1. Week 3 26-04-2015 (Edge detection en thresholding)

1.1. Week3 26-05-2015

Gemaakt door Thijs van Tiem & Jos Roijakkers

1.2. Doel

Het doel van dit onderdeel is te kijken naar de verschillende algoritmes en methodes die er gebruikt kunnen worden voor edge-detection. Hiervoor hebben wij een document gemaakt namelijk: “onderzoek edge-detecion.docx”. Uiteindelijk hebben we de methode uit het onderzoek niet gebruikt maar de laplacian of gaussian methode.

Het totale doel van deze opdracht is dus om meer te weten te komen over edge-detection en thresholding

1.3. Hypothese

Wij verwachten dat de eigen gemaakte code slechter zal functioneren dan die van de leraar maar wel bruikbaar zal zijn. Met bruikbaar zijn bedoelen we dat het mensen zal kunnen herkennen op een correcte wijze. Wel verwachten we dat deze code beter zal functioneren dan de vorige omdat we hier en daar wat tips hebben gekregen van Arno.

1.4. Werkwijze

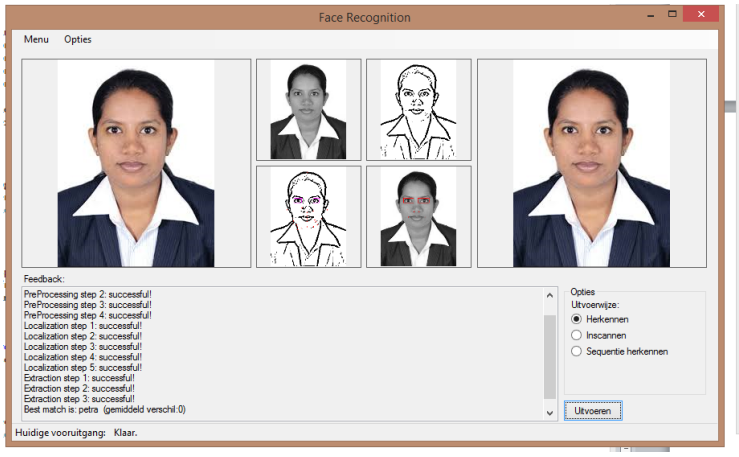
We vergelijken de uitvoer met de code van de leraar met die van de student en kijken naar de verschillen in output bij de herkenning. Hiervoor word gekeken naar de gegeven output van de afbeeldingen en van de terug gegeven waardes. Hierna wordt er gekeken naar de formules en wordt er gekeken naar de originele code.

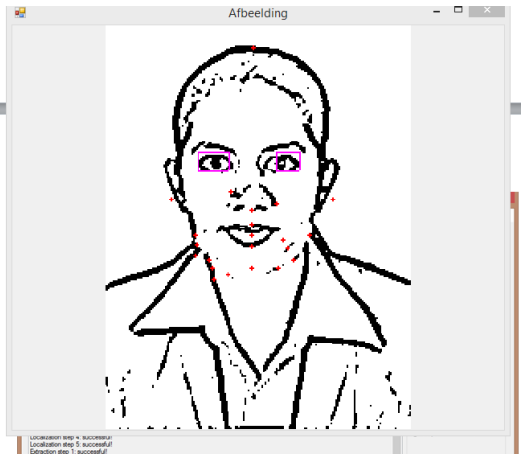
1.5. Resultaten

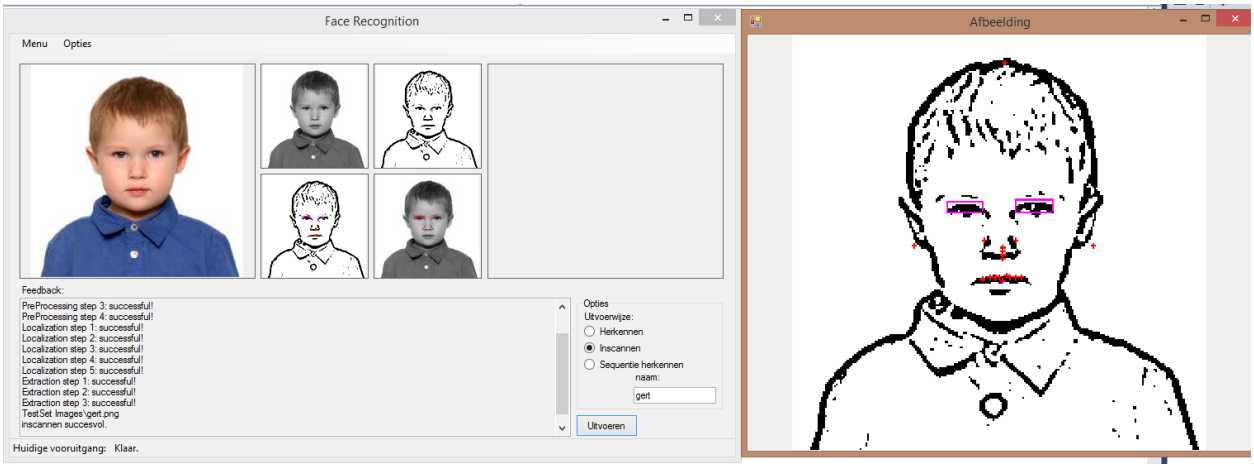
Deze keer wordt er niet naar de verschillende waardes gekeken in de uitvoer maar of de uitvoer van beide thresholds en edge-detections werkelijk hetzelfde is. Hiervoor worden dus meerdere resultaten vergeleken en worden de gebruikte formules nog even toegelicht. (de formules zijn in onderdeel 1.6 te vinden)

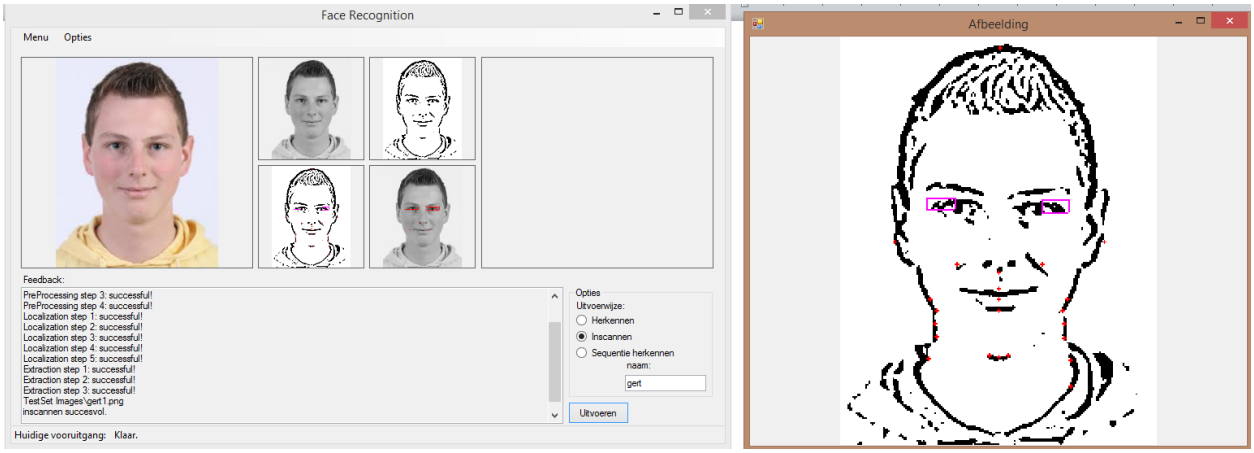
Code leraar:

Hieronder worden verschillende afbeeldingen met de code van de leraar weergeven:





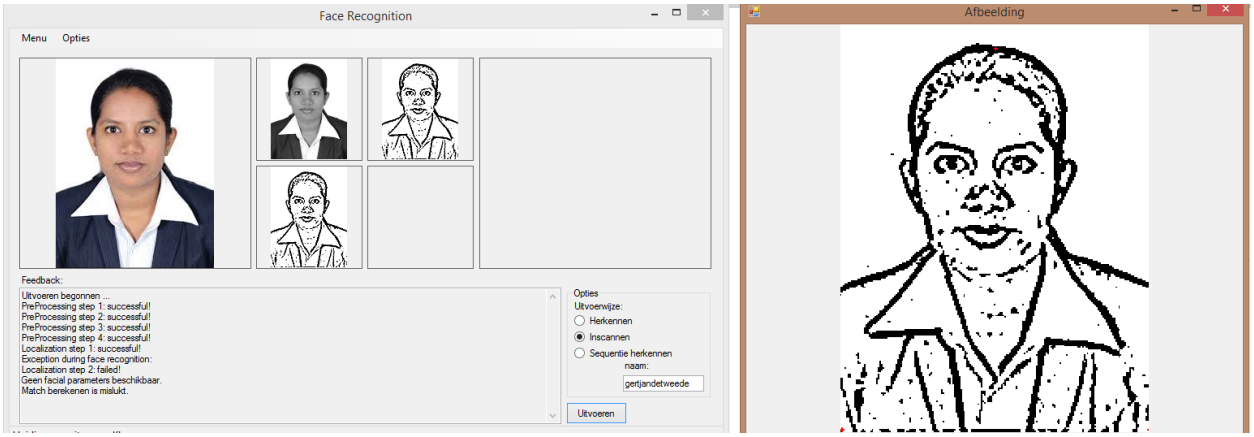


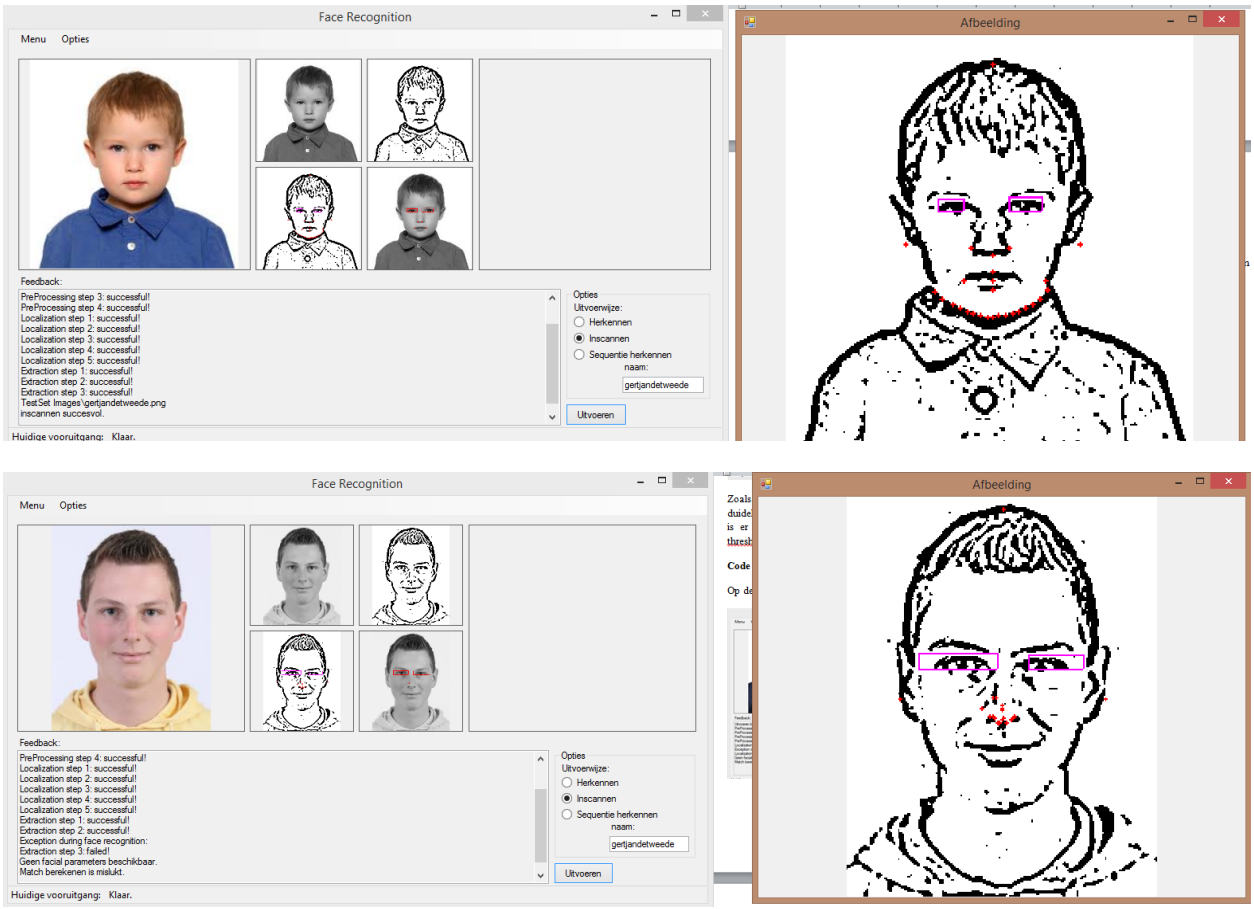


Zoals je bij de leraar ziet is de thresholding heel dun en komen de verschillen tussen randen en niet randen heel duidelijk over. Op deze manier kan er door het programma duidelijk gedetecteerd worden waar de randen liggen en is er ook een duidelijk verschil tussen de randen en andere verschillen. Prima om te gebruiken als Vision thresholding voor gezichtsherkenning.

Code Leerling:

Op de volgende pagina worden de zelfde weergaves als die van de leraar weergeven alleen met onze thresholding.





Zoals je hier gaat het maar op een van de drie afbeeldingen goed met onze thresholding/edge-detection. Dit komt omdat onze edge-detection te snel verschillen in randen detecteert en dit dik verwerkt in de lijnen. Op deze manier weet de gezichtsherkenning niet meer wat nu de werkelijke randen zijn en zal de gezichtsherkenning hier op falen.

Uit deze resultaten valt dus tijdelijk te concluderen dat onze edge-detection te precies werkt voor het vision algoritmen

1.6. Verwerking

Het programma werkt maar 1/3 van de keren met ons algoritmen. Zoals hierboven al werd benoemd komt dit door onze thresholding. De lijnen zijn te dik en snel gemaakt om een echt gezicht te herkennen. De belichting en schaduwen van het plaatje bepalen dus als snel of een afbeelding zal werken of niet. Het huidige algoritmen is dus te gevoelig en dus niet geschikt voor gezichtsherkenning.

Onze formule werkt als volgt:

Eerst werken wij met pre-processing volgens de formule uit de pdf van Arno (zie afbeelding hieronder) :

1. Het maken van een *edge detection* methode

Edge detection is een erg belangrijk onderdeel van de *pre-processing*. Het zorgt ervoor dat de *object recognition* veel soepeler verloopt. In de applicatie wordt op dit moment gebruik gemaakt van de volgende (9x9) *Laplacian operator* met als *kernel*.

$$kernel = \begin{matrix} & & & 1 & 1 & 1 & & & \\ & 0 & & 1 & 1 & 1 & & 0 & \\ & & & 1 & 1 & 1 & & & \\ 1 & 1 & 1 & -4 & -4 & -4 & 1 & 1 & 1 \\ 1 & 1 & 1 & -4 & -4 & -4 & 1 & 1 & 1 \\ 1 & 1 & 1 & -4 & -4 & -4 & 1 & 1 & 1 \\ & & & 1 & 1 & 1 & & & \\ & 0 & & 1 & 1 & 1 & & 0 & \\ & & & 1 & 1 & 1 & & & \end{matrix}$$

De afbeelding zoals hierboven hebben wij verwerkt in de volgende formule :

const int StudentPreProcessing::kernel5x5[5][5] =

{ { 2, 4, 5, 4, 2 }, // = 17
{ 4, 9, 12, 9, 4 },
{ 5, 12, 15, 12, 5 },
{ 4, 9, 12, 9, 4 },
{ 2, 4, 5, 4, 2 }, }; // =17
// = 159

const int StudentPreProcessing::LapKernel5x5[5][5] =

{ { -1, -1, -1, -1, -1, },
{ -1, -1, -1, -1, -1, },
{ -1, -1, 24, -1, -1, },
{ -1, -1, -1, -1, -1, },
{ -1, -1, -1, -1, -1 }}; //laplacian kernel is het gewicht altijd 0

Uiteindelijk hebben we dan een edge-detection die aan de hand van de onderliggende afbeelding word toegelicht:

Hij pakt een pixel en gaat daaromheen kijken naar e omliggende pixels en probeert de randen te detecteren zet alles daaropvolgend in een intensity image. Om te voorkomen dat de pixels buiten het plaatje gebruikt worden, wordt er een offset gebruikt om zo alleen de pixel binnen het plaatje te gebruiken.

Bij de thresholding gebeurd er het volgende:

Er wordt aan de hand van de Otsu methode een pixel gepakt. Wanneer de gezeten waarden aan de hand van Otsu formule hoger is dan de waarde van de Otsu methode word deze op zwart gezet en wanneer hij hieronder zit word hij op het wit gezet. Naderhand word dit omgedraaid en heb je dus de edges. Helaas werkt de Otsu methode tegoed voor edges. Hij zet bij ons de waarden te laag waardoor hij te snel een edge detecteert. De Otsu methode is dus niet geschikt voor dit vision algoritmen.

1.7. Conclusie

Onze edge-detection is niet bruikbaar voor gezichtsherkenning vanwege zijn precisie in het scheiden. Hij vindt namelijk bijna alle edges terwijl hij alleen de randen van het gezicht moet hebben. Voor de rest zijn we tevreden over het resultaat maar hadden wij graag gehad dat hij alle afbeeldingen had kunnen uitvoeren.

1.8. Evaluatie

We zijn aan de hand van onze hypothese zoals hierboven genoemd tevreden met onze resultaten ook al functioneert deze code zeker niet beter dan de vorige codes. In tegendeel is hij voor gezichtsherkenning juist slechter maar functioneert de edge wel prima!