1. Edge detection

1.1. Namen en datum

Hamza ait Messaoud, 01-01-2018

1.2. Doel

Bepalen welke vorm van edge detection het beste resultaat oplevert, voor deze opdracht.

1.3. Methoden

We nemen de 3 besproken methodes om edge detection te bepalen, namelijk; Laplacian of Gaussian, Sobel en Canny.

Canny

De Canny-operator werkt in een meerstappenproces. Allereerst wordt het beeld afgevlakt door Gaussiaanse convolutie. Vervolgens wordt een eenvoudige tweedimensionale eerste afgeleide operator toegepast op het afgevlakte beeld, dit om gebieden van het beeld met hoge eerste ruimtelijke derivaten te markeren. Randen geven ribbels in het beeld. Het algoritme gaat vervolgens langs de bovenkant van deze richels en stelt alle pixels die niet daadwerkelijk op de richel staan op nul om een dunne lijn in de uitvoer op te leveren, een proces dat bekend staat als non-maximal suppression. Het trackingproces gebruikt hysteresis die door twee tresholds worden geleid: T1 en T2, met T1> T2. Tracking kan alleen beginnen op een punt op een rand hoger dan T1. Het volgen gaat dan vanaf dat punt in beide richtingen verder totdat de hoogte van de rand onder T2 valt. Deze hysteresis zorgt ervoor dat ruisige randen niet worden opgesplitst in meerdere fragmenten. (R. Fisher, 2018)

Laplacian of Gaussian

Laplacian of Gaussian(LoG) is, zoals de naam doet vermoeden, een combinatie van Laplacian en Gaussian filter. Dit is omdat de Laplacian filter gevoelig is voor ruis en dit onderdrukt wordt door de Gaussian filter. Een voorbeeld van een LoG kernel is Figuur 1, deze dient geconvolueerd te worden met de afbeelding. Als de uitkomst van de berekende pixel hoger is dan 255 krijgt deze de waarde 255, als de waarde lager is dan 0 dan krijgt de pixel de waarde 0 alles daartussen wordt erkend aan de pixel. (R. Fisher, Laplacian/Laplacian of Gaussian, 2018)

0	1	0
1	-4	1
0	1	0

Figuur 1. LoG kernel

Sobel

De operator gebruikt twee kernels van 3x3 die zijn geconvolueerd met de oorspronkelijke afbeelding om de benaderingen van de derivaten te berekenen: een voor horizontale wijzigingen en een voor verticale. Als we A definiëren als het bronbeeld en Gx en Gy de twee afbeeldingen zijn die op elk punt respectievelijk de horizontale en verticale afgeleide benaderingen bevatten (Figuur 2) vervolgens moet de berekening uit Figuur 3 uitgevoerd worden. (R. Fisher, Sobel Edge Detector, 2018)

$G_x =$

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

* A

$G_y =$

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

* A

Figuur 2. Sobel Kernel

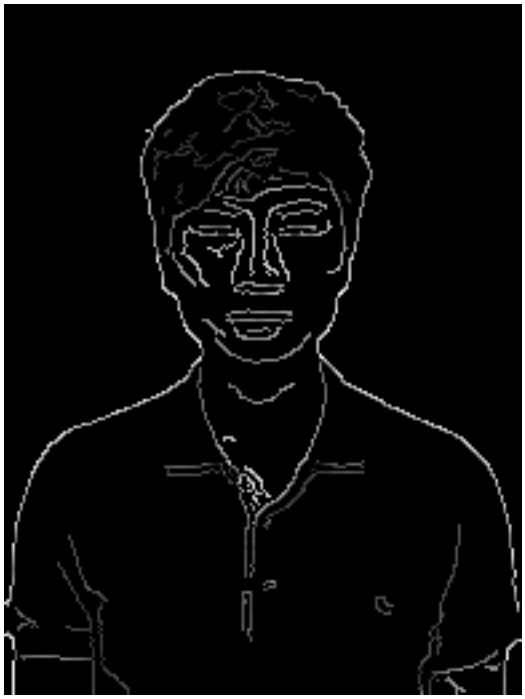
$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Figuur 3.

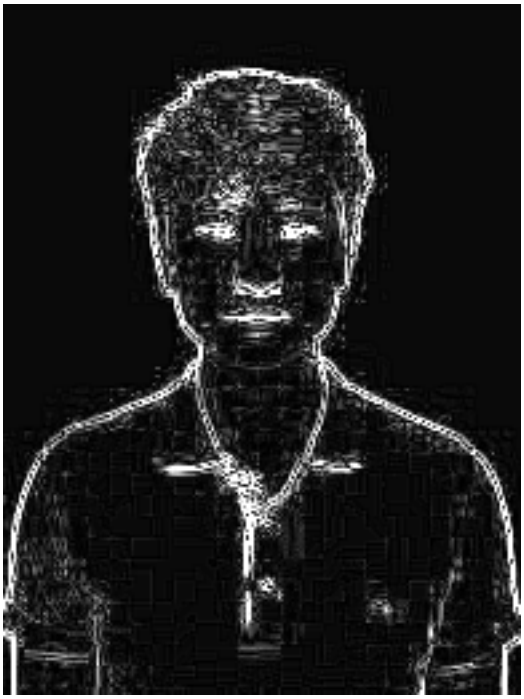
1.1. Keuze

De keuze is gevallen op Laplacian of Gaussain. Dit is niet omdat het de beste resultaten oplevert mar omdat het programma moeilijker omgaan met anderen vormen van edge detection.

De methode die de beste resultaten opleverde was Canny.



Figuur 1. Canny



Figuur 2. Laplacian Of Gaussian

Figuur 3. Sobel

1.2. Implementatie

Hierbij wordt de methode LoG toegepast met een 9x9(Figuur 4.) kernel. De kernel heeft dezelfde waarde als die vanuit de opdracht. Hierbij wordt elke pixel doorlopen en vervolgens een Convolutie berekening gemaakt met de kernel waarbij alle waardes bij elkaar op worden geteld. Als de totale waarde hoger uitkomt dan 255 wordt de pixel wit gemaakt lager dan 0 zwart en anders krijgt de pixel de totaalwaarde. In figuur 4 vindt u de pseudo code van de implementatie.

Tresholding is een simpele functie waarbij alle pixels boven een bepaalde waarde (bijv. 200) zwart gemaakt worden gemaakt en alles daar onder wit wordt.

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

Figuur 4. LoG 9x9

```
for x = 0 to x < WIDTH do
  for y = 0 to y < HEIGHT do
    if border
      pixels[y * WIDTH + x] = BLACK
      continue

    total = 0
    for logKernelX = -LOG_KERNEL_HALF_SIZE to logKernelX <= LOG_KERNEL_HALF_SIZE do
      for logKernelY = -LOG_KERNEL_HALF_SIZE to logKernelY <= LOG_KERNEL_HALF_SIZE do
        total += pixels[x + logKernelX, y + logKernelY] * LOG_KERNEL[((LOG_KERNEL_HALF_SIZE + lo
+ (LOG_KERNEL_HALF_SIZE + logKernelY) * LOG_KERNEL_SIZE)]

    if total > 255
      pixels[y * WIDTH + x] = BLACK
    else if total < 0
      pixels[y * WIDTH + x] = WHITE
    else
      pixels[y * WIDTH + x] = total
```

Figuur 5. Pseudo code van de implementatie

1.3. Evaluatie

De implementatie wordt getest door de resultaten te vergelijken met de default-settings en door te controleren of alle afbeeldingen succesvol worden gescand door de default-settings.