Implementatieplan edge detection

Door Renske Kuip en Olivier Verwoerd

Vision - Arno Kamphuis
april 2018

Inhoud

Doel **3**

Methoden **3**

Canny 3

Prewitt operator 4

Sobel operator 4

Robert's cross operator 5

Laplacian 5

Laplacian of Gaussian 6

Voor- en nadelen 6

Thresholding 7

Keuze **7**

Implementatie **8**

Edge detection 8

Thresholding 9

Evaluatie **10**

Bronnen **11**

Doel

Het doel van het project is om een bruikbare edge detection methode, gevolgd door thresholding, te implementeren zodat het resultaat kan worden gebruikt door de gegeven gezichtsherkenningssoftware. De huidige software zet een kleuren afbeelding om naar een grijswaardenafbeelding, hier moeten nog bewerkingen worden uitgevoerd om de lijnen te vinden. Hiervoor dient de edge detection methode, deze accentueert de opvallende overgangen van kleur/intensiteit in de afbeelding waarna vervolgens de thresholding deze omzet naar een zwart-wit afbeelding.

Er zijn voor zowel de edge detection als thresholding meerdere methoden die we kunnen implementeren. We zullen deze moeten afwegen en met elkaar vergelijken zodat we een keuze kunnen maken.

We hebben dit doel omgezet naar een hoofdvraag, deze luidt: “Welke combinatie van edge detection en thresholding methode zal voor gezichtsherkenning het beste werken?” Door middel van literatuurstudie en vergelijking van het gevonden materiaal wordt deze vraag beantwoordt. In een meetrapport zullen de resultaten van gekozen methoden worden beschreven.

Methoden

Edge detection methoden zijn op te delen in twee categorieën. Dit zijn search-based en zero-based. Search-based methoden (kernels) kijken naar de eerste afgeleide, ofwel naar de snelheid van de verandering van de intensiteit van de pixel en omliggende pixels. Zero-based kijkt naar de tweede afgeleide, ofwel de afgeleide van de afgeleide van een functie. Voor allebei deze categorieën zijn een aantal methoden die worden toegelicht. Dit zijn voor search-based:

- Canny
- Sobel
- Prewitt
- Robert’s cross

Voor zero-based:

- Standaard laplacian
- Laplacian of Gaussian

Canny

Canny’s methode bestaat uit een aantal vaste stappen die gemaakt worden. De methode houdt zowel edge detection als thresholding in.

Je haalt als eerste een Gaussian filter over je afbeelding. Deze maakt de afbeelding iets gladder waardoor de ruis verminderd. Dit filter maakt gebruik van een kernel ter grootte van

(2*k*+1)×(2*k*+1). Vaak is een 5x5 filter genoeg. Hoe groter de Gaussian kernel, hoe minder de performance van de detector.

Vervolgens ga je op zoek naar edges en elke kant ze op gaan. Het Canny algoritme gebruikt vier filters om horizontaal, vertikaal en diagonale edges te vinden.

Hierna verdun je de edges door middel van *non-maximum* suppressie. Hierbij bekijk je van elke pixel naar de waardes van de omliggende pixels in de y-richting. Als de huidige pixel een hogere edge-waarde heeft dan de omliggende, zal de pixel worden vergeleken met de omliggende in de verticale as en zal de waarde hetzelfde blijven. Anders zal hij worden ‘suppressed’.

Om de laatste ruis weg te halen worden edge pixels met een lage gradatie weggehaald. Er worden twee threshold waardes aangemaakt en de pixels worden vergeleken met deze waardes. Als de gradatie waarde van de pixel hoger is dan de hoge threshold wordt deze gezien als een sterke edge pixel en is hij lager dan de lage threshold waarde. Dan zal hij worden gezien als zwakke edge pixel en zal worden ‘suppressed’.

Prewitt operator

De Prewitt operator bestaat uit de twee onderstaande 3x3 kernels.

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

Deze kernels zijn gemaakt om horizontale en verticale edges te herkennen. Je berekent allebei de kernels apart van elkaar voor elke pixel in de afbeelding en telt de resultaten bij elkaar op. Op deze manier vindt je de edges in beide richtingen optimaal. Er wordt echter gebruik gemaakt van de constante waarde (1), waardoor de methode vrij foutgevoelig is.

Sobel operator

Sobel’s operator is in feite hetzelfde als de Prewitt operator alleen met andere waarden.

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

Gx

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

Gy

Er wordt meer nadruk gelegd op de pixels in het midden, waardoor je met meer precisie edges kunt vinden. Dit is een van de meest gebruikte edge detection methoden. Net als Prewitt is het een eenvoudige methode om te implementeren.

Robert's cross operator

De Robert's Cross operator maakt gebruik van spatiale gradatie meting op een afbeelding. Er wordt gebruik gemaakt van twee 2x2 kernels die in een kruisvorm lopen. Het is een vergelijkbare methode als de Sobel en Prewitt operators, maar deze methode werkt het best voor schuine edges.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

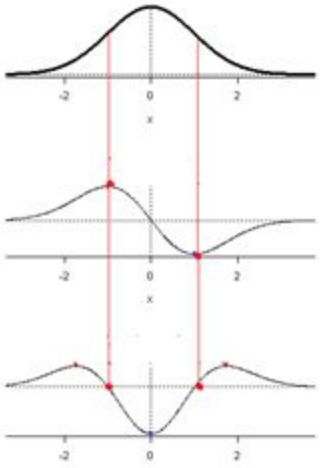
Je past de kernels weer apart toe en voegt de resultaten van de metingen bij elkaar. Hieruit kun je de absolute waarde van de gradatie en de oriëntatie.

Laplacian

Bij de Laplacian methode is aangeraden ook eerst een Gaussian filter gehaald om hem minder gevoelig te maken voor ruis, want de kernels die worden gebruikt voor deze methode zijn hier zeer gevoelig voor. Hierdoor krijg je net als voor andere methodes dat je minstens twee keer over een afbeelding moet; één keer voor de smoothing en (minstens) één keer voor de Laplacian uitvoering. Het ligt eraan of je kiest voor twee 1D kernels (één voor de x en één voor de y richting) of één 2D kernel. De 2D kernel is het meest voorkomend en de basis ziet er als volgt uit:

0	1	0
1	−4	1
0	1	0

Normaal gesproken wordt de Laplacian methode toegepast op een single gray level afbeelding. Het filter (of meervoud indien je met 1D kernels werkt) gaat op zoek naar de tweede afgeleide van de verandering tussen de pixels. Een lijn, of edge, heeft in feite twee punten waar verandering in intensiteit zit. In de afbeelding hieronder is dit afgebeeld.



De piek in de bovenste grafiek is de daadwerkelijke edge, bijv een lijn van 1 pixel breed. In de tweede grafiek vind je twee pieken waar de verandering van pixel intensiteit is en hoe snel deze verandering daalt of stijgt. We willen deze twee punten weten, want hier zitten de randen van de edge. Door nog een tweede keer de convolutie uit te voeren, vindt je de tweede afgeleide (de derde grafiek) waar de twee pieken aan de randen van de edge zitten en is het 0-punt de edge (vandaar de naam *zero-based*). Op deze manier kun je preciezer de edges vinden.

Laplacian of Gaussian

De (2D) Laplacian methode staat weliswaar uit slechts één kernel, maar door de smoothing van de Gaussian heb je alsnog extra operaties die moeten worden uitgevoerd. Hiervoor is een oplossing bedacht: de Laplacian of Gaussian (LoG) methode. Dit is in feite een combinatie van de beide kernels in één kernel voor het vinden van de tweede afgeleide en tegelijkertijd smoothen. Dit maakt het een efficiëntere methode m.b.t. berekeningen en verhelpt de ruisgevoeligheid van de Laplacian. De kernel voor deze methode is te vinden door de Laplacian en Gaussian kernels met elkaar te combineren, waardoor een ontstaat. Naarmate de sigma van deze verdeling (en de grootte van de kernel) worden de waarden van de kernel bepaald.

Voor- en nadelen

Voordeel van Canny edge detectie is dat het een vrij precieze methode is. Nadeel is echter dat bij het ‘verzachten’ met de Gaussian filter je ook de edges minder duidelijk maakt waardoor ze moeilijker te vinden zijn. Er is een grotere kans dat je zwakke edges mist. De Canny methode is ook relatief zwaarder qua hoeveelheid berekeningen dan de andere methodes.

De search-based algoritmes als Prewitt hebben gemeenschappelijke nadelen. Zo is de grootte van kernel vastgesteld en kun je deze niet aanpassen aan de grootte van de afbeelding. Ze zijn ook minder precies dan de zero-based methodes. Wel zijn ze minder gevoelig voor ruis dan Laplacian. Ze kunnen door de twee aparte kernels makkelijker de richting van een edge vinden, al is Robert’s Cross minder goed voor de toepassing van horizontale en verticale edges.

Voordeel van 2D Laplacian is dat je maar één berekening nodig hebt doordat de kernel wordt samengevoegd. Hierdoor is de methode sneller vanwege de weinige rekenkundige operaties. Een nadeel is echter dat de richting van de edges minder goed te vinden zijn, omdat je maar één kernel gebruikt en niet zowel een horizontale en verticale kernel. Ook is het zoals eerdergenoemd enorm gevoelig voor ruis. Hier komt het voordeel van de LoG methode naar voren. Hoewel je bij Laplacian van tevoren de Gaussian filter kunt toepassen, zorgt dat voor meer bewerkingen. Door de combinatie met de Gaussian filter is de LoG kernel op zichzelf minder gevoelig voor ruis én hij heeft minder operaties om uit te voeren.

Thresholding

De huidige methode voor de thresholding is door een vaste waarde te nemen en alle pixels boven of onder deze waarde op de voor- of achtergrond te plaatsen. Dit is snel, maar erg foutgevoelig. Voor betere resultaten kunnen we de thresholding dynamisch maken. Het dynamisch maken van de thresholding vereist eerst te weten welke waardes op welke afbeelding werken. Door alle waardes af te lopen komen daar een lijst met goede threshold waardes uit. Door deze te gaan vergelijken en verbanden maken met de minimale, maximale en gemiddelde waarde van de afbeelding kan hiermee een formule worden gemaakt die dynamisch de beste antwoorden zal geven.

Keuze

Wij hebben gekozen om de zero-based methodes toe te passen, omdat deze naar verwachting de meeste precisie bieden. De search-based methodes bieden minder precisie en vergen meer berekeningen. Het grootste nadeel van deze methodes is de gevoeligheid voor ruis, maar met de Gaussian filter (zij het als aparte kernel of verwerkt binnen de LoG) is dit goed te compenseren. We hebben besloten om zowel Laplacian met aparte Gaussian filter als een LoG filter te implementeren, zodat we de resultaten naast elkaar kunnen leggen om te vergelijken. Hierdoor kan worden gezien of er daadwerkelijk verschil in de methodes is.

Dit willen we samenvoegen met een dynamische thresholding. Een statische waarde voor de thresholding is weliswaar sneller, maar hiermee verwachten we minder goede resultaten. Een waarde die wordt berekend zorgt voor betere threshold afbeeldingen.

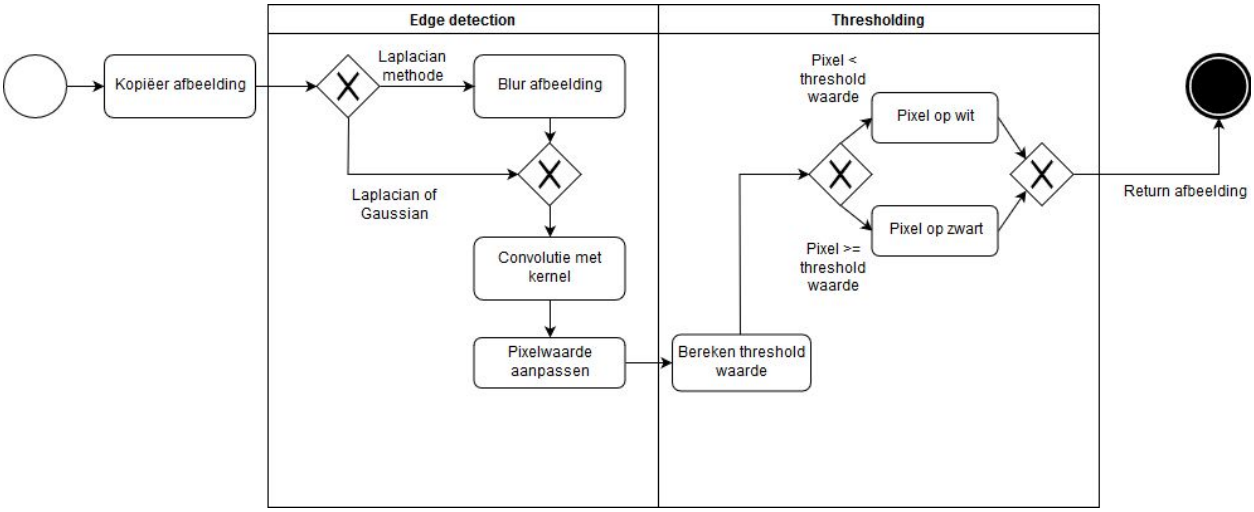
Implementatie

De gezichtsherkenning software waarmee gewerkt wordt is geschreven in C++. In de Visual Studio 2015 omgeving zullen de edge detection en thresholding worden geïmplementeerd. De onderdelen vallen binnen de pre-processing omgeving van de software.

Edge detection

De edge detection filter zal beginnen met een intensiteit afbeelding (grayscale). Omdat we geen bewerkingen over elkaar willen gebruiken, beginnen we met een blanco afbeelding in de exacte grootte als het origineel.

We willen zoals eerder beschreven gebruik maken van de Laplacian methode met Gaussian filter en de Laplacian of Gaussian filter. Deze methoden willen we implementeren zoals hiervoor besproken. In de activity diagram is globaal te zien hoe de code eruit zal zien, met daarin de edge detection en thresholding als lanes afgebeeld. De *exclusive gateway* binnen de edge



detection geeft de twee verschillende methodes weer.
Per pixel vindt er een convolutie plaats met het desbetreffende kernel en het resultaat wordt in een kopie van de afbeelding geplaatst. De kopie van de afbeelding is belangrijk, anders zullen de nieuwe waardes worden gebruikt in de volgende convolutie van de pixels ernaast. Je wilt juist de originele waarden van de pixels gebruiken bij de bewerkingen.

Voor de Gaussian filter wordt gebruik gemaakt van een 5x5 kernel. Er is gekozen voor een relatief kleine filter, omdat er op deze manier niet te veel details verloren gaan maar wel degelijk ruis wordt verminderd. Hieronder een mogelijk 5x5 Gaussian kernel die gebruikt zal worden. Er is een mogelijkheid dat deze waarden niet optimaal zijn en er tijdens de implementatie gekozen wordt voor andere waarden.

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 7 & 11 & 7 & 2 \\ 3 & 11 & 17 & 11 & 3 \\ 2 & 7 & 11 & 7 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Voor de Laplacian wordt de 2D kernel gebruikt. Om ook brede edges te kunnen detecteren wordt dit een 5x5 kernel. Het lay-out zal vergelijkbaar zijn als de volgende afbeelding.

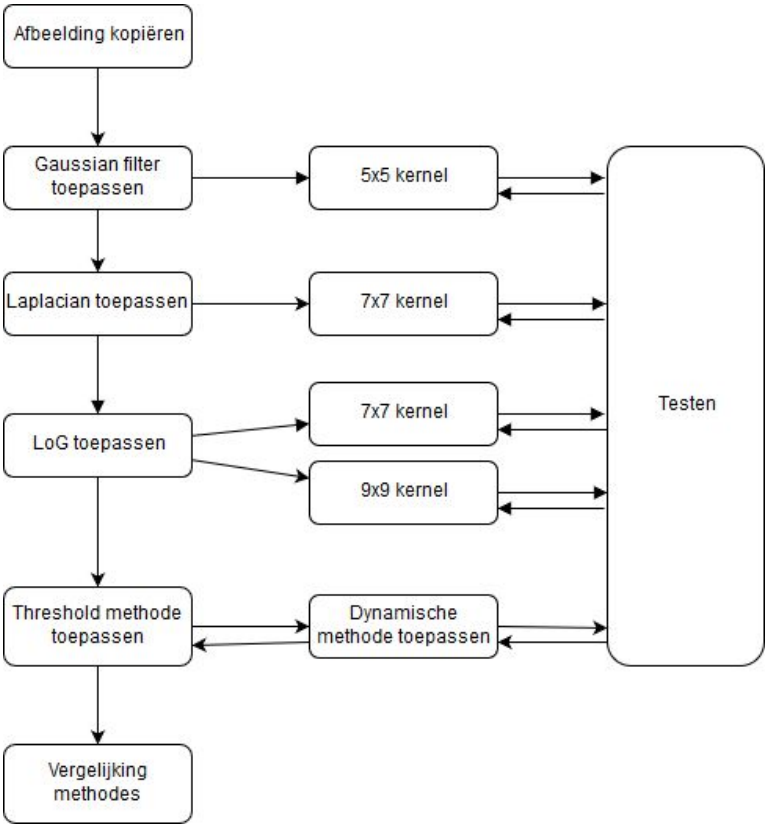
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

De LoG kernel zal berekend moeten worden aan de hand van de sigma. Twee veel gebruikte sigma's zijn σ = 1.05 en σ = 1.4. Omdat er veel verschil in resultaat kan zijn, zal er naar twee verschillende kernels worden gekeken. Op deze manier hopen we het beste resultaat te vinden.

Het resultaat van de edge detection is een grayscale afbeelding waarin de donkere pixels duidelijk de edges weergeven. Deze afbeelding is echter nog niet bruikbaar voor de gezichtsherkenning, omdat er nog veel huis aanwezig is. Om deze reden is er ook nog een thresholding nodig.

Thresholding

De threshold bewerkingen gelden per pixel. Er is geen extra afbeelding nodig omdat de bewerkingen dan geen overlap door middel van kernels hebben.
De functie thresholding krijgt een afbeelding mee (waar edge detection is op toegepast) die wordt omgezet naar enkel zwart en wit. Per pixel wordt er gekeken of de waarde in het bereik 0 tot 255 ligt. De threshold waarde wordt bepaald door een steekproef van 100 pixels. Van deze pixels wordt de minimale, maximale en gemiddelde waarde van de pixel intensiteit bepaald. Op basis van deze gegevens wordt de threshold waarde berekend. Als de pixel waarde minder is dan de threshold waarde zal de waarde van de pixel op minimaal worden gezet (wit). Als de pixelwaarde hetzelfde is of meer dan de threshold waarde, wordt deze op maximaal (zwart) gezet.
We hebben een indeling gemaakt van de volgorde waarin alles geïmplementeerd zal worden. Dit is de zien in onderstaande diagram. Op deze manier kan goed worden gecontroleerd of alles aanwezig is en blijft het overzichtelijk. De stappen zullen tussen de teamleden worden verdeeld. Na elke stap zal opnieuw worden getest om bugs vroegtijdig te detecteren en te kunnen verhelpen. Na deze stappen hopen we een goed eindresultaat te hebben en de nieuwe afbeelding terug te kunnen geven aan de functie, zodat de gezichtsherkenning hiermee kan werken.



Evaluatie

Wat we willen aantonen in onze verdere meetrapporten, is dat de LoG methode een efficiënte en goede manier is om edge detection uit te voeren. We verwachten dat hij een stuk sneller is dan de reguliere Laplacian methode zoals beschreven eerder in dit document. Naar verwachting zullen weliswaar de resultaten van de detectie zelf niet ver naast elkaar liggen, omdat het vergelijkbare operaties zijn, maar de performance zal weldegelijk variëren.

Voor de thresholding geldt zal de dynamische methode zeker verschil uitmaken. Zowel in snelheid, die achteruit zal gaan, maar met zeker betere resultaten.

Om daadwerkelijk te bevestigen of de edge detection en thresholding correct is geïmplementeerd, zullen deze functies voldoende moeten worden getest. Er zal onder meer goed getest moeten worden op snelheid en succes rate van de gezichtsherkenning. Hierdoor kunnen we realistisch meten of er daadwerkelijk verschil zit in de methoden.

Bronnen

Vision timer:
https://github.com/arnokamphuis/vision-timer/blob/master/Timer/Release/Timer.exe
https://www.mathworks.com/discovery/edge-detection.html
https://en.wikipedia.org/wiki/Canny_edge_detector
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf
https://www.omicsonline.org/open-access/methods-of-image-edge-detection-a-review-2332-0796-1000150.php?aid=57249
https://softwarebydefault.com/tag/laplacian-of-gaussian/
http://haishibai.blogspot.nl/2009/10/image-processing-c-tutorial-5-log.html