Auteur : Jalu Glissenaar, Thomas Smeele
Datum : 1 April 2019
Project : Vision
Team : Kaasbroodje

Randdetectie & Thresholding Implementatieplan

1. Doel

Ten eerste, het doel van deze implementatie is bedoeld als een project waar wij veel van kunnen leren aan computer vison. Daarom zullen wij ook veel experimenten doen om meer te kunnen leren. Deze doen we door middel van trial en error.

De huidige standaard applicatie werk momenteel niet helemaal goed. Bij sommige plaatjes kan hij geen goede lokalisatie doen op het gezicht. Wij gaan ervan uit dat de oorzaak van dit probleem is dat de randdetectie gedeelte van het algoritme niet het beste resultaat geeft, waardoor de post-processing gedeelte de afbeelding niet goed kan herkennen.

Ons hoofddoel is daarom dus om een beter resultaat te kunnen krijgen door de randdetectie methode te veranderen. Beter in deze zin is het eindresultaat, qua accuratie en detail waardoor de lokalisatie gedeelte van de algoritme ook beter kan werken.

Naast onze hoofddoel willen we ook op de snelheid van het algoritme letten. We willen niet dat het goed resultaat heeft maar erg lang zou duren. Daarom wordt deze aspect ook overwogen voor het kiezen van de methode die we gaan implementeren.

2. Methoden

3.1 Robert's Cross

- Input

De input van deze methode is een *grayscale image.*

- Hoe werkt het

De Roberts Cross-operator voert een eenvoudige, snel te berekenen, 2-D ruimtelijke gradiënt meting uit op een afbeelding. De operator bestaat uit een paar 2 × 2 convolutie *kernels*, zoals weergegeven in:

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \& \quad G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Eén *kernel* is eenvoudig de andere die 90 ° is geroteerd. Deze kernels zijn ontworpen om maximaal te reageren op randen die lopen op 45 ° tot het pixelraster, één *kernel* voor elk van de twee loodrechte oriëntaties. Op elk punt in de afbeelding kunnen de resulterende gradiënt benaderingen worden gecombineerd om de totale gradiënt waarde te berekenen:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Hoewel meestal, wordt het berekend met behulp van een snellere manier:

$$|G| = |G_x| + |G_{xy}|$$

De oriëntatie hoek van de rand die aanleiding geeft tot de ruimtelijke gradiënt wordt gegeven door:

$$\theta = \mathrm{ATAN}(G_y/G_x) - 3\pi/4$$

- Voordeel en Nadeel

Het voordeel van deze methode is dat hij erg simpel is en dus ook erg snel is. Het nadeel van deze simpelheid is dat hij niet accuraat kan zijn. Deze methode is ook erg sensitieve voor ruis.

3.2 Sobel's Methode

- Input

De input van deze methode is een *grayscale image.*

- Hoe werkt het

De meeste rand detectiemethoden werken in de assumptie dat de rand optreedt waar een discontinuïteit in de intensiteit is of waar een zeer steile intensiteit gradiënt in het beeld te vinden is. Met gebruik van deze assumptie, als men de afgeleide intensiteitswaarde over het beeld neemt en de maximale waardes vindt, dit zou dan de rand zijn. De gradiënt is een vector waarvan de componenten meten hoe snel de pixelwaarde verandert in de x- en y-richting. Die componenten van de gradiënt kunnen dus worden gevonden met behulp van de volgende benadering:

De operator gebruikt twee *kernels* van 3 × 3 die worden ge*convolueerd* met het originele beeld *A.*

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \times A \qquad \& \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \times A$$

De x-coördinaat neemt toe in de horizontale richting en y-coördinaat in de verticale richting. Aangezien dat de (0,0) punt linksboven begint.

Op elk punt in de afbeelding kunnen de resulterende gradiënt benaderingen worden gecombineerd om de totale gradiënt waarde te berekenen:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Met de G_x en de G_y kunnen we ook de richting berekenen:

$$\theta = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

- Voordeel en Nadeel

Voordeel van deze methode is de simpelheid, detectie van randen en hun oriëntaties. Het nadeel is dat deze methode erg sensitieve is voor ruis en inaccuraat kan zijn.

3.3 Prewitt

- Input

De input van deze methode is een *grayscale* beeld of RGB beeld. RGB als input zou deze filter over alle channels gaan, dit levert een verbetering op ten koste van duurdere rekenkosten.

- Hoe werkt het

Deze methode is bijna hetzelfde als Sobel's methode, een methode om horizontaal en verticaal randen te vinden. Maar niet als Sobel's, deze methode geeft niet weer waarde aan pixels die dichter bij het midden zitten.

Deze operator gebruikt twee *kernels* van 3×3 die worden ge*convolueerd* met het originele beeld *A.*

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \times A \qquad \& \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \times A$$

Verder werkt deze methode hetzelfde als Sobel's methode.

- Voordeel en Nadeel

Het resultaat is bijna hetzelfde als die van Sobel's, maar uit bepaalde onderzoeken blijkt dan Prewitt net iets beter is.

"The experiments results shown that Prewitt edge detection technique is better than the Sobel edge detection technique. Though Prewitt technique is similar to Sobel but there is difference of spectral response"

3.4 Laplacian of Gaussian

- Input

De input van deze methode is een *grayscale image.* Sommige hebben ook een grayscale image nodig die al gefilterd is met Gaussian, dit heeft deze namelijk nodig maar sommige doen het zelf en sommige verwachten dat de input al gefilterd is.

- Hoe werkt het

De Laplacian is een 2D isotrope meeting voor de 2e ruimtelijke afgeleide van een afbeelding. De Laplacian van een afbeelding markeert regio's met een hoge intensiteit verandering en wordt daarom vaak gebruikt voor randdetectie.

De Laplacian wordt toegepast op een afbeelding die eerst is gladgestreken met iets dat een Gaussiaans filter om de gevoeligheid voor ruis te verminderen.

De Laplacian L (x, y) van een afbeelding met pixel intensiteitswaarde I (x, y) wordt gegeven door:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial x^2}$$

Zoals hier getoond, wordt de input geïllustreerd als een reeks afzonderlijke pixels; een discrete *convolution-kernel* kan de tweede afgeleide in de definitie van de Laplacian benaderen. Drie veel gebruikte kleine *kernels* worden zijn:

$$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}-1 & 2 & -1\\ 2 & -4 & 2\\ -1 & 2 & -1\end{bmatrix}$$

- Voordeel en Nadeel

Het voordeel van deze methode is het vinden van de juiste plaatsen van randen, en de breedte van de geteste pixel. Dit zorgt voor betere accuratie dan andere methodes. Het nadeel is dat als ruis door de Gaussian filter heen komt dat het een groot effect heeft op het eindresultaat omdat deze methode heel erg gevoelig voor is.

3.5 Canny

- Input

De input van de Canny methode is een *grayscale image.*

- Hoe werkt het

De Canny randdetectie algoritme is bij velen bekend als de optimale rand detector. De bedoelingen van Canny was om vele rand detectoren die er al waren toen hij aan zijn werk begon te verbeteren. Hij was erg succesvol in het bereiken van zijn doel, zijn ideeën en methoden is te vinden in zijn paper, "A Computational Approach to Edge Detection". Maar hoe werkt zijn methode nou?

De eerste stap van de methode is om de afbeelding met de Gaussian Blur filter te filteren. Deze *blur* verwijdert de meeste ruis waarvoor de Canny methode gevoelig voor is.

Daarna gebruikt Canny, Sobels filter die twee *kernels* gebruikt van 3 × 3 die worden ge*convolueerd* met het originele beeld *A.*

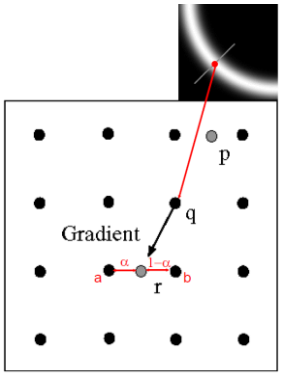
$$G_x = \begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \times A \qquad \& \qquad G_y = \begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \times A$$

Bereken vervolgens de grootte en hoek van de richting gradiënten:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

$$\theta = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

De volgende stap is om een zogenaamde “Non Maximum Suppression” techniek te doen. Nadat de sobel’s filter over de afbeelding is gegaan krijg je veel dikke randen, de uiteindelijke afbeelding moet dunnere randen hebben. Deze techniek zorgt daarvoor.



Figuur 1. Rand met meerdere pixels

Non Maximum Suppression werkt door de pixel met de maximale waarde in een rand te vinden. In de bovenstaande afbeelding treedt het op wanneer pixel q een intensiteit heeft die groter is dan p en r, waarbij pixels p en r de pixels zijn die in de gradiënt richting van q gaan. Als deze voorwaarde waar is, houden we de pixel, anders plaatsen we de pixel op nul (maak er een zwarte pixel van).

Non Maximum Suppression kan worden bereikt door de pixels te interpoleren voor een grotere nauwkeurigheid:

$$r = ab + (1 - a)a$$

De volgende stap is *Double Thresholding*. We merken dat het resultaat van *Non Maximum Suppression* niet perfect is, sommige randen zijn mogelijk geen randen en er is wat ruis in het beeld. *Double thresholding* zorgt hiervoor.

De laatste stap *Edge Tracking by Hysteresis*. Nu we hebben vastgesteld wat de sterke randen en zwakke randen zijn, moeten we bepalen welke zwakke randen daadwerkelijke randen zijn. Om dit te doen, voeren we een rand-volg-algoritme uit. Zwakke randen die zijn verbonden met sterke randen, zijn werkelijke/echte randen. Zwakke randen die niet zijn verbonden met sterke randen, worden verwijderd.

- Voordeel en Nadeel

Het voordeel van deze methode is dat het de juiste randen kan vinden, ook die niet duidelijk zichtbaar zou zijn voor *computer vision*. Deze methode kan ook beter om met ruis omdat er een *Gaussian* filter wordt gebruikt. Het nadeel is dat het een complex computer algoritme is en daarom veel tijd kost. Verder kan er ook een *false zero crossing* error optreden.

3.6 Samenvatting

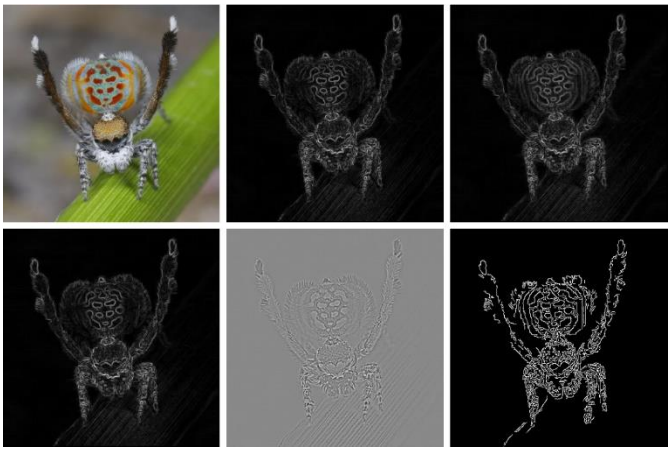
Deze hoofdstuk is bedoeld om een globaler overzicht te kunnen krijgen van alle voordelen en nadelen van alle beschikbare methodes. Deze tabel is uit de paper van (Maini & Aggarwal, 2009) gehaald.

Operator	Advantages	Disadvantages
Classical (Sobel, prewitt, Kirsch,…)	Simplicity, Detection of edges and their orientations	Sensitivity to noise, Inaccurate
Zero Crossing(Laplacian, Second directional derivative)	Detection of edges and their orientations. Having fixed characteristics in all directions	Responding to some of the existing edges, Sensitivity to noise
Laplacian of Gaussian(LoG) (Marr-Hildreth)	Finding the correct places of edges, Testing wider area around the pixel	Malfunctioning at the corners, curves and where the gray level intensity function varies. Not finding the orientation of edge because of using the Laplacian filter
Gaussian(Canny, Shen-Castan)	Using probability for finding error rate, Localization and response. Improving signal to noise ratio, Better detection specially in noise conditions	Complex Computations, False zero crossing, Time consuming

Table 1: Some Advantages and Disadvantages of Edge Detectors

3. Keuze

Uit ons onderzoek hebben we veel verschillende methoden gevonden die natuurlijk allemaal hun eigen voor en nadeel hebben. De ene is heel snel en heeft een redelijk goed resultaat, en de ander is iets langzamer maar ook net iets beter. En andere zijn gewoon erg traag maar hebben een heel hoog kwaliteit als eindresultaat.



Figuur 2. Linksboven: (origineel). Middel boven: (Sobel). Rechtsboven: (Sobel RGB). Rechtsonder: (Prewitt). Middel onder: (Laplace). Linksonder: (Canny)

Zoals wij eerder in deze document hebben aangekondigd, ons doel is om het lokalisatie gedeelte van het algoritme beter te kunnen laten werken door een de randdetectie gedeelte te verbeteren, terwijl we de snelheid van het algoritme in gaten houden.

Wij hebben onze keuze gemaakt door behulp van een beslissingsmatrix. In deze matrix hebben wij 3 attributen toegekend aangezien ons doel; Snelheid, Sensitiviteit & Kwaliteit. We hebben de hoogste weegfactor natuurlijk op kwaliteit gelegd, aangezien dat het ons hoofddoel is met deze implementatie.

Kwaliteits Attribut	Weeg factor	Robert's Cross		Sobel, Prewitt		Laplacian of Gausian		Canny	
		Waarde	Score	Waarde	Score	Waarde	Score	Waarde	Score
Snelheid	0.3	1.05s	95	1.29s	70	1.16s	80	4.21s	25
Sensitiviteit	0.2	-	30	-	30	--	20	++	80
Kwaliteit	0.5	+	60	+	75	-	20	++	90
Gewogen Gemiddelde			64.5		64.5		38		68.5
Minimum Score			30		30		20		25

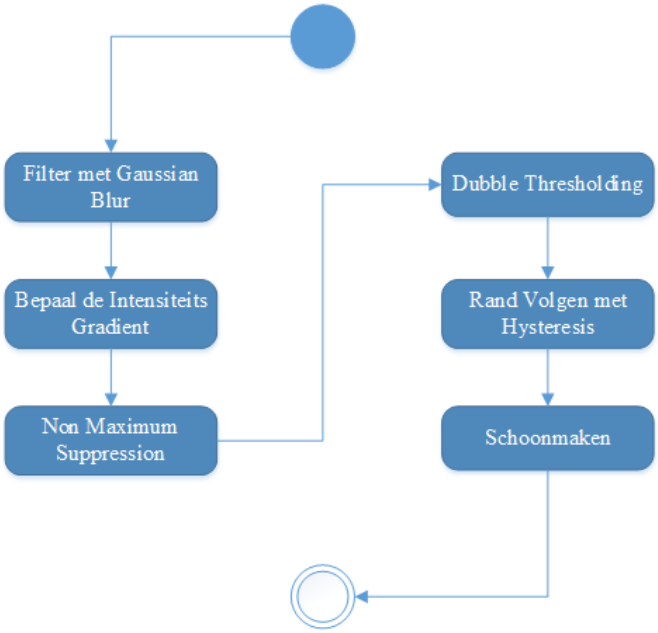
Uit ons beslissing matrix blijkt dus dat de optie met de hoogste waarde voor ons project Canny's methode is. Dus onze keuze is Canny's methode, maar aangezien er verschillende Canny's methode implementaties zijn te vinden gaan wij ook met verschillende Canny's methodes experimenten. Verschillende methodes in deze zin is dat er sommige net iets anders werken. Bijvoorbeeld een stap van het algoritme wordt in de ene implementatie gebruikt, en in de andere niet.

4. Implementatie

De implementatie zou worden gemaakt in de gereserveerde/standaard plek voor deze leer project. Deze vindt zich plaats in de *StudentPreprocessing.cpp* file, en om specifieker te zijn in *StepEdgeDetection* en *StepThresholding* functies. We willen deze functies er netjes uit laten zien, zoals ze een "main" zijn.

Om die nette "main" te behouden gaan we dus waarschijnlijk extra bestanden toevoegen. In deze nieuwe bestanden komen dan uitgebreiden functies of sub functies ten

plaats. We zullen ook een onderscheid maken tussen het randdetectie gedeelte en de thresholding. Daarvoor zullen we twee verschillende bestanden voor maken.



Figuur 3 Activity Diagram van de mogelijke implementatie

Hierboven is een *Activity Diagram* van een volledig Canny's methode. Dit is een uitgebreide versie van de Canny's methode. Deze methode heeft een nette resultaat met verdunde lijnen, maar is ook duur qua computerkracht vergeleken met andere methodes.

Dit was in het begin wel onze keuze, maar tijdens de implementatie waren we erachter gekomen dat het in de scope van dit project niet nodig was om de volledige versie van het algoritme te implementeren. We hebben dus geen *Non Maximum Supression*, *Double Thresholding* en Rand Volgen met *Hysteresis* gedaan.

Uiteindelijk hebben we dus de *Gaussian Blur* geïmplementeerd. We hebben door trial en error verschillende kernels getest. Uit deze test bleekt dat een van Laplacian of Gaussian kernel het beste resultaat had. De kernel met de -4 waarde in het midden was de beste, we hebben de 9x9 kernel versie hiervan gebruikt.

Voor de thresholding hebben we ook verschillende methodes uitgeprobeerd naast de *Double Thresholding*. We hebben op basis van een histogram threshold gedaan. Maar ze werkten allemaal niet zo goed. Normale threshold met een *hard coded* waarde werkte beter als we de juiste waarde hadden. Dus dat hebben we ook gehouden.

5. Evaluatie

Wat gaan we evalueren van onze implementatie? Aangezien ons oorspronkelijke doel van deze implementatie; lokalisatie gedeelte te verbeteren en snelheid. Gaan we onze implementatie ook daarop evalueren.

We gaan kijken of door onze implementatie de lokalisatie echt verbeterd is. Bij de origenele implementatie kan hij namelijk bij sommige afbeeldingen de ogen of neus niet goed lokaliseren. We nemen aan dat er meer afbeeldingen kunnen gelokaliseert vergeleken met de originele algoritme.

We gaan ook de snelheid van onze implementatie evalueren. Niet alleen de randdetectie en thresholding maar naar de hele algoritme als een applicatie. We denken dat er niet er veel verschil in zou zitten, niet sneller niet langzamer.

Literatuurlijst

Maini & Aggarwal. (2009, 15 Maart). Study and Comparison of Various Image Edge Detection Techniques. Geraadpleegd op 29 Maart 2019 van https://www.cscjournals.org/manuscript/Journals/IJIP/Volume3/Issue1/IJIP-15.pdf.

Mahendran. (2012, 5 November). A Comparative Study on Edge Detection Algorithms for Computer Aided Fracture Detection Systems. Geraadpleegd op 29 Maart 2019 van https://pdfs.semanticscholar.org/e3ec/ed9329be5fc635358b1abcf5ca74fc69b7bc.pdf.

Vincent & Folorunso. (2009). A Descriptive Algorithm for Sobel Image Edge Detection. Geraadpleegd op 29 Maart 2019 van http://proceedings.informingscience.org/InSITE2009/InSITE09p097-107Vincent613.pdf.

Tsankashvili. (2018, 29 April). Comparing Edge Detection Methods. Geraadpleegd op 29 Maart 2019 van https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e.

Liang. Canny Edge Detection. Geraadpleegd op 29 Maart 2019 van http://justin-liang.com/tutorials/canny/.