Edge Detection

Jeroen Kok
Stefan
23-02-2017

1. Doel

In opdracht van de docent moesten wij een onderdeel kiezen uit het Face Recognition programma, daar onderzoek naar doen en dat dan verbeteren.
Wij hebben gekozen voor Edge-detection omdat wij denken daar het meeste verschil in te kunnen maken. Wij denken dat de edge-detection sneller kan zijn dan het nu is.

Ons doel is de huidige implementatie voor de Edge-detection te vervangen met een algoritme dat sneller is dan degene die nu wordt gebruikt. De betrouwbaarheid van het systeem mag echter niet te veel omlaag gaan. Hiervoor gaan wij onderzoek doen naar een aantal methodes om te kijken welke de beste verbetering in het systeem zal brengen.

Wij zullen daarna een aantal methodes uitkiezen die wij vervolgens grondig zullen testen op basis van verschillende attributen, zoals de tijd die het kost en betrouwbaarheid, om te kijken welke de beste is.

2. Methoden

Voor dit onderzoek hebben wij zoveel mogelijk methodes proberen te vinden en hebben wij kort gekeken waar welke goed voor is. De methode die wij hebben gevonden zijn:

- Canny
- Deriche
- Basic Gradient Detection (BGD)
- Robert Cross
- Laplacian
- Sobel
- Prewitt
- Mexican Hat
- Hough Transformatie

Canny & Deriche

Canny is een methode die veel wordt gebruikt. Maar het is niet wat wij zoeken; het is meer een stappenplan dat als kernel verwijst naar Sobel, Prewiit, ect. Het is zelf geen edge-detection, maar een systeem voor ‘goede’ edge-detection. Ditzelfde geldt voor Deriche.

BGD, Roberts Cross en Laplacian

Deze methode zijn kernels waar in Canny en Deriche naar worden verwezen. Als je een van deze kernels over een plaatje haalt, dan krijg je een plaatje terug met alle edges. Hoewel ze allemaal kernels zijn; gaat het om de getallen die in de kernels staan en hoe groot ze zijn. En die leveren elk een ander resultaat. Als een kernel groter is zal deze trager werken maar het resultaat zal niet per se slechter of beter zijn, Daarom lijkt het ons slim om meer op de kleinere kernels, zoals Basic Gradient Detection en Roberts te focussen.
Hoewel elke methode erg op elkaar lijkt, zijn er subtiele verschillen die toch het resultaat wezenlijk kunnen veranderen.

De kleinste en simpelste is BGD

1	-1

(In verband met overzicht heb ik besloten om alleen de kernel van één richting te geven. De andere 3 richtingen zijn identiek en over het algemeen vanzelfsprekend.)

Door zijn grootte zijn er per plaatje de minst mogelijke berekeningen nodig. Namelijk 2* het aantal pixels - per richting van edge-detection.
Deze kernel vertelt je in essentie hoe sterk het gradient veranderd op die pixel.
Het nadeel van BGD is dat het de edge tussen de 2 pixels ‘weergeeft’. Dit kan echter niet, je kunt niet zomaar een pixel ertussen verzinnen. Daarom bestaat er een variant op BGD:

1	0	-1

Met de 0 op de originele pixel. Hierdoor wordt de ‘edge’ op de middelste pixel gegeven. Máár kost elke pixel wel 1 berekening meer. Hoewel de kans bestaat dat dit verwaarloosbaar klein is, zijn wij op zoek naar de hoogste snelheid, dus het is iets wat wij in gedachten willen houden.

Een andere variant op BGD is Roberts Cross:

1	0
0	-1

Er is verder niks speciaals aan deze kernel, behalve dat hij dubbel zo groot is als BGD en dat deze diagonaal werkt.

Een andere manier om edges te vinden is de Laplacian edge-detection:

-1	2	-1

Bij de BGD kijk je naar het verschil in gradiënt, ook wel de eerste afgeleide. Bij Laplacian kijk je naar de 2e afgeleide. Bij BGD zijn de witte vlekken de edges, maar bij Laplacian zal er een donkere en een witte streep ontstaan. Deze 2 strepen samen geven de edge aan in de richting van donker naar licht.

Het voordeel van Laplacian is dat het veel details bewaard, maar de zwakkere edges niet goed zijn te vinden. Bij BGD is dit omgekeerd; de zwakkere edges zijn duidelijker, maar er gaat veel detail werk verloren.

Prewitt, Sobel en de Mexican Hat

Zowel Laplacian als BGD en RC hebben één groot probleem; Ruis. Mocht een van de pixels die gebruikt worden een abnormale waarde hebben, dan zal het resultaat ook abnormaal zijn. Daarom zijn er mensen gekomen die daarop iets hebben

bedacht; je kunt namelijk ook een vorm van ruisfiltering toepassen. Hoewel ruis minder sterk is, zullen zwakkere edges ook wegvallen.

De Prewitt kernel is de 2e variant van BGD maar dan naar boven en beneden uitgebreid wat voor ruisvermindering zorgt:

1	0	-1
1	0	-1
1	0	-1

Het voordeel is dat abnormale pixels minder invloed hebben. Maar er treed ook een algemeen verlies van detail op. Sobel is weer een variant van Prewitt, die de nadruk legt op de pixels in het midden van de kernel en daardoor minder details kwijt raakt:

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

Sobel zegt dat de pixels horizontaal belangrijker zijn dan die diagonaal. Hierdoor gaat er minder detail verloren dan met Prewitt, maar minder ruisgevoelig dan BGD.

Voor Laplacian is er ook een variant met ruisfiltering. Ook bekend als de Mexican hat vanwege zijn vorm, bevat deze een gaussian filter óver de laplacian edge-detection heen:

1	-4	1
-4	12	-4
1	-4	1

De Mexican Hat heeft echter een nadeel; de waardes in de kernel kunnen wat extreem worden naarmate de kernel kleiner wordt. Dit komt simpelweg door aliasing en de enige oplossing in deze situatie is door een grotere kernel te gebruiken.
In ons geval - wij zoeken naar snelheid - zijn grotere kernels echter kostbaar. Niet alleen is deze kernel groter; hij is ook ondeelbaar. Je hoeft hem maar één keer over een plaatje te halen, maar elke pixel moet 9 keer berekend worden.

Hough Transformation

De Hough transformatie is een honourable mention. Hoewel dit een effectieve vorm van edge-detection kan zijn, is dit een erg complexe manier van edge-detecten. Naast dat de transformatie zelf ingewikkeld en duur is, levert het een plaatje op dat onbruikbaar is door het aan ons gegeven programma.
Door de ingewikkeldheid hebben wij besloten deze wel te noemen, maar hebben wij simpelweg niet de tijd om er goed onderzoek naar te doen.

3. Keuze

Omdat wij voor een zo snelle mogelijke methode willen, prioriteren wij voor de methodes met een kleinere kernel. tevens mag het aantal false-positives en false-negatives niet te veel stijgen.

Daarom hebben wij gekozen voor BGD, sobel en laplacian.

BGD heeft de kleinste kernel, maar is erg ruisgevoelig. Prewitt en Sobel hebben ingebouwde ruis vermindering dus deze zullen beter bestand zijn tegen ruis en dus minder false-positives en false-negatives hebben. Echter hebben Sobel en Prewitt wel een grotere kernel dan BGD een grotere kernel, maar is beter bestand tegen ruis dan BGD. Laplacian huppelepup.

Hough vonden wij te ingewikkeld en ziet er erg traag uit. Mexican Hat is

hierom kiezen wij voor Basic Gradient Detection (BGD) gekozen. Omdat wij BGD erg simpel vinden, denken wij dat dit de beste methode voor ons doel is. Door de simpliciteit denken wij dat het een snellere methode zal zijn dan bijvoorbeeld Robert Cross, Prewitt of Sobel.

Maar omdat het hier om enkel de inhoud van de kernels gaat, willen wij ook Laplacian en Prewitt testen om te zien of de grootte van de kernels verschil maakt in de snelheid en kwaliteit van het resultaat.

Wij kijken in eerste instantie niet naar Canny of Deriche omdat zij zelf geen edge detection zijn, maar kernels van anderen gebruiken.

De hough transformatie lijken ons een te complexe methoden om uit te zoeken in verband met de beperkte tijd die wij voor dit project hebben.

4. Implementatie

Wij zullen alleen de BGD, Sobel en Laplacian methodes gaan implementeren. De implementatie van de edge-detection methoden zal gebeuren in de al bestaande software in het bestand: StudentPreProcessor. Elke edge-detection methode zal zijn eigen functie krijgen die wordt aangeroepen binnen de functie: stepEdgeDetection.

Bij het detecteren van edges zullen wij het thresholding gedeelte niet implementeren maar alleen het detecteren van de edges.

Voor alle drie de methodes moet 2 keer over het plaatje worden gegaan omdat de kernels voor deze methode maar voor één richting is. Eerst zal de kernel voor de horizontale richting op elke pixel worden toegepast daarna zal de kernel voor de verticale richting op dezelfde manier worden toegepast. Als dit gebeurd is zal het resultaat van de verticale kernel en het resultaat van de horizontale kernel worden samengevoegd tot 1 afbeelding met hierin alle mogelijke edges.

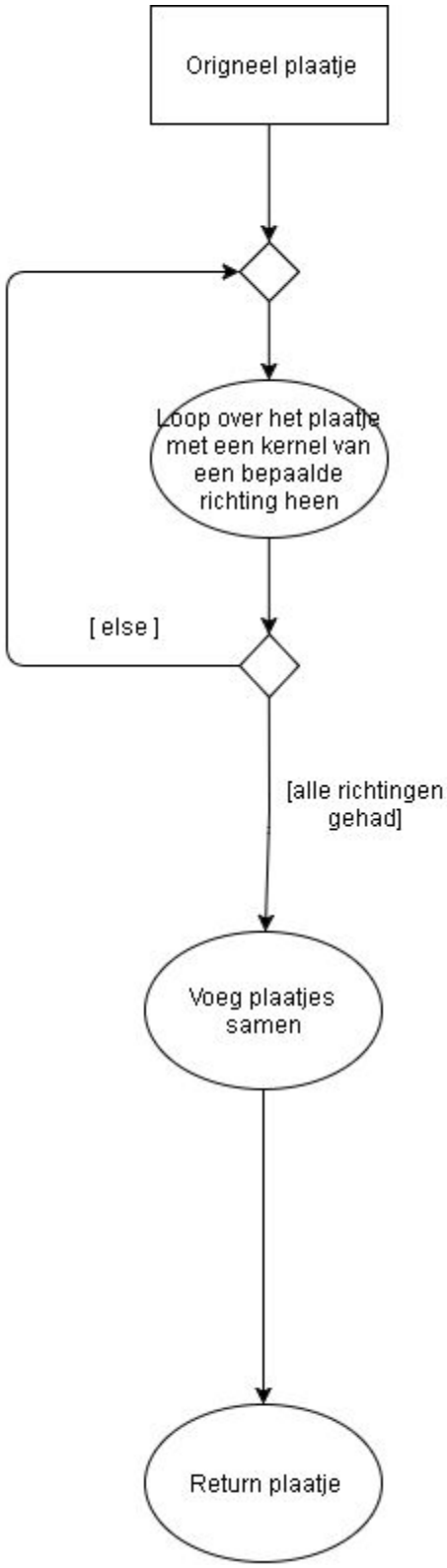
Een horizontale kernel en een verticale kernel voor edge-detection kunnen niet worden samengevoegd omdat dit de edge-detecterende eigenschappen van de kernel zal opheffen.

Het kan gebeuren dat pixels die buiten het plaatje liggen benodigd zijn, bijvoorbeeld als een kernel op de allereerste pixel(linksboven) van het plaatje word toegepast dan valt een deel van de kernel buiten het plaatje. Om dit op te lossen zijn meerdere methodes:

- De afbeelding in horizontale en verticale richting uitrekken.
- De pixel die buiten de afbeelding dezelfde kleur geven als de pixels in kernel die wel binnen de afbeelding vallen.
- De pixels die buiten de afbeelding vallen niet meenemen in de edge detection. De afbeelding zal dus kleiner worden.

Wij zullen de benodigde pixels die buiten de afbeelding liggen weghalen. Wij hebben voor deze methode gekozen omdat de kernels van onze edge-detection methodes maximaal maar 3 pixels bij 3 pixels groot zijn, hierdoor zal alleen de buitenste rij pixels worden weggelaten. deze methode kost minder handelingen en zal daarom simpeler om toe te passen zijn dan de andere methodes.

Wij nemen aan de het implementeren van de edge-detection methode niet erg ingewikkeld zal zijn en dat het binnen een week kan gebeuren. De methode die ons het lastigst lijkt en ook het meeste werk lijkt ons Sobel omdat deze de grootste kernel heeft, Al lijkt Sobel nog steeds erg op de andere methodes.


Origneel plaatje
Loop over het plaatje met een kernel van een bepaalde richting heen
[else]
[alle richtingen gehad]
Voeg plaatjes samen
Return plaatje

5. Evaluatie

Doel:

Met dit onderzoek willen wij testen of BGD het meest efficiënte edge-detection methode is.

Waarom is dit belangrijk:

Object herkenning begint steeds meer toegepast te worden in systemen. Deze systemen moeten bijvoorbeeld real-time veel gezichten van mensen herkennen. Om dit mogelijk te maken moet het systeem snel een beeld kunnen bewerken. Omdat edge-detection een belangrijk gedeelte hiervan is willen wij testen of het edge-detection sneller kan.

Testen:

Wij gaan dit testen op 2 manieren. De eerste test is snelheid en de 2de test zal betrouwbaarheid zijn.

Om de snelheid te meten zullen wij de BGD methode met de methode sobel en laplacian gaan vergelijken. Elke methode zal meerdere keren getimed worden op verschillende foto's. Per foto zal er een beste methode zijn en aan het einde zullen wij kijken welke methode het vaakst gewonnen heeft.

Om de betrouwbaarheid te testen zullen wij letten op hoeveel edges de BGD vergeleken met de andere methodes genereert, daarmee letten wij vooral op de false negatives. Dus de edges die er wel zijn maar de methode niet herkent heeft. wij zullen de methodes op verschillende foto's testen en daarna zullen wij de uitkomsten vergelijken. De uitkomst zal beoordeeld worden naarmate welke edges de methode in vergelijking met de andere methodes gevonden heeft.

Verwachtingen:

Wij verwachten dat BGD de snelste methode zal worden omdat het e kleinste kernel heeft. We denken BGD mee de meeste false-positives zal vinden omdat deze methode geen ruisvermindering heeft.

} Doel : Kijken of BGD het efficienste kernel is
} Aantoenen dat BGD het snelste kernel is, zonder te slechte resultaten te leveren.
} Relevantie: In de praktijk wil je veel verschillende gezichten herkennen in zo min mogelijk tijd
} Wat meten: Hoelang doet het programma er over met de verschillende kernels.
} Hoe: dat stukje code timen
} Wij denken dat BGD sneller zal werken dan de andere kernels.Q. Maar niet zoveel dat het de gezichtsherkenning zal vermoeilijken.

Random stuff to get my point

Bronnen

1. http://users.polytech.unice.fr/~lingrand/Ens/up/Lesson7and8-segmentation.pdf
2. https://www.researchgate.net/publication/239398674_An_Isotropic_3_3_Image_Gradient_Operator
3. https://pdfs.semanticscholar.org/55e6/6333402df1a75664260501522800cf3d26b9.pdf
4. http://sites.google.com/site/setiawanhadi2/1CannyEdgeDetectionTutorial.pdf
5. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.1860&rep=rep1&type=pdf
6. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.1860&rep=rep1&type=pdf
7. http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm
8. https://ena.etsmtl.ca/pluginfile.php/59679/mod_resource/content/0/Deriche%20Wikipedia.pdf
9. http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html
10. http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm
11. https://www.cis.rit.edu/people/faculty/rhody/EdgeDetection.htm