Implementatieplan Edge Detection

Quinten Perquin & Gregory Rochat - xx-xx-2018

**Doel**

Het doel is om “edge detection” en “thresholding” te realiseren, dit ter ondersteuning van object en gezichtsherkenning in image processing.

**Methoden**

**Laplacian Edge Detector:**

Bij de Laplacian wijze van het detecteren van randen wordt gebruik gemaakt van maar 1 kernel, welke een optelling is van twee masks, een horizontale en een verticale. het berekent in één pass de tweede afgeleide en op basis hiervan randen vinden en benadrukken. (Source/Rewrite)

**Sobel Edge Detector:**

Met de Sobel wijze van rand-detectie worden twee masks gemaakt, één voor horizontaal en één voor verticaal om zo randen en hun directie te vinden en benadrukken. Elke kernel berekent en werkt met de eerste afgeleide. (Source/Rewrite)

**Balanced Histogram Thresholding:**

Bij deze wijze van thresholding wordt uitgegaan dat het plaatje uit twee klassen bestaat, namelijk een voorgrond en een achtergrond. Het weegt deze twee klassen en probeert ze vervolgens te balanceren door staat elementen uit het zwaardere plaatje weg te halen. (Source/Rewrite)

**Otsu's thresholding:**

Bij deze wijze van thresholding wordt uitgegaan dat het plaatje uit twee klassen bestaat, namelijk een voorgrond en een achtergrond. Het weegt deze twee klassen en berekent een balans/middelwaarde aan de hand van het histogram. (Source/Rewrite)

**Basic thresholding:**

Bij deze manier van thresholding wordt er een middelpunt meegegeven om als ''threshold waarde" te gebruiken. Alles onder deze waarde zal zwart worden, alles daarboven wordt wit. Bij deze methode moet de gebruiker zelf bepalen wat de threshold waarde is, wat automatisering minder goed maakt.

**Keuze**

Wij hebben de keuze gemaakt om alle besproken methoden uit te werken. Dit doen wij omdat er dan beter gekeken kan worden welke van hen het beste past bij het gewenste doel.

Tevens kan er dan ook gekozen worden om alle methoden te implementeren en, aan de hand van terugkoppeling vanuit het verdere proces, altijd de beste gekozen worden voor elke situatie.

Onze verwachting is dat de Laplacian Edge Detector de snelste manier van edge detection zal zijn. Basic thresholding zal waarschijnlijk de snelste variant van thresholding zijn, maar is ook het minst geschikt voor automatisering. (Source?)

**Implementatie**

**StudentKernel**

Om een kernel gemakkelijk te kunnen implementeren is er voor gekozen om een Kernel klasse te maken. Aan deze klasse wordt een vector van vectoren van integers meegegeven welke de data bevat voor de kernel. Aan de klasse kan de waarde van een bepaalde positie gevraagd worden aan de hand van een x en een y coordinaat.

**StudentLaplaceEdgeDetector** & **StudentSobelEdgeDetector**

De edge detector methoden worden ieder in een aparte klasse geimplementeerd. Bij het constructen van de klasses wordt de sourceImage en de kernel meegegeven. Met een aparte aanvraag aan de klasses kan dan de Edge Detection methode uitgevoerd worden.

**StudentHistogram**

Om een histogram makkelijk te kunnen maken van

**StudentOtsuThreshold & StudentBasicThreshold & StudentBalancedHistogramThreshold**

De thresholding methoden worden ieder in een aparte klasse geimplementeerd. Bij het constructen van de klasses wordt de source image meegegeven. In de constructor wordt de Histogram klasse aangemaakt voor de source image. Met een aparte aanvraag aan de klasses kan de Thresholding methode uitgevoerd worden.

**Locatie voor implementatie**

De locatie waarin deze implementaties worden aangeroepen is StudentPreProcessing.cpp. Specifiek in de functies: stepEdgeDetection() en stepThresholding()

**Evaluatie**

Voor elke geschreven klasse zal in ieder geval elke methode getest worden op invoer en resultaat.

In het geval van de Edge Detector methoden zal hier gekeken worden naar de nauwkeurigheid van het resultaat a.d.h.v. de meegegeven kernel. Ook zal er een vergelijking gemaakt worden tussen de 2 Edge Detection implementaties qua snelheid.

Voor de Thresholding methoden zal er, aan de hand van visuele feedback, getest worden op de werkbaarheid van de outputs. Er zal dus gekeken worden in hoeverre de vervolgstappen in de face recognition met elke methode’s output om kunnen gaan. Dit gebeurt door een aantal images door alle 3 de methoden te voeren en dan op te slaan. Ook deze methoden zullen getest worden op snelheid.

Voor de Histogram en Kernel classen zal specifiek de opslag van data bekeken worden na invoer en na gebruik.