauwert $g' \in G'$ über eine Abbildung f zugeordnet:

$$f : G \to G' \qquad , f(g) = g'$$

Lösung in MATLAB:

```
%read image
Image = imread('sample.png');

%LUT for inverse image
LUT_Inv = uint8([255:-1:0]);

%apply LUT
ImageInv = intlut(Image, LUT_Inv);

%LUT for screener
LUT_Scr = uint8(1.5*[0:255]);

%apply LUT
```

```
ImageScr = intlut(Image, LUT_Scr);
```

2.2 Lineare und nichtlineare Grauwerttransformationen

Eine allgemeine lineare Grauwerttransformation lässt sich in folgender Notation schreiben:

$$f : [0, 255] \to [0, 255] \in \mathbb{R}$$

$$f(g) := \begin{cases} 0 & \text{,falls } (g + c_1) \cdot c_2 < 0 \\ 255 & \text{,falls } (g + c_1) \cdot c_2 > 255 \\ (g + c_1) \cdot c_2 & \text{,sonst} (c_1, c_2 \in \mathbb{R}) \end{cases}$$

2.2.1 Spreizung

Sind die Grauwerte eines Bildes auf das Intervall $[g_1, g_2]$ beschränkt, so kann durch Anwendung der linearen Grauwerttransformation als Spreizung die werte auf das gesamte Intervall $[0, 255]$ verteilt werden:

$$f : [0, 255] \to [0, 255] \in \mathbb{R}$$

$$f(g) := \begin{cases} 0 & \text{,falls } g < g_1 \\ 255 \cdot \frac{g - g_1}{g_2 - g_1} & \text{,falls } g \in [g_1, g_2]. \\ 255 & \text{,falls } g > g_2 \end{cases}$$

Lösung in MATLAB:

```
%read image
Image = imread('sample.png');

%LUT for spreading gray values
LUT_Spread = uint8(([0:255]-50)*2);

%apply LUT
ImageSpread = intlut(Image, LUT_Spread);
```

2.2.2 Gammakorrektur

Ein wichtiges Beispiel der nichtlinearen Grauwerttransfomration ist die Gammakorrektur. Der Ursprung dieser nitlinearen Korrektur der Grauwert liegt in der Nichtlinearität von Aufnahme- und Darstellungssystemen.:

$$f : [0, 255] \to [0, 255] \in \mathbb{R}$$

$$f(g) := 255 \cdot \left(\frac{g}{255}\right)^{\gamma}$$

Lösung in MATLAB:

```
%read image
Image = imread('sample.png');

%LUT for gamma correcture
LUT_Gamma = uint8(255*([0:255]/255).^0.5);

%apply LUT
ImageGammad = intlut(Image, LUT_Gamma);
```

2.3 Histogrammausgleich

Die Idee des Histogrammausgleichs ist, die Grauwerte so zu verteilen, dass jeder gleich häufig vorkommt. Dies kann allerdings nicht ganz erreicht werden, da die Grauwerte diskret sind. Näherungsweise kann die kumulierte Häufigkeit $h_I(g)$ herangezogen werden. Bei einer konstanten absoluten Häufigkeit, würde die kumulierte Häufigketi linear anwachsen.
Die entsprechende Transformation:

$$f : [0, 255] \to [0, 255] \in \mathbb{R}$$

$$f(g) := g_{max} \cdot \sum_{g'=0}^{g} p_I(g')$$

Lösung in MATLAB:

```
%read image
Image = imread('sample.png');

[Hist, Val] = imhist(Image);
CumHist = cumsum(Hist)/sum(Hist);

%LUT for histogram equalization
LUT_Equ = uint8(CumHist*255);
%apply LUT
ImageEqu = intlut(Image, LUT_Equ);
```

2.4 Arithemtische und logische Bildverknüpfungen

Während die Punktoperationen auf Einzelbildern vor allem der besseren optischen Darstellung von Bildern dienen, eröffnen Punktoperationen auf mehreren Bildern ein grosses Repertoire an Methoden, die schon erste einfache Computer Vision Anwendungen erlauben.

2.4.1 Differenzbildung zur Motiondetektion

Eine Methode zur Change Detektion basiert auf der Berechnung von Differenzbildern. Dabei wird vorausgegangen, dass eine Serie von Bildern als Funktion der Zeitstempel aufgnommen werden:

$$I(t) = I(t_k) = I(k \cdot \Delta t) = I_k$$

Differenzbild Berechnung:

$$\Delta I_{k+n} = \frac{1}{2} \cdot (255 + I_{k+n} - I_k) = \frac{1}{2} \cdot (255 + I((k+n) \cdot \Delta t) - I(k \cdot \Delta t)$$

Auf das Differenzbild kann noch eine Schwellwertioperation agewandt werden:

$$f : [0, 255] \rightarrow \{0, 255\}$$

$$f(g) := \begin{cases} 0 & \text{, falls } g < g_1 \vee g > g_2 \\ 255 & \text{, falls } g \in [g_1, g_2] \end{cases}$$

$$\text{mit} \quad g_1 = 128 - \mathit{threshold}, \quad g_2 = 128 + \mathit{threshold}$$

Lösung in MATLAB:

```
function [ThreshImage, DiffImage] =
    MotionDetektionFunct(ImageAct, ImageOld)

global Threshold

%this is the threshold value (chosen manually)
Threshold = 15;

%cast
ImageAct = double(ImageAct);
ImageOld = double(ImageOld);

%calcuate difference image
DiffImage = (255 + ImageAct - ImageOld)/2;

%calculate the threshold image
ThreshImage = (abs(DiffImage-128) > Threshold);

DiffImage = uint8(DiffImage);
```

2.4.2 Hintergrundschätzung durch gleitenden Mittelwert

Ziel ist es, eine möglichst exakte Hypothese des unbeweglichen Hintergrundes $B(t_k) = B(k \cdot \Delta t) = B_k$ der Bilder I_k zu ermitteln. Angenommen, dass der Hintergrund jedes Pixels mehrheitlich sichtbar ist, kann durch ein einfaches gleitendes Mittel eine brauchbare Hypothese B_k bestimmt werden:

$$B_k = \alpha \cdot B_{k-1} + (1-\alpha) \cdot I_k \qquad , a \in]0,1[$$

Eine plötzlich auftretende stationäre Änderung in der Bildfolge I_k ist nach $n \cdot \Delta t$ Zeitshcritten zu folgendem Anteil p in die Hintergrundhypothese B_k integriert:

$$p = 1 - \alpha^n$$

Lösung in MATLAB:

```matlab
function [ThreshImage, DiffImage, BackGround] =
    GleitendesMittelFunct(ImageAct, BackGround, Params)

%make everything double
BackGround = double(BackGround);
ImageAct = double(ImageAct);

%calcuate forground estimate
DiffImage = abs(BackGround-ImageAct);
%estimate new background as sliding average
BackGround = Params.AvgFactor*BackGround+...
    (1-Params.AvgFactor)*ImageAct;

%calculate the threshold image
ThreshImage = DiffImage > Params.Threshold;
```

2.4.3 Hintergrundschätzung durch statistische Analyse

Die grundlegende Idee ist, für jedes Pixel individuell die Helligkeitsschwankungen zu messen und durch ein einfaches statistisches Modell zu approximieren. Letzteres besteht in einer Gauss‘schen Approximation der Grauwertfluktuationen des Pixels durch das Paar aus Mittelwert und Varianz (μ, σ). Hierbei wird für jedes Pixel individuell die Schätzung von Mittelwert und Varianz über die Zeitschritte $k \cdot \Delta t$ folgendermassen durchgeführt:

$$\begin{aligned} \mu_k &= \alpha \cdot \mu_{k-1} + (1-\alpha) \cdot I_k \\ \sigma_k &= \alpha \cdot \sigma_{k-1} + (1-\alpha) \cdot (\mu_k - I_k)^T \cdot (\mu_k - I_k) \qquad , \alpha \in]0,1[\end{aligned}$$

Das Aufdatieren erfolgt nur, wenn sich der Mittelwert μ_k und aktueller Pixelwert I_k nur um weniger als $thr \cdot \sigma_k$ unterscheiden.

Kapitel 3

Filteroperatoren im Ortsraum

Bei diesen Filteroperationen werden für die Transformation enes Pixels auch die Werte der Nachbarpixel in Betracht gezogen.

$$f : G \to G'$$
$$f(g) = f_{I_{p,q}}(I_{m,n}) = g' \qquad , (p,q) \neq (m,n)$$

Klassifikationen der Filteroperatoren:

homogene Filter:
Die Berechnung der Transformation $f(g)$ wird für jedes Pixel unabhängig von dessen Position im Bild vorgenommen.

inhomogene Filter:
Die Berechnung der Transformation $f(g)$ hängt explizit von der Position des Pixels im Bild ab.

lineare Filter:
Die Berechnung der Transfomration $f(g)$ hängt für jedes Pixel linear ovn $g = I_{m,n}$ und den Werten der Nachbarpixel $I_{p,q}$ ab.

nicht lineare Filter:
Für die Berechnung der Transformation $f(g)$ eines Pixels werden die Werte von $g = I_{m,n}$ und der Nachbarpixel $I_{p,q}$ in nicht linearer Weise verknüpft.

3.1 Lineare Filter - Faltung

Mathematische Beschreibung der Faltung $I \otimes h$ eines Bildes $I_{m,n}$ mit einer Maske $h_{p,q}$:

$$I \otimes h : I_{m,n} \rightarrow \sum_{p=-u}^{u} \sum_{q=-v}^{v} I_{m-p,n-q} \cdot h_{p,q}$$

Rechengesetzte:

Kommutativität: $I \otimes J = J \otimes I$

Assoziativität: $(I \otimes J) \otimes K = I \otimes (J \otimes K)$

Distributivität: $I \otimes (J + K) = I \otimes J + I \otimes K$

skalare Assoziativität: $a \cdot (I \otimes J) = (a \cdot I) \otimes J = I \otimes (a \cdot J)$

Durch Anwendung der Rechenregeln kann eine Maske effizient angewendet werden, indem sie aus zwei eindimensionalen Masken zusammengesetzt wird:

$$I \otimes h = (I \otimes h_x) \otimes h_y$$

3.1.1 Glätten von Bildern (Tiefpass)

Bei der Glättung wird ein Pixel durch die Mittelung des Pixels mit den Nachbarpixeln ersetzt. Dies dient zur Rauschunterdrückung oder als Vorstufe zum Dezimieruren der räumlichen Auflösung (eng. down sampling).
Verwendete Standardfilter sind Rechteck- oder Gaussmasken:

$$R = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Damit der Wertebereich der Abbildung $G' = [0\ 255]$ bleibt, muss die Summe über alle Maskenkoeffizienten 1 ergeben.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%convert to double
Image = double(Image);

%apply a filter of size 2x2
ImageFiltered = imfilter(Image,ones(2)/4);
imshow(Image1,[0 255]);
```

3.1.2 Kantenhervorhebung (Hochpass)

Harte Grauwertübergänge eines Bildes werden verstärkt, während weiche Übergänge abgeschwächt werden. Hierbei ist der Gradienten Filter eine wichtige Filterklasse. Diese berechnet den Gradienten, d.h. die

Ableitung der Grauwerte $I_{m,n}$ eines Bildes.

$$\frac{\partial I(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{I(x+\Delta x, y) - I(x-\Delta x, y)}{2 \cdot \Delta x} = \frac{I_{m,n+1} - I_{m,n-1}}{2}$$
$$\frac{\partial I(x,y)}{\partial y} = \lim_{\Delta y \to 0} \frac{I(x, y+\Delta y) - I(x, y-\Delta y)}{2 \cdot \Delta y} = \frac{I_{m+1,n} - I_{m-1,n}}{2}$$

Mit folgenden Filtern kann diese Berechnung des Gradienten durchgeführt werden:

$$h_x = \frac{1}{2} \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad h_y = \frac{1}{2} \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Da diese fehleranfällig auf Rausche sind, wird jeweils mit einem Glättungsfilter kombiniert.

Prewitt-Filter:

$$h_x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Sobel-Filter:

$$h_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%convert to double
Image = double(Image);
```

```

%choose the filters
Sobel = 1;
if Sobel == 1
      DX = [1;2;1] * [-1 0 1];
      DY = DX';
else
      DX = ones(3,1) * [-1 0 1];
      DY = DX';
end

%apply the DX and DY filter
ImageDx = uint8(128+imfilter(Image, DX));
ImageDy = uint8(128+imfilter(Image, DY));
```

Der Gradient kann auch als Vektor dargestellt werden mit dem Betrag $\frac{\mathrm{d}I}{\mathrm{d}r}$ und Winkel α:

$$\frac{\mathrm{d}I}{\mathrm{d}r} = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \qquad \alpha = \arctan\frac{\frac{\partial I}{\partial y}}{\frac{\partial I}{\partial x}}$$

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%convert to double
Image = double(Image);

%choose the filters
Sobel = 1;
if Sobel == 1
      DX = fspecial('sobel')';
   DY = fspecial('sobel');
else
      DX = fspecial('prewitt')';
   DY = fspecial('prewitt');
end

%apply the DX and DY filter
ImageDx = imfilter(Image, DX);
```

```
ImageDy = imfilter(Image, DY);

%calculate the norm of the derviative
ImageDr = sqrt(ImageDx.^2 + ImageDy.^2);

%determine the angle (atan2 gives back the whole interval
    ]-pi , pi[ )
Angle = pi+atan2(ImageDy, ImageDx);
%use only those values that are above a given threshold
Angle(ImageDr < threshold) = 0;

%plot
imshow(ImageDr, [0 255]);
imshow(Angle, [0 255]);
map=colormap(jet);
map(1,:) = 0;
colormap(map)
colorbar;
```

3.1.3 Bildschärfung

Für die Bildschärfung wird der Laplace Operator verwendet, welcher ein linearer richtungsunabhängiger Operator ist. Er ist definiert als die Summe der zweiten Ableitung:

$$\Delta I = L = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Die Approximation für diskrete Werte $I_{m,n}$ des Bildes (nur für x, y analog dazu):

$$\begin{aligned} I_{xx} &\approx \frac{I_x(x+1,y) - I_x(x,y}{1} \\ &\approx \frac{I(x+1,y) - I(x,y}{1} - \frac{I(x,y) - I(x-1,y}{1} \\ \frac{\partial^2 I}{\partial x^2} &\approx I_{x+1} - 2 \cdot I_x + I_{x-1} \end{aligned}$$

Die dazugehörige Maske:

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Für die Bildschärfung wird der Laplace Operator in Kombination mit dem Identischen Operator I verwendet:

$$I + \beta \cdot L$$

β: Grad der Bildschärfung

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%convert to double
Image = double(Image);

%define the filter
Beta = 0.5;
Mask = [0 0 0; 0 1 0; 0 0 0] + Beta * [0 -1 0; -1 4 -1; 0 -1 0];

%apply the filter
ImageEnh = uint8(imfilter(Image, Mask));
```

3.2 nichtlineare Filter

3.2.1 Rangordnungsoperatoren - Median-Filter

Bei Randordnungsverfahren wird der Pixelwert anhand der Nachbarpixel neu gesetzt. Die grösse der Umgebung kann dabei variieren. Bekannte Verfahren sind:

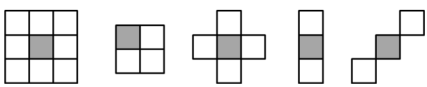
- Minimum Filter: Es wird der minimale Pixelwert der Nachbarpixel gewählt
- Median Filter: Es wird der mittlere Pixelwert der Nachbarpixel gewählt
- Maximum Filter: Es wird der maximale Pixelwert der Nachbarpixel gewählt

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%apply a median filter
ImageMedian = medfilt2(ImageNoise, [3 3]);
ImageMedian = ordfilt2(ImageNoise, 5, ones(3,3));

%apply a min filter
ImageMin = ordfilt2(ImageNoise, 1, ones(3,3));

%apply a max filter
ImageMax = ordfilt2(ImageNoise, 9, ones(3,3));
```

3.2.2 morphologische Operatoren

Bei morphologischen Operationen wird die Umgebung eines Pixels auf eine bestimmte Struktur hin untersucht. Dazu wird ein Strukturelement verwendet, welches einen definierten Referenzpunkt hat.

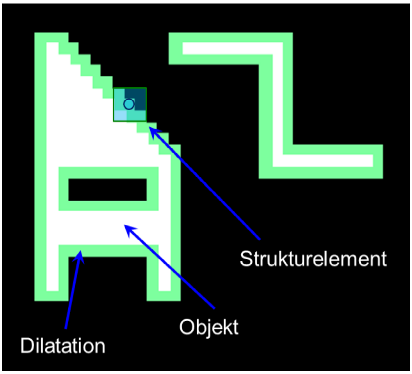


Morphologische Operatoren dienen dazu die Form von Objekten in definierte Weise zu verändern. Dies sind typisch:

- Löschen kleiner Objekte (z.B. Pixelrauschen)
- Schliessen von Löchern in Objekten
- Zusammenfassen von räumlich nahen Objekten
- Löschen aller Pixel im Inneren eines Objektes
- Reduzieren eines Objektes auf das Skelett

Dilatation

Bei der Dilatation wird das Strukturelement über das Bild geschoben. Dabei werden alle Referenzpunkte des Strukturelementes markiert, bei denen eine nicht verschwindende Schnittmenge mit dem Objekt auftritt.

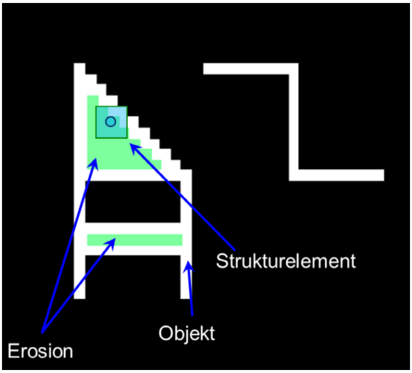


Mathematische Beschreibung:

$$I \otimes h = \left\{ (m,n) | (\hat{h})_{m,n} \cap I \neq \{\} \right\}$$

Erosion

Bei der Dilatation wird das Strukturelement über das Bild geschoben. Dabei werden alle Referenzpunkte des Strukturelementes markiert, bei denen das Strukturelement ganz in dem Objekt enthalten ist.



Mathematische Beschreibung:

$$I - h = \{(m, n)|(h)_{m,n} \subset I\}$$

Schliessung & Öffnung

Wen die Dilatation und Erosion kombiniert werden, entstehen weitere wichtige Operatoren. Wird zuerst eine Dilatation dann eine Erosion ausgeführt, so entsteht eine Schliessung. Umgekehrt eine Öffnung. Mit der Schliessung können kleine Lücken geschlossen werden. Die Öffnung kann Bildrauschen entfernen, da kleine Teile entfernt werden.

Schliessung:

$$I \bullet h = (I \otimes h) - h$$

Öffnung:

$$I \circ h = (I - h) \otimes h$$

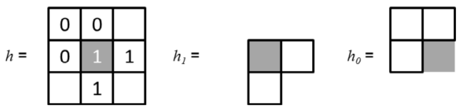
Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
```

```
Image = imread('sample.png');

%define the structure element
StrucElem = strel('rectangle',[2 2]);

%do a dilation (result is binary)
ImageDil = imdilate(Image, StrucElem);

%do an erosion (result is binary)
ImageErode = imerode(Image, StrucElem);

%do a closure
ImageClose = imclose(Image, StrucElem);

%do an opening
ImageOpen = imopen(Image,StrucElem);
```

Hit- und Miss-Operation

Bei dieser Operation wird nicht nur geschaut, ob das Strukturelement in einem Objekt enthalten ist, sondern ob die Nachbarschaft eine vorgegebene Struktur hat.



0: Das Pixel muss den Wert 0 haben

1: Das Pixel muss den Wert 255 (bzw. 1) haben

„”: Der Wert des Pixels wird nicht in Betracht gezogen.

Die mathematische Beschreibung:

$$I \pm h = (I - h_1) \cap (I^C - h_0)$$

I^C: Das Komplement aller Pixel ungleich Null vom Bild I

Hit- und Miss-Operation wird meist in Verbindung mit Verdünnungsund Verdickungs-Operationen verwendet:

$$\text{thin}(I, h) = I \cap (I \pm h)^C$$

Kapitel 4

Fourier-Transformation

4.1 1D Fourier-Transformation

Die Fourier Transformationen dient dazu, für eine Funktion $h(x)$ im Ortsraum das zugehörige Frequenzspektrum im Ortsfrequenzraum $\hat{h}(f)$ zu bestimmen. Dabei gibt es grundsätzlich vier mögliche Anwendungsfälle und zugehörige Formulierungen der Fourier Transformation (FT):

Kontinuierliche FT einer aperiodischen Funktion $h(x)$:

$$\hat{h}(f) = \int_{-\infty}^{\infty} h(x) \cdot \mathrm{e}^{-\mathrm{j} \cdot 2\pi \cdot f \cdot x} \mathrm{d}x \qquad h(f) = \int_{-\infty}^{\infty} \hat{h}(x) \cdot \mathrm{e}^{\mathrm{j} \cdot 2\pi \cdot f \cdot x} \mathrm{d}f$$

Kontinuierliche FT einer X_0-periodischen Funktion $h(x)$:

$$\hat{h}[n] = \frac{1}{X_0} \int_{0}^{X_0} h(x) \cdot \mathrm{e}^{-\mathrm{j} \cdot 2\pi \cdot f_0 \cdot x} \mathrm{d}x \qquad h(f) = \sum_{n=-\infty}^{\infty} \hat{h}[n] \cdot \mathrm{e}^{\mathrm{j} \cdot 2\pi \cdot n \cdot f_0 \cdot x}$$

$f_0 = \frac{1}{X_0}$

Im Falle einer periodischen Funktion besteht das Frequenzspektrum nur aus diskreten Werten an den Frequenzen $f_n = n \cdot f_0$.

Diskrete FT einer aperiodischen Funktion für äquidistante Abtastpunkte $h_n = h(n \cdot x_s)$:

$$\hat{h}(f) = \sum_{n=-\infty}^{\infty} h[n] \cdot \mathrm{e}^{-\mathrm{j} \cdot 2\pi \cdot f \cdot n \cdot x_s} \qquad h[n] = \frac{1}{f_s} \int_0^{f_s} \hat{h}(f) \cdot \mathrm{e}^{\mathrm{j} \cdot 2\pi \cdot n \cdot x_s \cdot f} \mathrm{d}f$$

$f_s = \frac{1}{x_s}$

Bei Abtastung eines Signals mit dem Intervall x_s entsteht ein periodisches Frequenzspektrum der Periode f_s.

Diskrete FT einer X_0-periodischer Funktion für äquidistante Abtastpunkte $h_n = h(n \cdot x_s)$:

$$\hat{h}[k] = \sum_{n=0}^{N-1} h[n] \cdot \mathrm{e}^{-\mathrm{j} \cdot 2\pi \cdot \frac{k \cdot n}{N}} \qquad h[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{h}[k] \cdot \mathrm{e}^{\mathrm{j} \cdot 2\pi \cdot \frac{k \cdot n}{N}}$$

$x_s = \frac{X_0}{N}$

Wird ein periodisches Signal mit $n \cdot x_0$ Intervallpunkten abgetastet, so hat die FT ein ebenfalls diskretes und periodisches Frequenzspektrum der Periode $f_s = \frac{1}{x_s} = \frac{N}{X_0}$.

Rechenregeln

Linearität:	$\mathcal{F}\{\lambda \cdot h(x)\} = \lambda \cdot \mathcal{F}\{h(x)\}$
	$\mathcal{F}\{h(x) + g(x)\} = \mathcal{F}\{h(x)\} + \mathcal{F}\{g(x)\}$
Verschiebung:	$\mathcal{F}\{h(x + x_0)\} = \mathrm{e}^{\mathrm{j} \cdot 2\pi \cdot f \cdot x_0} \cdot \mathcal{F}\{h(x)\}$
Faltungssatz:	$\mathcal{F}\{h(x) \otimes g(x)\} = \mathcal{F}\{h(x)\} \cdot \mathcal{F}\{g(x)\}$

4.2 Diskrete 2D Fourier-Transformation

Bei Annahme dass die Funktion in x-Richtung X_0-periodisch und in y-Richtung Y_0-periodisch ist ($X_0 = N \cdot x_s$, $Y_0 = M \cdot y_s$), so ist die 2D

DFT folgendermassen definiert:

$$\hat{h}[l,k] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} h[m,n]\cdot \mathrm{e}^{-\mathrm{j}\cdot 2\pi\cdot\left[\frac{l\cdot m}{M}+\frac{k\cdot n}{N}\right]}$$

$$h[m,n] = \frac{1}{M\cdot N}\sum_{l=0}^{M-1}\sum_{k=0}^{N-1} \hat{h}[l,k]\cdot \mathrm{e}^{\mathrm{j}\cdot 2\pi\cdot\left[\frac{l\cdot m}{M}+\frac{k\cdot n}{N}\right]}$$

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%calculate DFT (keep size of image)
Image_FFT = abs(fft2(Image));

%use fftshift to center the DFT
Image_ShiftFFT = fftshift(Image_FFT);

%caluclate the inverse fft
Inv_FFT = abs(ifft2(Image_ShiftFFT));
```

4.3 Periodische Strukturen

Periodische Strukturen führen auch zu einem periodischen Frequenzspektrum. Wenn das Bild jedoch nicht über den Bildrand hinaus periodisch ist, treten Fehler in der DFT auf. Dies kann korrigiert werden, indem eine Fensterfunktion angewendet wird, welche zum Rand des Bildes stetig nach Null abfällt.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
```

```
Image = imread('sample.png');

%apply Hanning window
Hann = hann(ImgS(1))*hann(ImgS(2))';
Image_FFT_Ham = fft2(double(Image).*Hann);
imshow(log(abs(fftshift(Image_FFT_Ham))),[]);
```

4.4 Unterdrückung periodischer Störungen

Periodische Störungen haben auch eine periodische Auswirkung im Frequenzspektrum. Durch einen Nochtfilter können diese selektiv ausgeblendet werden. So kann ein Grossteil des Rauschens eliminiert werden.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');
%shortcut
ImgS = size(Image);

SizeFFT = 256;

%do fft calculation
Image_FFT = fft2(Image, SizeFFT, SizeFFT);

%use imtool to find the positions of the frequencies
%imtool(log(abs(fftshift(Image_FFT))),[]);

NotchPos = [102, 107, 143, 95, 170, 116, 160, 151,114,
    165, ...
    90, 143, 140, 185, 118, 70]';

%apply the filter
Filter = NotchFilter( size(Image_FFT), NotchPos,
    6*ones(1,length(NotchPos)) );

%do the inverse fft
ImageIfft = abs(ifft2(Image_FFT.*fftshift(Filter)));
```

```
%chose only the relevant part
ImageFilt = ImageIfft(1:ImgS(1), 1:ImgS(2));
```

4.5 Deconvolution

Ein Bild, welches durch einen Filter unscharf gezeichnet wurde, kann durch Deconvolution wieder hergestellt werden. Dabei wird der Faltungssatz angewendet.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%construct the maks
Mask = fspecial('gaussian',7,10);

%sigma of added gaussian noise
V = .000;

%apply noise and blur
BlurredNoisy = imnoise(imfilter(Image,Mask),'gaussian',0,
    V);

%image buffer for the result
WT = zeros(size(Image));
WT(5:end-4,5:end-4) = 1;

%we assume to now only the size of the mask
InitMask = ones(size(Mask));
%perform deconvolution (P - recovered Mask, J - deblurred
    Image)
[J P] = deconvblind(double(BlurredNoisy),InitMask,
    20,10*sqrt(V),WT);
```

Kapitel 5

Segmentierung und Merkmalsextraktion

5.1 Automatisierte Schwellwertbestimmung

Das Verfahren von Otsu basiert darauf, dass alle Grauwerte in zwei Klassen eingeteilt werden ($C_0 = \{0, 1, ..., K\}$ und $C_1 = \{K+1, ..., 255\}$). K stellt den zu optimierenden Schwellwert dar. Es wird angenommen, dass das Bild eine bimodale Grauwertverteilung hat. Das Bild zerfällt in zwei Klassen mit relativ homogenen Grauwerten. Der Schwellwert K wird so optimiert, dass die Varianzen innerhalb der Klassen möglichst klein sind.

Wahrscheinlichkeit für Auftreten von Klasse C_0 bzw. C_1:

$$\omega_0 = \sum_{g=0}^{K} p_I(g) \qquad \omega_1 = \sum_{g=K+1}^{255} p_I(g)$$

Mittelwert der Grauwerte von Klasse C_0 bzw. C_1:

$$\mu_0 = \frac{1}{\omega_0} \sum_{g=0}^{K} p_I(g) \cdot g \qquad \mu_1 = \frac{1}{\omega_1} \sum_{g=K+1}^{255} p_I(g) \cdot g$$

Varianz der Grauwerte von Klasse C_0 bzw. C_1:

$$\sigma_0{}^2 = \frac{1}{\omega_0} \sum_{g=0}^{K} p_I(g) \cdot (g - \mu_0)^2 \qquad \sigma_1{}^2 = \frac{1}{\omega_1} \sum_{g=K+1}^{255} p_I(g) \cdot (g - \mu_1)^2$$

Intra-Klassen Varianz:

$$\sigma_W{}^2 = \omega_0 \cdot \sigma_0{}^2 + \omega_1 \cdot \sigma_1{}^2$$

Optimaler Schwellwert nach Otsu:

$$K_{opt} = \arg\min\{\sigma_w\}$$

Dargestellt mit der Inter-Klassen Varianz:

$$\sigma_B{}^2 = \omega_0 \cdot (\mu_0 - \mu_T)^2 + \omega_1 \cdot (\mu_1 - \mu_T)^2$$

Globaler Mittelwert aller Grauwerte:

$$\mu_T = \sum_{g=0}^{255} p_I(g) \cdot g$$

Da die Summe der Intra-Klassen Varianz und der Inter-Klassen Vrianz konstant ist, kann anstatt σ_W minimiert, σ_B maximiert werden. Dies hat den Vorteil, dass die Inter-Klassen Varianz einfach berechnet werden kann.

$$K_{opt} = \arg\max\{\sigma_B\}$$
$$\sigma_B{}^2 = \omega_0 \cdot \omega_1 \cdot (\mu_0 - \mu_1)^2$$

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

```

```
%get graythreshold (value from 0.0 to 1.0)
Threshold = graythresh(Image);

%make binary image
BW =im2bw(Image,Threshold);
```

Lösung in MATLAB:

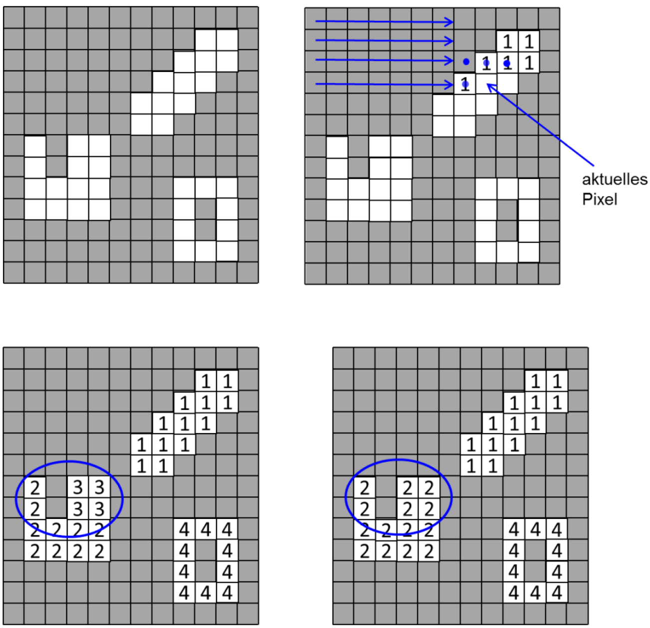
```
function Threshold = OwnOtsu(Image)


%determine image histogram
[Hist, Vals]=imhist(Image);

%normalize it
Hist = Hist/sum(Hist);


%define mean
Mean = Hist.*[0:255]';

%apply Otsu's method
Thresh = [];
%loop over grey values
for Ind = 1:255
    %proability for class 0
    w0 = sum(Hist(1:Ind));
    %proability for class 1
    w1 = sum(Hist(Ind+1:end));
    %mean for class 0
    m0 = sum(Mean(1:Ind))/w0;
    %mean for class 1
    m1 = sum(Mean(Ind+1:end))/w1;
    %store the intra-class probability for this threshold
    Thresh = [Thresh, w0*w1*(m0-m1)^2];
end

%find the maximum value
[MaxThresh, MaxVal]=max(Thresh);
%have to subtract one because we used the index of [0 255]
Threshold = MaxVal-1;
```

5.2 Beschreibung der ROIs

Für die Verarbeitung müssen ROIs (Region of Intress) beschreiben werden, um in einem weiteren Schritt deren Merkmale zu extrahieren.

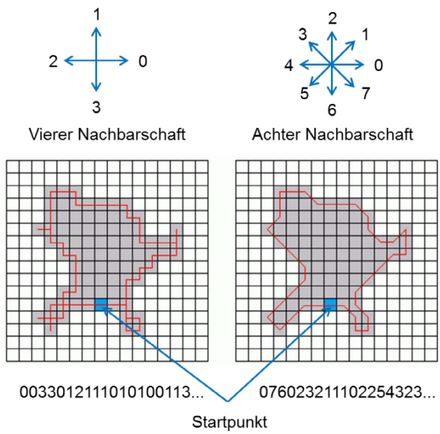
5.2.1 Region Labeling

Beim Region Labeling wird das Bild zeilenweise durch iteriert. Dabei wird jedem Pixel ungleich Null ein Label zugeordnet. Wenn ein Nachbarpixel (Achter-Nachbarschaftsrelation) bereits ein Label enthält, erhält das aktuelle Pixel das gleiche Label, sonst ein neues. Am schluss wird noch überprüft, ob die Label Werte der Nachbarpixel unterschiedlich sind. Falls dies der Fall ist, so wird in einer Lookup Tabelle festgehalten, dass die entsprechenden Label Werte zu einem Objekt gehören. In einem weiteren Durchgang werden die Labelwerte einer Gruppe durch den kleinsten Wert der Gruppe ersetzt.



5.2.2 Kettencode

Da das Region Labeling häufig einen relativ grossen Speicherbedarft hat, werden in der Regel Kettencodes zur Beschreibung von ROIs verwendet.
Die Idee ist, die Randpixel einer ROI, ausgehen von einem definierten Startpunkt, in sukzessiver Folge nur durch die jeweilige Schrittrichtung vom letzten zum aktuellen Randpixel zu definieren.



5.3 Merkmalsextraktion

Basierend auf der Beschreibung der ROIs mittels Region Labels oder Kettencodes können nun einfach verschiedene ROI Merkmale extrahiert werden.

Fläche der ROI:

$$A = \sum_{I_{mn} \in ROI} 1$$

Berechnung anhand von Kettencodes mit dem Crack Code (stellt die Trennlinie zwischen Vorder- und Hintergrund dar):

$$A = -\oint_{Rand} y(x)\mathrm{d}x = \sum_{I_{mn} \in \{2-Seg.\}} m - \sum_{I_{mn} \in \{0-Seg.\}} m$$

Massnemittelpunkt der ROI:

$$x_s = \frac{1}{A} \sum_{I_{mn} \in ROI} n \qquad y_s = \frac{1}{A} \sum_{I_{mn} \in ROI} m$$

Umfang der ROI:

Im Falles einer Crack Code basierten Beschreibung der ROIs ist der Umfang einfach durch die Anzahl der Segmente des Codes geben.

Orientierung der ROI:

Die Orientierung einer ROI ist definiert als der Winkel zwischen der x-Achse und der längeren der beiden Halbachsen der ROI. Dabei sind die Halbachsen bestimmt durch die Eigenvektoren der symmetrischen Matrix bestehend aus den zweiten Momenten M_{xy} der ROI:

$$M = \begin{bmatrix} M_{xx} & M_{xy} \\ M_{xy} & M_{yy} \end{bmatrix}$$

$$M_{xx} = \frac{1}{A} \sum_{I_{mn} \in ROI} n^2 - {x_s}^2 \qquad M_{yy} = \frac{1}{A} \sum_{I_{mn} \in ROI} m^2 - {y_s}^2$$

$$M_{xy} = \frac{1}{A} \sum_{I_{mn} \in ROI} n \cdot m - x_s \cdot y_s$$

Bounding Box

Dies ist das kleinste Rechteck, das die ROI noch ganz umschliesst. Es ist vor allem nützlich, um schnell Tests bezüglich von Einflussregionen durchzuführen.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%do labeling (use 8 neighbors, the default)
[LabelImage, NumberLabels] = bwlabel(Image);

%do feature extraction
Prop = regionprops(LabelImage,'Area','Centroid','Orientation', 'BoundingBox', 'ConvexHull');
%the result is the structure array Prop, with NumLabels x 1 entries

for Ind=1:size(Prop,1)
  %%Area
  Area=Prop(Ind).Area;
  %%Center
    Cent=Prop(Ind).Centroid;
    X=Cent(1);Y=Cent(2);
  %%ConvexHull
  Xv = Prop(Ind).ConvexHull(:,1);
    Yv = Prop(Ind).ConvexHull(:,2);
    line(Xv, Yv, 'LineWidth',1,'Color',[1 0 0]);
  %%Orientation
  Orient = Prop(Ind).Orientation;
  %draw a line through the centroid with given orientation
    Len = sqrt(Prop(Ind).Area);
    Shift = Len*exp(-i*pi*Orient/180);
    line([X-real(Shift) X+real(Shift)], [Y-imag(Shift) Y+imag(Shift)], 'Color', [0 1 0]);
  %%BoundingBox
  BBox = Prop(Ind).BoundingBox;
    %construct the bounding box using line or rectangle
    rectangle('Position', BBox, 'EdgeColor',[0 1 0]);
end
```

Kapitel 6

Linien-Segmentierung und Merkmalsextraktion

6.1 Kantendetektion

Die Bestimmung der Kanten durch Anwendung des Gradienten Filters und einem Schwellwert erzeilt kein zufriedenstellendes Kantenbild. Es treten mehrere Probleme dabei auf:

- Die Kanten sind deutlich breiter als ein Pixel, was eine nicht präzise räumliche Lokalisierung der Kanten bedeutet und zusätzlich eine unnötige Redundanz darstellt.
- Die Wahl eines globalen Schwellwertes ergibt nicht in allen Bildbereichen zufriedenstellende Resultate: Entweder treten Kanten zu häufig auf, oder Kanten werden nicht detektiert. Auch ist die Schwellwertwahl manuell.
- Zusammengehörige Kanten sind, vor allem bei hohem Schwellwert, unterbrochen.

Der Canny Algorithmus zur Kantendetektion löst diese Probleme. Er geht dabei folgendermassen vor:

1. Glättung
Das Bild wird mittels Gauss Filter geglättet (mit parametrierbarer Breit σ)

$$G_{pq}(\sigma) = \frac{1}{2\pi \cdot \sigma^2} \cdot \mathrm{e}^{-\frac{p^2+q^2}{2\cdot\sigma^2}}$$

2. Kantenfilter
Anwendung eines Standard Knatenfilters (Sobel, Prewitt) zur Bestimmung des Betrages und der Richtung des Gradienten in jedem Bildpunkt.

3. Bestimmung der lokalen Maxima
Basierend auf der Richtung des Gradienten wird für jeden Bildpunkt ermittelt, ob es sich um ein lokales Maximum handelt. Hierzu wird die Änderung der Grauwerte entlang der Richtung des Gradienten betrachtet und das Pixel nur dann selektiert, wenn es einen Gradienten Betrag grösser als die Nachbarpixel hat.

4. Kantenextraktion
In einem letzten Schritt werden nun die eigentlichen Kanten selektiert. Hierzu ist ein (manueller) Schwellwert („oberer Schwellwert“) für den Gradienten Betrag zu setzen, oberhalb dessen ein Pixel als Kante betrachtet wird. Wird ein Pixel gefunden, das dem oberen Schwellwertkriterium genügt, werden alle Pixel entlang einer Linie, die dem lokalen Maximum folgt und die einem schwächeren Schwellwertkriterium („unterer Schwellwert“) genügen, zur Kante hinzugefügt. Durch diese Hysterese kann der Canny Algorithmus sehr dynamisch auf Kontrastschwankungen im Bild reagieren.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%upper and lower threshold for edge detection (relative to
    max gradient
%value)
```

```
Threshold = [0.0 0.05];
%width of Gaussian
Sigma = 1;
%apply a certain threshold
[EdgeCanny, Threshold] = edge(Image,'canny', Threshold,
    Sigma);
imshow(EdgeCanny, [0 255]);
```

6.2 Linienextraktion (Hough-Transformation)

Die grundsätzliche Idee der Hough Transformation besteht in der Verwendung der Gradienten Information (Betrag und Richtung), um alle Kantenpunkte vom x-y-Raum in einen neuen Parameterraum zu transformieren, in dem Punkte, die auf einer gemeinsamen Geraden liegen, zum gleichen Parametersatz gehören.
Parametrierung einer Geraden (Hessesche Normalform):

$$\rho = x \cdot \cos\alpha + y \cdot \sin\alpha$$

Die Hough Transformation nutzt die Tatsache aus, dass alle Punkte auf einer gemeinsamen Gerade die gleiche Parameterwerte von α und ρ besitzten.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

% Detect the edges, the result is a binary image
EdgeCanny = edge(Image, 'canny', [0 0.1], 1);

% Hough transformation, calculate the accumulator Hough
[Hough, Alpha, Rho] = hough(EdgeCanny,'RhoResolution',
    2,'Theta',-90:2:89.5);

% Find at most 5 peaks with threshold 15 and minimim
    distance of 15, 15
```

```
% pixel
NumPeaks = 5;
HoughPeaks = houghpeaks(Hough,
    NumPeaks,'Threshold',5,'NHoodSize',[15,15]);

% Find the lines that correspond to the peaks; fill gabs
    of 15 pixel and
% suppress all (merged lines) that have a length less than
    30 pixel
Lines = houghlines(EdgeCanny, Alpha, Rho, HoughPeaks,
    'FillGap', 15, 'MinLength', 50);
```

6.3 Detektion von Kreisen mittels Hough Transformation

Ausgehend von einem Punkt (x, y) auf dem Rand des Kreises kann der Mittelpunkt des Kreises erreicht werden, indem man sich um die Länge R (Radius des Kreises) in die negative Richtung des Gradienten Vektors bewegt.

$$x_c = x - R \cdot \frac{\partial I}{\partial x} \cdot \left[\frac{\mathrm{d}I}{\mathrm{d}r}\right]^{-1} \qquad y_c = y - R \cdot \frac{\partial I}{\partial y} \cdot \left[\frac{\mathrm{d}I}{\mathrm{d}r}\right]^{-1}$$

Den Punkt (x_c, y_c) für alle Werte des Kreises akkumuliert, sollte ein Peak im Mittelpunkt des Kreises resultieren.

Lösung in MATLAB:

```
clear 'all'; close 'all'; format compact;

%read image
Image = imread('sample.png');

%choose the filters
Sobel = 1;
if Sobel == 1
    DX = fspecial('sobel')';
    DY = fspecial('sobel');
else
```

```
    DX = fspecial('prewitt')';
    DY = fspecial('prewitt');
end

%apply the DX and DY filter (use symmetric boundary
    conditions to avoid
%border effects
ImageDx = imfilter(double(Image), DX, 'symmetric');
ImageDy = imfilter(double(Image), DY, 'symmetric');

%calculate the norm of the derviative
ImageDr = sqrt(ImageDx.*ImageDx+ImageDy.*ImageDy);

UseCanny = 1;
if UseCanny == 1
    %apply a certain threshold
    [EdgeCanny, Threshold] = edge(Image,'canny', [0.0
        0.1], 1);

    %we use both the indices and the x-y-values of the
        edges
    Indices = find(EdgeCanny ~= 0);
    [Yw, Xw] = find(EdgeCanny ~= 0);
else
    %we require a threshold; the edges could also be
        chosen with a Canny edge
    %detection in parallel
    Threshold = 0.3*max(max(ImageDr));
    %we use both the indices and the x-y-values of the
        edges
    Indices = find(ImageDr > Threshold);
    [Yw, Xw] = find(ImageDr > Threshold);
end

%this is the accumulator image
Acc = zeros(size(Image));
%here the range of radius' is chosen
for Radius = 20:30
    Dx = ImageDx(Indices);
    Dy = ImageDy(Indices);
    Dr = ImageDr(Indices);
    if Invert == 1
        xc = Xw+Radius*Dx./Dr;
        yc = Yw+Radius*Dy./Dr;
    else
        xc = Xw-Radius*Dx./Dr;
        yc = Yw-Radius*Dy./Dr;
    end
```

```
    %we construct all indices corresponding to a fixed
        radius
    Inds = round(yc)+size(Image,1)*(round(xc)-1);
    %map out values out of the image boundary
    Inds = Inds(0 < Inds & Inds < prod(size(Image)));
    %accumulate
    Acc(Inds) = Acc(Inds)+1;
end
```