Implementatieplan

Bart van Netburg en Marty vos

12-4-2019

Inhoudsopgave

[Doel 3](#_Toc6050778)

[Methoden 4](#_Toc6050779)

[Canny 4](#_Toc6050780)

[Stappen: 4](#_Toc6050781)

[Bronnen: 4](#_Toc6050782)

[Roberts 4](#_Toc6050783)

[Stappen: 4](#_Toc6050784)

[Bronnen: 4](#_Toc6050785)

[Prewitt 5](#_Toc6050786)

[Stappen: 5](#_Toc6050787)

[Bronnen: 5](#_Toc6050788)

[Sobel 5](#_Toc6050789)

[Stappen: 5](#_Toc6050790)

[Bronnen: 5](#_Toc6050791)

[Deriche 6](#_Toc6050792)

[Stappen: 6](#_Toc6050793)

[Bronnen: 6](#_Toc6050794)

[Laplacian 6](#_Toc6050795)

[Stappen: 6](#_Toc6050796)

[Bronnen: 6](#_Toc6050797)

[Keuze 7](#_Toc6050798)

[Implementatie 8](#_Toc6050799)

[Mask class 8](#_Toc6050800)

[GaussianFilter class 8](#_Toc6050801)

[Tresholding 8](#_Toc6050802)

[stepEdgeDetection 8](#_Toc6050803)

[Evaluatie 9](#_Toc6050804)

# Doel

Het doel van dit project is om een eigen implementatie te maken van een edge detectie algoritme.

# Methoden

## Canny

Stappen:  
 Gaussian filter

haal vervolgens de volgende twee kernels over de image:

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

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

Gx = Gy =

Combineer de twee resulterende afbeeldingen met de volgende formule:

|G| = √(Gx2 + Gy2)

Maar om de berekening te versnellen word vaak de volgende formule gebruikt:

|G| = |Gx| + |Gy|

Non-maximum onderdrukking

Hysteresis Thresholding

### Eigenschappen:

Zes stappen.  
Maakt gebruik van Sobel.  
Dunne lijnen dankzij de Non-maximum onderdrukking.  
Weinig ruis door de Hysteresis Thresholding.

### Bronnen:

<https://docs.opencv.org/3.4.3/da/d22/tutorial_py_canny.html>

<http://robotics.technion.ac.il/courses/Advanced_Laboratory/Lab7/ARL_7_read.pdf>

<https://en.wikipedia.org/wiki/Canny_edge_detector>

## Roberts cross

### Stappen:

Haal eerst de volgende twee kernels over de image heen:

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

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

Gx Gy

Bereken vervolgens het gradiënt aan de hand van de twee nieuwe afbeeldingen die zijn ontstaan uit bovenstaande kernels en het origineel. Dat kan met de volgende formule:

|G| = √(Gx2 + Gy2)

### Eigenschappen:

Drie stappen  
Twee kleine kernels  
Erg gevoelig voor ruis  
Kernel voor de X en de Y richting

### Bronnen:

<http://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm>

<http://robotics.technion.ac.il/courses/Advanced_Laboratory/Lab7/ARL_7_read.pdf>

<https://en.wikipedia.org/wiki/Roberts_cross>

## Prewitt

### Stappen:

haal eerst de volgende twee kernels over de image:

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

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

Gx = Gy =

Combineer de twee resulterende afbeeldingen met de volgende formule:

|G| = √(Gx2 + Gy2)

Maar om de berekening te versnellen word vaak de volgende formule gebruikt:

|G| = |Gx| + |Gy|

### Eigenschappen:

Drie stappen  
Kernel voor de X en de Y richting  
Variatie op Sobel

### Bronnen:

<https://en.wikipedia.org/wiki/Prewitt_operator>

<http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm#4>

Sobel

### Stappen:

haal eerst de volgende twee kernels over de image:

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

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

Gx = Gy =

Combineer de twee resulterende afbeeldingen met de volgende formule:

|G| = √(Gx2 + Gy2)

Maar om de berekening te versnellen word vaak de volgende formule gebruikt:

|G| = |Gx| + |Gy|

### Eigenschappen:

Drie stappen  
Kernel voor de X en de Y richting  
Erg gevoelig voor ruis, maar minder gevoelig dan Roberts cross

### Bronnen:

<https://en.wikipedia.org/wiki/Sobel_operator>

<http://robotics.technion.ac.il/courses/Advanced_Laboratory/Lab7/ARL_7_read.pdf>

Deriche

### Stappen:

stappen:  
 IIR filter

vinden van de intensiteitsgradiënten van de afbeelding

Non-maximum onderdrukking

Hysteresis Thresholding

### Eigenschappen:

Vijf stappen  
Lijkt op Canny  
Maakt gebruik van IIR filter  
Zou betere resultaten moeten geven dan Canny

### Bronnen:

<https://en.wikipedia.org/wiki/Deriche_edge_detector>

## Laplacian

### Stappen: Gaussian filter

Laplacian filter

Thresholding

Voorbeelden van Laplacian filter kernels:

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

### Eigenschappen:

Drie stappen  
Maakt gebruik van maar één kernel.  
Zonder Gaussian filter erg gevoelig voor ruis

### Bronnen:

<http://www.aishack.in/tutorials/sobel-laplacian-edge-detectors/>  
<http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm#2>

# Keuze

Wij hebben gekozen voor Laplacian. Wij willen graag een simpele methode die makkelijk te begrijpen is en licht draait. Voor de laplacian hoef je maar 2 kernels over de afbeelding te halen, een gaussian filter en de laplacian filter zelf. Voor het filter gebruiken wij het volgende kernel:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | -4 | -4 | -4 | 1 | 1 | 1 |
| 1 | 1 | 1 | -4 | -4 | -4 | 1 | 1 | 1 |
| 1 | 1 | 1 | -4 | -4 | -4 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

9 /

Wij hebben meerdere laplacian kernels geprobeert maar deze gaf het beste resultaat.

# Implementatie

Wij hebben ten eerste twee nieuwe classes aangemaakt, de Mask class en de GaussianFilter class.

## Mask class

Met de Mask class kun je een Mask object aanmaken. Bij het aanmaken van dit object moet je een masker als vector<vector<int>> meegeven en optioneel nog een int die aangeeft door hoeveel het masker moet worden gedeeld. Standaard staat die op nul, in dat geval gebruikt hij de som van het masker.

Vervolgens kun je van het Mask object de apply functie aanroepen met twee maal een intensityImage als parameter. De eerste is de originele afbeelding. De tweede is waar het resultaat in terecht komt.

## GaussianFilter class

Met de GaussuanFilter class kun je heel makkelijk een GaussianFilter genereren. Je kunt een Gaussian filter object aanmaken met als parameter het sigma en de maskSize.

Om vervolgens de kernel op te vragen kun je de getGaussianFilter methode aanroepen met als parameter de middelste waarde van het filter als int.

## Tresholding

Tresholding wordt op een hele simpele manier gedaan. Er wordt door de afbeelding heen geloopt. Als de waarde boven de vastgelegde threshold zit wordt de waarde op 255 gezet als hij er onder zit wordt de waarde op 0 gezet.

## stepEdgeDetection

Eerst wordt er een GaussianFilter aangemaakt. Met dit filter wordt een GaussianFilter Mask aangemaakt.

Vervolgens wordt er een edgeDetection Mask aangemaakt.

Daarna worden er twee IntensityImage’s aangemaakt om de resultaten van de maskers in op te slaan.

Eerst wordt de GaussianFilter Mask toegepast en daarna wordt op dat resultaat de edgeDetection Mask toegepast.

Wat daaruit komt is de return waarde.

# Evaluatie

De applicatie moet ten eerste werken zonder te crashen of vast te lopen.