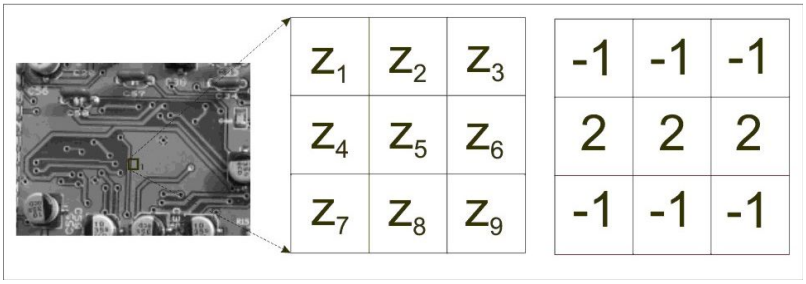
Rand- en lijndetectie

Kristof Teelen

Labo beeldverwerking / 2007-08

1 Lijndetectie

Beeldsegmentatie wordt in dit labo eerst geïllustreerd door de segmentatie van het beeld in lijnen. Horizontale lijnen kunnen gedetecteerd worden door een filtermasker over het beeld te laten lopen. Het masker is zo gekozen dat het reageert op de aanwezigheid van horizontale lijnen. Voor het filtermasker gegeven in figuur 1, verwachten we dat pixels die tot horizontale lijnen behoren een grotere respons op het filter genereren dan andere pixels.



Figuur 1: Berekening van de respons van een filter voor een beeldpixel. Selecteer de 3x3 regio rond een pixel uit het beeld (= centerpixel met intensiteit z_5) en bereken de filterrespons als $\sum_{i=1}^{9} z_i * w_i$, met w_i de gewichten van de filter en waarbij z_5 overeenkomt met de centerpixel. Het resultaat wordt toegewezen aan de centerpixel. Herhaal dit voor alle pixels in het beeld. Om de pixels te vinden waar dit filter een hoge respons genereert, wordt hun waarde vergeleken met een drempelwaarde T.

Je kan een beeld filteren door gebruik te maken van de volgende matlaboperaties:

```
im = rgb2gray(imread('print.jpg'));
h = [-1 -1 -1;2 2 2;-1 -1 -1];
imf = imfilter(im,h);
figure, imshow(im,[])
figure, imshow(imf,[])
figure, imshow(imf > T,[])% Kies een geschikte waarde voor T
```

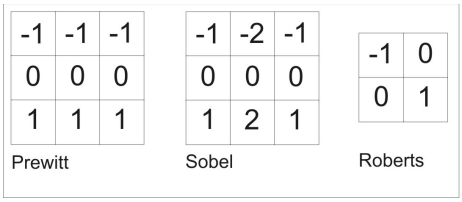
Waar krijg je de hoogste respons in het beeld en waarom? Maak zelf een verticaal en diagonaal filter. Waarom is de som van de coëfficiënten van dit filtermasker gelijk aan nul?

2 Randdetectie

Randen in een beeld worden gekenmerkt door discontinuïteiten in de intensiteit in een beeld. Deze discontinuïteiten kunnen gedetecteerd worden door een eenvoudig filtermasker over het beeld te laten lopen. De gewichten in het masker worden zo gekozen dat de randen in een bepaalde richting een hoge respons geven. Bij voldoende hoge respons wordt een pixel dan geclassificeerd als een randpixel. Enkele detectoren zijn gebaseerd op discrete benaderingen van de eerste afgeleide zoals de randdetectoren van **Roberts**, **Prewitt**

Figuur 2: Grijswaardenbeeld en overeenkomstig randenbeeld

en **Sobel**. De maskers worden met een bepaalde directionaliteit opgesteld: voor horizontale, verticale of diagonale randen. In de onderstaande figuur wordt de horizontale versie van het masker getoond. Door de maskers te draaien kan je ook diagonale en verticale lijnen detecteren. Sobel en Prewitt zijn de meest gebruikte, waarbij Sobel een betere ruisonderdrukking heeft. Door meer gewicht aan de centrale pixel toe te kennen krijg je smoothing. De som van de gewichten in de maskers sommeren tot 0, zodat in gebieden met een constante intensiteit de respons ook 0 is.



Figuur 3: Randenfilters: Roberts, Prewitt en Sobel

De **Log** of **Laplacian-of-Gaussian** filter is een randfilter dat eerst het beeld 'smoother' maakt (m.b.v. de Gaussiaan) en dan de Laplaciaan berekend. Daaruit kan de randinformatie bepaald worden als de nuldoorgang van de Laplaciaan. De **zero-crossings** detector is gebaseerd op hetzelfde concept als de LoG-methode, maar de convolutie gebeurt met een zelf gespecifieerde filterfunctie. Hoe smoother het beeld wordt, hoe minder randen van details er gevonden worden.

De **Canny** edge detector is een van de krachtigste methodes om randen te deteteren, dit is een algoritme dat tevens edge linking uitvoert. De methode werkt als volgt:

1. Convolueer een beeld met een Gaussiaan van schaal sigma.

2. Bereken de lokale grootte (en richting) van de gradiënt voor elke pixel. Hoge waarden duiden op overgangen in intensiteit.

3. Zoek de locatie van randpixels met de hoogste lokale gradiëntmagnitude door lokaal de maxima te bepalen (non-maximal suppression), deze stap zorgt ervoor dat je een rand van 1 pixel breedte bekomt, i.p.v. een band van randpixels.

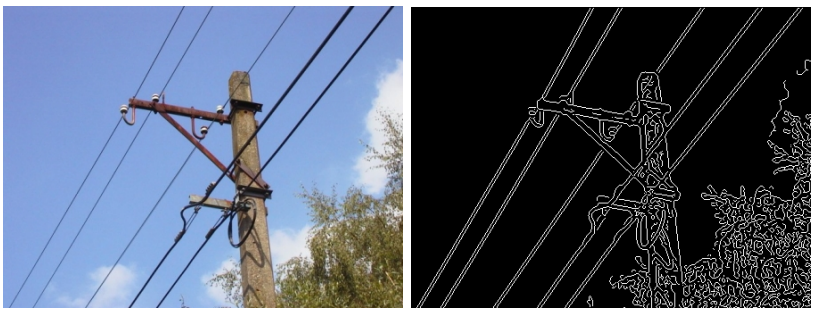
4. De overblijvende randpixels worden dan verdeeld volgens 2 drempelwaardes: $T1$ en $T2$ met $T1 \geq T2$, sterke ($\geq T1$) en zwakke ($\geq T2$ en $\geq T1$) randpixels.

5. Randpixels met een magnitude boven $T1$ worden sowieso als randpixels geclassificeerd. Het linken van edges gebeurt vanuit deze sterke randpixels. Aangrenzende randpixels boven $T2$ worden telkens weer toegevoegd tot de magnitude van de volgende randpixel onder $T2$ zakt. Deze vorm van hysteresis zorgt ervoor dat ruizige randen niet in meerdere delen uiteen vallen.

Je kan zelf het masker bepalen en dan gebruiken om het beeld te filteren met de functie **imfilter**. Voer dit eens uit voor de horizontale en verticale Sobelfilter. Op welke manier verkrijg je dan 1 respons uit de resultaten van een horizontale en verticale Sobelfilter?

Alle bovenstaande randdetectie-algoritmes en filters kunnen geïmplementeerd worden met de matlab-functie **edge**. Hierin kan je ook alle parameters voor de verschillende randdetectiemethodes aanpassen. Kijk eens in de Matlab help hoe deze algoritmes werken en welke parameters meegegeven kunnen worden voor elk algoritme. Probeer de verschillende methodes uit, vergelijk ze met elkaar en zorg dat je goed begrijpt wat de invloed van de verschillende parameters op de werking van de algoritmes is. Gebruik hiervoor het beeld **clown.jpg**.

3 Lijndetectie m.b.v. de Houghtransformatie



Figuur 4: Beeld en overeenkomstig randenbeeld

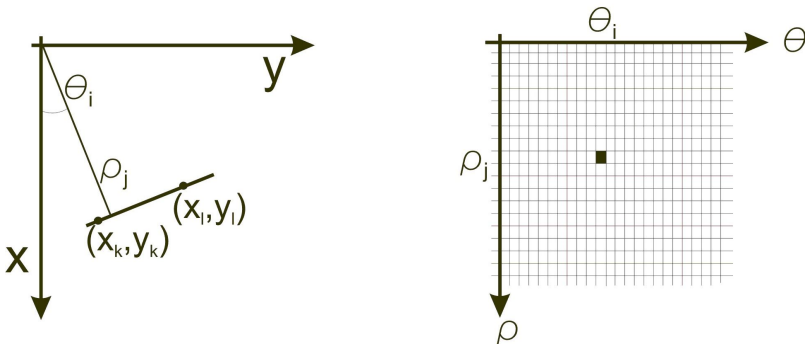
De Houghtransformatie berekent de rechte lijnen in een beeld door te zoeken welke rechte lijnen door de randpuntenpunten van dat beeld lopen. Er kunnen oneindig veel lijnen $y_i = ax_i + b$ door een punt (x_i, y_i) lopen met telkens verschillende waarden voor a en b. In de parameterruimte ab kunnen al deze lijnen voorgesteld worden door verschillende punten (a, b). Als lijn met dezelfde a en b zowel door de punten (x_i, y_i) en (x_j, y_j) loopt, kan je dit uitdrukken door hetzelfde punt in de parameterruimte a, b. Alleen kan je met deze lijnvoorstelling geen verticale rechten voorstellen (want dan is $a = \infty$), en stappen we over naar de representatie $x_i \cos(\theta) + y_i \sin(\theta) = \rho$. In figuur 5 wordt getoond dat θ de hoek is die de loodrechte op de lijn maakt met de x-as en ρ de afstand tot de lijn uitdrukt.

De Houghtransformatie verdeelt de parameterruimte in accumulatorcellen (θ_i, ρ_j) met een zekere grootte. De cellen liggen verdeeld tussen minimale en maximale waarden voor θ en ρ. θ ligt tussen + en -90^o, terwijl ρ kleiner is in absolute waarde dan D, de afstand tot de hoeken van het beeld. De Houghtransformatie gaat nu voor elke mogelijke combinatie (i, j) van θ en ρ alle mogelijke lijnen in het beeld bekijken. Het aantal randpixels dat op deze lijn (voor (θ_i, ρ_j)) ligt in het beeld, wordt opgeteld bij de cel voor die (θ_i, ρ_j) in de parameterruimte. De cellen in de parameterruimte waarvoor een hoge waarde bereikt wordt, zullen dan overeenkomen met rechte lijnen in een beeld.

Matlab voorziet de nodige commando's om de Hough transformatie van een beeld te berekenen: **hough**, **houghpeaks** en **houghlines**. Lees de informatie in de help en probeer de functies uit op het testbeeld **lines.jpg** en bekijk de resultaten:

```
[h,theta,rho] = hough(f);
figure, imshow(mat2gray(h),'XData',theta,'YData',rho);
```

θ wordt uitgezet langs de horizontale as in het getoonde beeld, ρ verticaal. We merken in de *accumulatorruimte* 5 duidelijke punten op, overeenkomstig met de 5 lijnen in het beeld.



Figuur 5: Houghtransformatie: links wordt een rechte lijn in de beeldruimte getoond met parameters (θ_i, ρ_j). Deze lijn gaat door 2 randpixels (x_k, y_k) en (x_l, y_l) zodat de cel (θ_i, ρ_j) in de accumulatorruimte rechts met 2 verhoogd wordt.

Herhaal de bewerking met het beeld **wires.jpg**. Detecteer nu wel eerst de randen want de Hough transformatie vereist een beeld met randen als input. We kunnen nu met de functie **houghpeaks** de cordinaten van de N grootste pieken berekenen:

```
peaks = houghpeaks(h,5);
```

De functie houghlines berekent de lijnsegmenten:

```
lines = houghlines(f, theta, rho, peaks);
```

We plotten de lijnen bovenop de originele figuur met:

```
figure, imshow(f), hold on,
for k=1:length(lines);
xy=[lines(k).point1 ; lines(k).point2];
plot(xy(:,1), xy(:,2));
end;
```

We merken op dat alle gevonden rechten inderdaad overeen komen met lijnen die werkelijk in de figuur aanwezig zijn. Toch blijven er enkele problemen met deze methode:

- Als je veel lijnen opvraagd, gebeuren er ook valse positieve detecties: het algoritme detecteert lijnstukken die niet overeenkomen met de werkelijke lijnen in de figuur.
- Korte lijnen in het beeld worden moeilijk opgemerkt.
- In een digitaal beeld van een reële scène zijn de lijnen meestal niet helemaal recht, wat de detectie moeilijker maakt.
- Digitale lijnen bestaan uit afzonderlijke pixels, wat heel anders is dan een lijn uit de normale geometrie. Lijnen onder een kleine hoek met de hoofdrichtingen van het beeld zullen moeilijk als 1 lijn gedetecteerd kunnen worden met deze matlabimplementatie (zie het effect van **houghlines** op **lines.jpg**).