Implementatieplan Canny

Lars Versteeg
Youri Mulder

V2C, 2019

Inhoud

1.1 Namen en datum

Lars Versteeg, Youri Mulder 3/29/2019

1.2 Doel

Een visueel betere implementatie van de edge detection. Voor ons houdt dit in dat de randen dunner zijn en het voor het oog er mooier uitziet. Dit hoeft niet te betekenen dat de huidige implementatie van de feature detection en extraction er beter mee overweg kan. Ons doel is om de lijntjes dunner en duidelijker te krijgen.

1.3 Methoden

Je geeft hier aan welke methoden er zijn, wat de verschillen tussen de methodes zijn.

S.NO	Opertor	Complexity		Noise sensitivity	False Edges
		Time	Space		
1	Sobel	lower	high	Less Sensitivity	More
2	Canny	high	high	Least Sensitivity	Least
3	Robert	high	high	Sensitivity	More
4	Prewit	low	lower	Least Sensitivity	More
5	Laplacian of Gausian	low	least	Least Sensitivity	More
6	Zero crossing	low	less	Least Sensitivity	More

1.3.1 Multi-stage algoritmes

Canny

Canny maakt gebruikt van verschillende algoritmes om tot de uiteindelijke edge detection plaatje te komen. Canny edge detection maakt gebruik van de volgende stappen.

1. Smoothing, remove noise. (Gaussian filter)
2. Calculating the magnitude and gradient direction of the image.
3. Edge thinning (non-maximum suppression)
4. Thresholding (double thresholding weak and strong edge)
5. Hysteresis (suppress the weak edges if not connected to a strong edge)

Deriche (Canny-Deriche)

1. Smoothing, remove noise. (Gaussian filter)
2. Calculating the magnitude and gradient direction of the image.
3. Edge thinning (non-maximum suppression)
4. Thresholding (double thresholding weak and strong edge)
5. Hysteresis (suppress the weak edges if not connected to a strong edge)

Deriche lijkt qua implementatie identiek te zijn aan Canny, dit is niet het geval. Derichemaakt gebruik van IIR(Infinite Impulse Response) filter

First order versus Second order derivatives.

1.3.2 Kernels

Om de gradiënt magnitude te krijgen of een image te smoothen zijn er verschillende kernels beschikbaar. Hieronder staan de drie meest gebruikte gradiënt magnitude kernels.

1.3.2.1 Edge detection kernels

Sobel

Matrices voor de gradiënt magnitude x en y.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

Separatie van de convolutie.

$$\mathbf{G}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * (\begin{bmatrix} -1 & 0 & +1 \end{bmatrix} * \mathbf{A}) \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix} * (\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \mathbf{A})$$

Berekening van de intensiteit van de gradiënt.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

Sobel maakt gebruik van twee kernels om een plaatje met randen te maken. eentje is voor de x as(Gx) en de ander voor de y as(Gy). De direct naastgelegen pixels links en rechts in het geval van Gx worden zwaarder meegenomen in het geheel dan de schuin aangelegen pixels.

De richting van het gradiënt wordt bepaalt door de inverse tangens te nemen van de Gy en Gx.

Prewitt

Matrices voor de gradiënt magnitude x en y.

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$

Separatie van de convolutie.

Gx =

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

Berekening van de intensiteit van de gradiënt.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Berekening van de richting van de gradiënt.

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

Prewitt maakt gebruik van twee kernels om een plaatje met randen te maken. eentje is voor de x as(Gx) en de ander voor de y as(Gy). De direct naastgelegen pixels links en rechts in het geval van Gx worden **niet** zwaarder meegenomen in het geheel dan de schuin aangelegen pixels. Dit is het enige verschil met de sobel kernel.

De richting van het gradiënt wordt bepaalt door de inverse tangens te nemen van de Gy en Gx.

Roberts cross.

Matrices voor de gradiënt magnitude x en y.

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}.$$

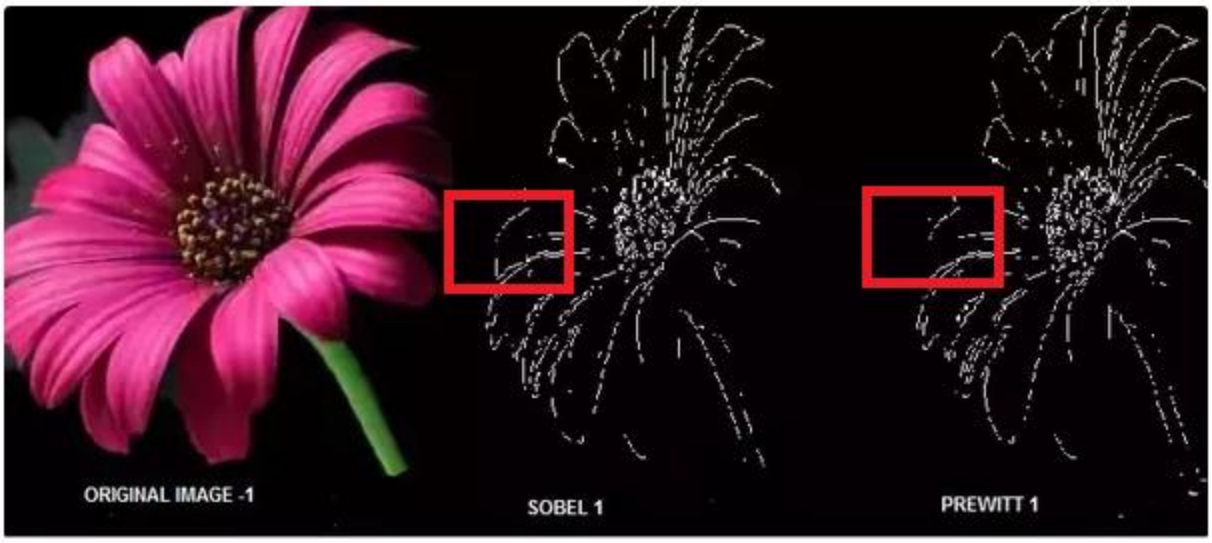
Berekening van de intensiteit van de gradiënt.

$$\nabla I(x, y) = G(x, y) = \sqrt{G_x^2 + G_y^2}.$$

Berekening van de richting van de gradiënt, omdat de gradiënten diagonaal worden berekend moet je de hoek recht trekken door er 3pi/4(135 graden) van af te halen.

$$\Theta(x, y) = \arctan\left(\frac{G_y(x, y)}{G_x(x, y)}\right) - \frac{3\pi}{4}.$$

Roberts cross is erg gevoelig voor ruis. Dit komt omdat er maar twee waardes worden gebruikt om de magnitude te berekenen.



Example 1 of Sobel And Prewitt Edge Detection Techniques.

https://www.quora.com/Why-is-Sobel-edge-detection-preferred-over-Prewitt-edge-detection
In de bovenstaande afbeelding zie je sobel en prewitt vergeleken. Het verschil is minimaal, maar sobel heeft minder last van ruis. Kijk bijvoorbeeld naar het rode vakje. Daar zie je bij sobel wel een gedetecteerde rand, maar bij prewitt is geen rand te zien. Als je naar het originele plaatje kijkt hoort daar wel een rand te zitten

1.3.2.2 Smoothing kernels

Gaussian filter (linear)

Deze filter is gebaseerd op een normaal verdeling. Dat betekend dat in het midden van de kernel hoge waardes zitten en naarmate je verder naar buiten gaat de lagere waardes. Dit zorgt ervoor dat de pixel die in het midden ligt het meeste aandeel heeft in het resultaat. Hierdoor houdt de pixel waarschijnlijk een waarde dichter bij zijn initiële waarde. De meest gebruikte kernel groottes zijn 3x3 en 5x5.

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	3
2	4	5	4	1

https://www.researchgate.net/figure/Gaussian-filter_fig1_321426272

Mean filter (Non-linear)

Bij de mean filter heeft de middelste pixel niet de hoogste bijdragen. De pixels aan de rand hebben evenveel bijdrage aan het resultaat als het midden. Een mean filter is sneller dan een gaussian filter, omdat de gewichten van alle vakjes hetzelfde zijn. Een mean filter is ook te gebruiken in verschillende groottes. De meeste gebruikte groottes zijn hier 3x3 en 5x5.

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

Een methode om kernel filters te gebruiken.
https://en.wikipedia.org/wiki/Kernel_(image_processing)
Een methode om gaussian filter te gebruiken
Je kunt hier kiezen uit verschillende sizes.

https://en.wikipedia.org/wiki/Gaussian_filter
Een methode om edge detection toe te passen.
https://en.wikipedia.org/wiki/Canny_edge_detector
Een methode om een threshold aan te geven en de edges buiten deze threshold te verwijderen.
https://en.wikipedia.org/wiki/Canny_edge_detector

1.4. Keuze

Je geeft een onderbouwing over waarom een bepaalde methode is gekozen, en/of waarom bepaalde settings zijn gebruikt.

Op het moment wordt gebruik gemaakt van Laplacian operator. Deze operator heeft een hogere snelheid en kost minder geheugen dan het Canny algoritme. Wij willen graag meer precisie, dus leveren wij graag tijd en ruimte in voor nauwkeurigheid.

Canny is een veel gebruikt algoritme en er is dus ook veel over te vinden. Dit vinden wij belangrijk, omdat wij naast onze eigen informatie ook graag ergens informatie vandaan willen halen zonder dat wij hier dagen naar opzoek zijn.

De standaard implementatie van canny heeft vaste stappen met vaste technieken. Je zou bij elke stap een andere techniek kunnen kiezen om het resultaat te verbeteren, maar wij gaan het standaard Canny algoritme implementeren, omdat wij denken dat deze het beste presteert.

1.4.1 Methode

1.4.1.1 Smoothing, remove noise. (Gaussian filter)

Zo wordt er bij de ruis verwijdering Gaussian gebruikt. Gaussian wordt hier vaak gebruikt met een kernel van 5x5 en een sigma van 1.4 t/m 1.6. Voor elk plaatje kan de ‘sweet spot’ van de sigma ergens anders liggen. Wij gaan dit testen. Een verdere uitwerking gaat te vinden zijn in ons meetrapport.

1.4.1.2 Calculating the magnitude and gradient direction of the image. (sobel filter)

In vergelijking met prewitt en roberts cross is sobel het mist gevoelig voor ruis. Daarom kiezen wij ervoor om sobel te gebruiken.

1.4.1.3 Edge thinning (non-maximum suppression)

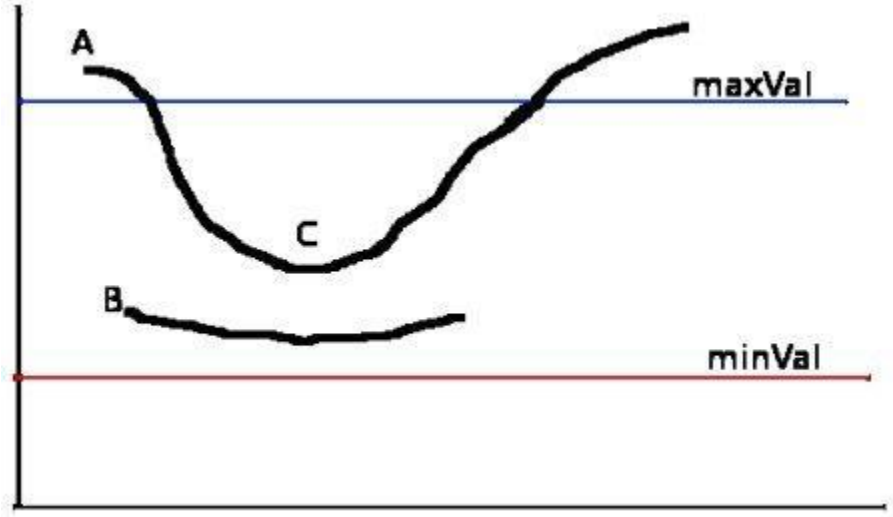
De enige bekende techniek is non-maximum suppression. Hier ga je kijken naar de buren van de pixel door middel van de richting van het gradiënt die is berekend in de vorige stap.

1.4.1.4 Thresholding (double thresholding weak and strong edge)

Bij thresholding ga je kijken naar potentiele randen. De randen die boven de threshold vallen zullen worden beschouwd als een rand. Alle waardes onder de threshold worden beschouwd als geen rand.

Omdat Canny in de volgende stap hysteresis doet moeten wij gebruik maken van een double threshold. Deze maakt een verschil tussen slappe en sterke randen.
Om hysteresis te doen heb je een double threshold nodig. De double threshold verdeelt het spectrum in 3 gebieden. Alles wat boven de hoogste threshold valt zal worden beschouwd als een sterke rand, alles wat tussen de hoogste en laagste threshold valt zal worden beschouwd als slappe rand en alles wat onder de laagste threshold valt zal worden beschouwd als geen rand.

hoogste threshold = maxVal
laagste threshold = minVal



https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
Boven de blauwe streep = sterke rand
Tussen de blauwe en rode streep = slappe rand
Onder de rode streep = geen rand

1.4.1.5 Hysteresis (suppress the weak edges if not connected to a strong edge)

Slappe randen worden nog niet beschouwd als een echte rand. Deze worden in deze stap mogelijk een echte rand als deze slappe rand verbonden zit aan een sterke rand. Als de rand niet verbonden zit aan een echte rand dan wordt het beschouwd als geen rand. De slappe rand zou ook diagonaal verbonden kunnen zitten aan een echte rand. In dit geval is het ook een echte rand.

1.4.2 Settings

1.4.2.1 Threshold

De waardes van de threshold zijn lastig te bepalen zonder de informatie van de afbeelding. De threshold waardes zullen verschillen per afbeelding. Het is mogelijk door middel van het Otsu's algoritme een threshold te bepalen. Deze threshold zou kunnen werken. Wij gaan dit uitproberen en dit wordt meegenomen in een meetrapport.

1.4.2.2 Gaussian sigma

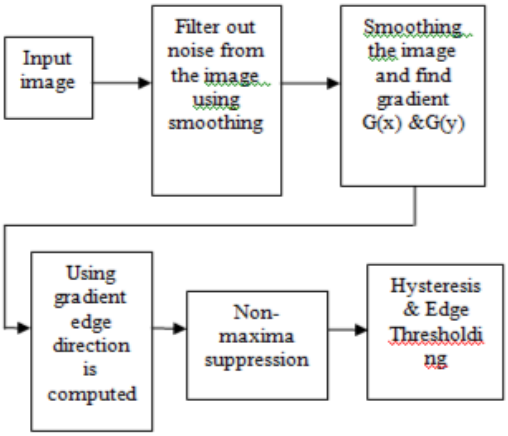
Net zoals bij de threshold waardes is het moeilijk om de sigma te bepalen. Elk plaatje heeft namelijk een andere waarde nodig, omdat in sommige afbeeldingen het belangrijk is dat niet al het detail verloren gaat en bij andere plaatjes kan het ervoor zorgen dat de ruis weg wordt genomen waardoor de rand detectie beter kan worden. Wij gaan verschillende sigma's proberen en nemen dit mee in een meetrapport.

1.5. Implementatie

Je geeft aan hoe deze keuze is geïmplementeerd in de code

Zoals al eerder verteld is Canny een multi-stage algoritme. Dit houdt in dat er meerdere stappen in het algoritme zitten om tot het eind resultaat te komen. Elke stap zullen wij apart behandelen en onze implementatie.

Wij kiezen ervoor elke functie te maken in een namespace, omdat wij het overbodig vinden een extra klasse aan te maken met zijn eigen data. Het lijkt ons handig dat je de functies ook kan gebruiken buiten Canny om.



ijarcs.info/index.php/Ijarcs/article/download/1603/1591

Om onze implementatie te realiseren maken wij twee nieuwe bestanden aan. Een header en een source bestand. De header gaat EdgeDetection.h heten en de source gaat EdgeDetection.cpp heten, omdat elke header nu ook al alleen uit een .h bestaat behouden we deze standaard. Zelf zouden wij .hh, .hpp .hxx willen gebruiken om duidelijk te krijgen of het een c bestand of c++ bestand is, maar om uniformiteit te behouden doen wij dit niet.

Om de code overzichtelijker te gaan we gebruik maken van een typedef van een std::vector<std::vector<double>>. Deze typedef kunnen we gebruiken in plaats van het helemaal voluit te hoeven typen. Dit zorgt ervoor dat het makkelijker leesbaar wordt.

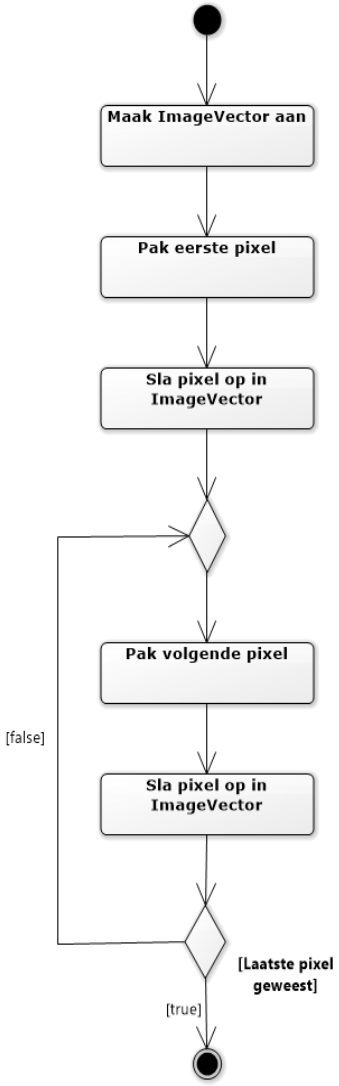
```
namespace EdgeDetection {
	typedef std::vector<std::vector<double>> imageVector;
};
```

De functies die wij gebruiken om Canny uit te voeren willen wij apart ook kunnen gebruiken als dit nodig gaat zijn, daarom hebben wij ervoor gekozen dit allemaal in een namespace EdgeDetection te plaatsen. Elke functie heeft een input en output sommige via een return type en sommige via een reference in de parameters.

1.5.1 imageVectorFromIntensityImage

```
imageVector imageVectorFromIntensityImage(const IntensityImage& image);
```

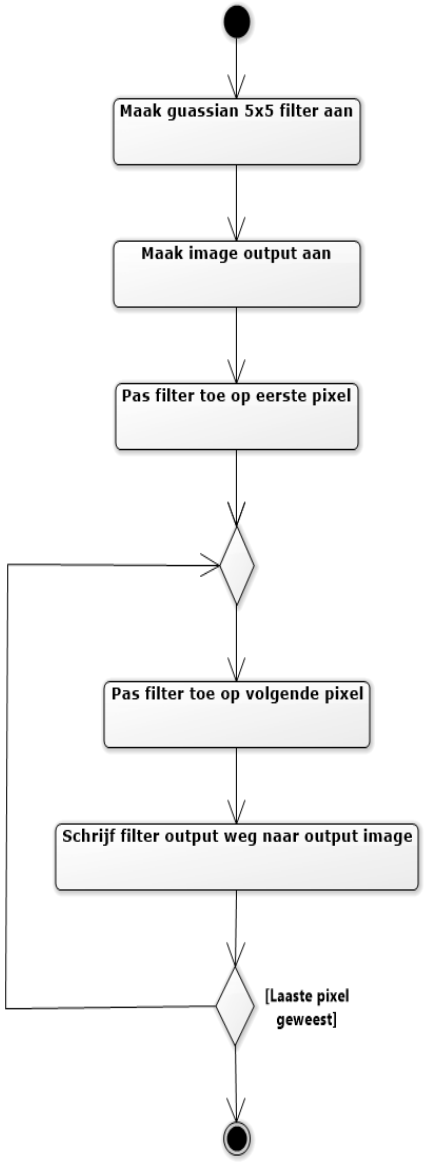
Deze functie wordt gebruikt om een IntensityImage naar onze interne implementatie van een image te krijgen de imageVector. Deze kunnen wij makkelijker gebruiken om de afbeeldingen te manipuleren.



1.5.2 applyGaussian

```
imageVector applyGaussian(const imageVector &image, double sigma);
```

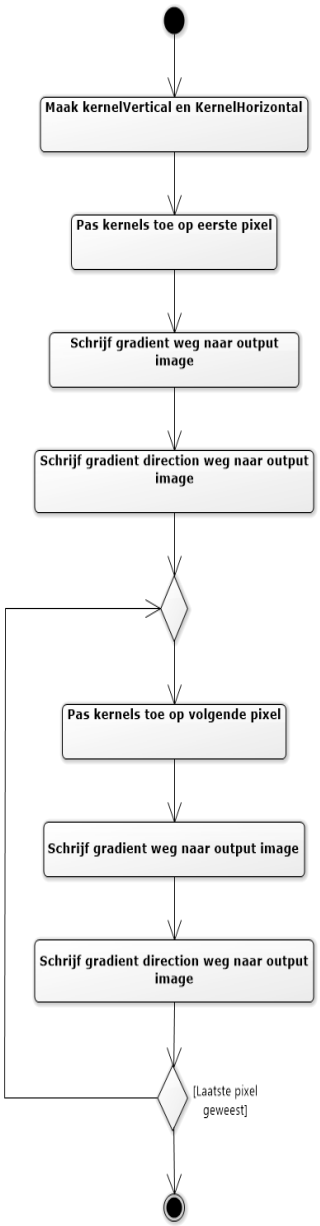
Deze functie wordt gebruikt om gaussian ruis uit het plaatje te halen. Image is de afbeelding waar je de gaussian filter op wil toepassen en sigma bepaald de standaardafwijking van de gaussian kernel. Dit is stap een van het Canny proces.



1.5.3 sobelFilter

```
void sobelFilter(const imageVector &sourceImage, imageVector &destImage, imageVector &directionImage);
```

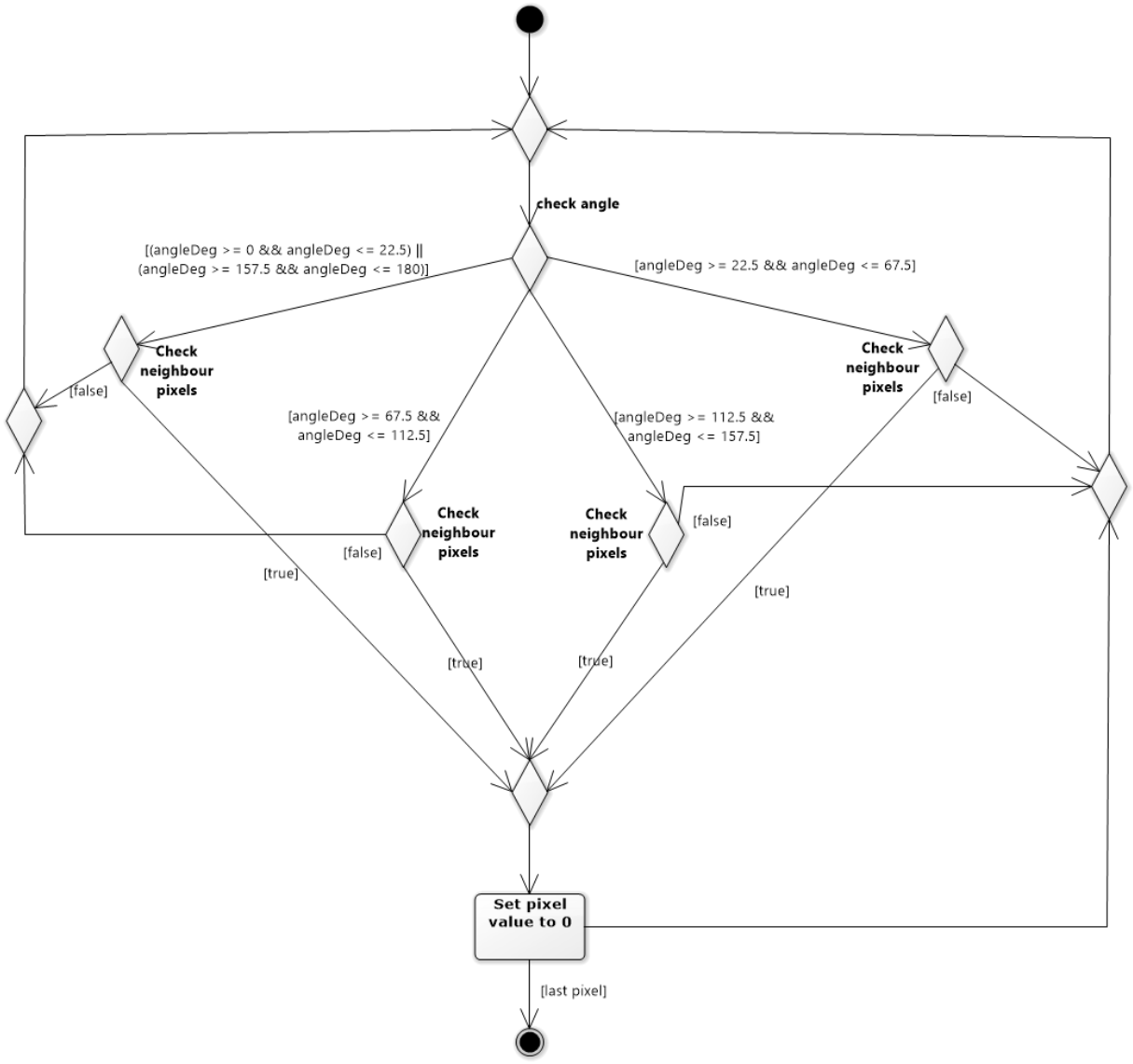
Deze functie wordt gebruikt om de sobel filter op een plaatje te gebruiken. De sourceImage is het plaatje waar je de sobel filter op wil toepassen. Uit de sobel filter komen twee waardes namelijk de nieuwe gradiënt van een pixel en de richting van deze gradiënt. De nieuwe gradient van een pixel wordt opgeslagen in de destImage en de bijbehorende richting wordt in de directionImage. Dit is stap twee van het Canny proces.



1.5.4 nonMaxSupp

void nonMaxSupp(imageVector &image, const imageVector &directions);
Non-maximum supression is stap drie van het Canny process. Deze functie zorgt ervoor dat de lijnen dunner worden. Dit is een van de belangrijkste functies voor het Canny algoritme. Door deze functie herken je gelijk dat het om het canny algoritme gaat, want dit is een van de weinige edge detection algoritme die dunne lijntjes terug geeft.

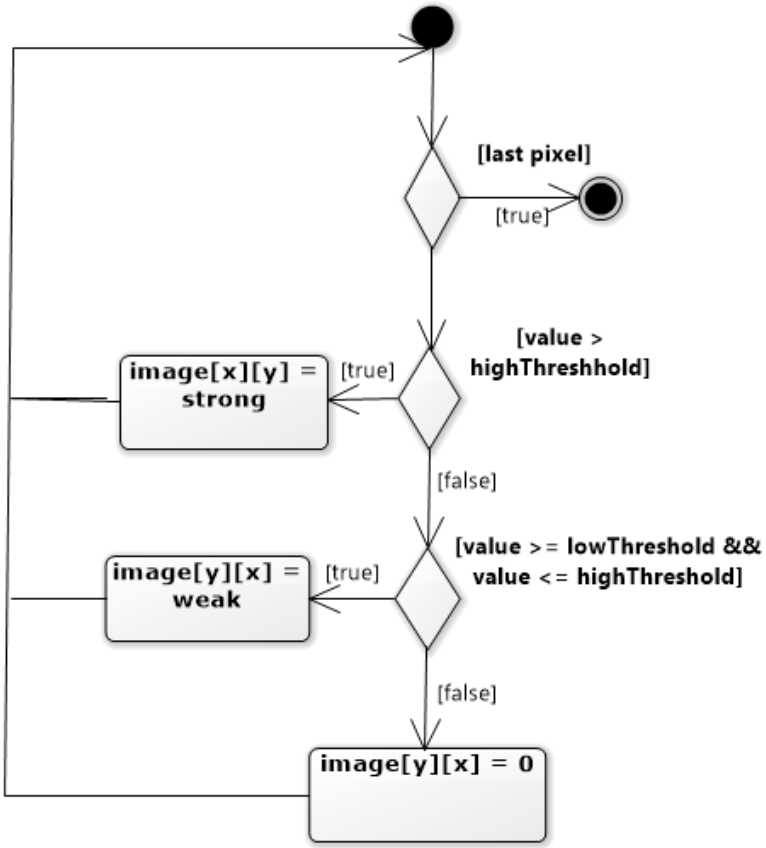
De image parameter is de beginsituatie met de dikke randen. De bijbehorende richtingen zijn opgeslagen in de directions. In dit geval wordt de invoer image aangepast. Er wordt dus geen nieuwe image aangemaakt die teruggegeven wordt.



1.5.5 doubleThreshold

void doubleThreshold(imageVector &image, const Intensity& lowThreshold, const Intensity& highThreshold, const Intensity& strong, const Intensity& weak);

Dit is de vierde stap van het Canny proces. Hier wordt gekeken waar er in het plaatje daadwerkelijk randen zitten. Dit wordt gedaan door een dubbele threshold. Deze threshold wordt meegegeven door de lowThreshold en highThreshold parameter. Een waarde die boven de hightThreshold ligt wordt beschouwd als een rand(strong). En een waarde die tussen de twee threshold waardes ligt wordt beschouwd als een mogelijk rand(weak). Dit wordt allemaal aangepast in de image die wordt meegegeven. Let dus op dat de image wordt aangepast.



1.5.6 tracking

```
void tracking(imageVector &image, const Intensity &strong, const Intensity &weak);
```

De tracking functie gaat voor iedere pixel na of deze een slappe rand is. als dit het geval is, kijkt hij of hij verbonden is met een sterke rand. In het geval dat hij aan een sterke rand vast zit, zet hij de slappe rand op sterk. Als de slappe rand niet aan een sterke rand vast zit wordt de rand op 0 gezet. Dus bestaat deze rand niet meer. Dit is de laatste en dus vijfde stap in het Canny process.

1.5.7 toHistogram

```
void toHistogram(const imageVector &image, std::array<double, 256> &histogram);
```

Deze functie maakt van de afbeelding een histogram. De histogram wordt gemaakt van de image die meegegeven wordt. De uitkomst wordt opgeslagen in de parameter histogram. Dit is geen stap van Canny, maar is wel nodig om otsu te kunnen uitvoeren. Otsu hebben wij toegevoegd om automatisch de threshold waarde te bepalen.

1.5.8 otsu

```
double otsu(int totalAmountOfPixels, std::array<double, 256>& histogram);
```

Deze functie wordt gebruikt om automatisch de doubleThreshold te bepalen. Door een histogram mee te geven die de beste waarde zoekt om zoveel mogelijk in de voorgrond pixels te krijgen als in de achtergrond pixels. De lowThreshold is * 0.5 van de highThreshold. Wij hebben ervoor gekozen om de uitkomt nog te vermenigvuldigen met een factor die wij in de hightThreshold stoppen. Zo kunnen wij beter bepalen wat de threshold wordt. Dit is dus een semiautomatische oplossing voor thresholding.

1.6. Evaluatie

De implementatie is vergeleken met de uitvoer van andere implementaties op het internet. Deze implementatie was geschreven in python. Hier kwam ongeveer dezelfde uitkomst uit als we de thresholding en sigma naar onze wens aanpasten.

In het meetrapport hebben wij de verschillende threshold waardes en sigma waardes vergeleken. Dit maakt een groot verschil voor de uitkomst van het plaatje. Ook hebben wij gekeken wat het verschil is tussen de snelheid van de standaard implementatie en onze implementatie.

Onze implementatie werkt helaas niet met het herkennen van de facial features. Om facial features te herkennen wordt gebruik gemaakt van dilation en erosion, omdat onze implementatie op sommige plekken zo een dun lijntje heeft kan de lijn verdwijnen. Als onze lijnen verdwijnen is het niet meer mogelijk om het gezicht te herkennen.

1.7. Bronnen

Tsankashvili, N. (2018, 29 april). Comparing Edge Detection Methods. Geraadpleegd op 2 april 2019, van https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e

CMU. (z.d.). Which Edges Matter? Geraadpleegd op 2 april 2019, van http://www.cs.cmu.edu/~aayushb/pubs/edges.pdf

Katiyar, S., & Arun, P. (z.d.). Comparative analysis of common edge detection techniques in context of object extraction. Geraadpleegd op 2 april 2019, van https://arxiv.org/ftp/arxiv/papers/1405/1405.6132.pdf

https://en.wikipedia.org/wiki/Sobel_operator
https://en.wikipedia.org/wiki/Prewitt_operator
https://en.wikipedia.org/wiki/Roberts_cross