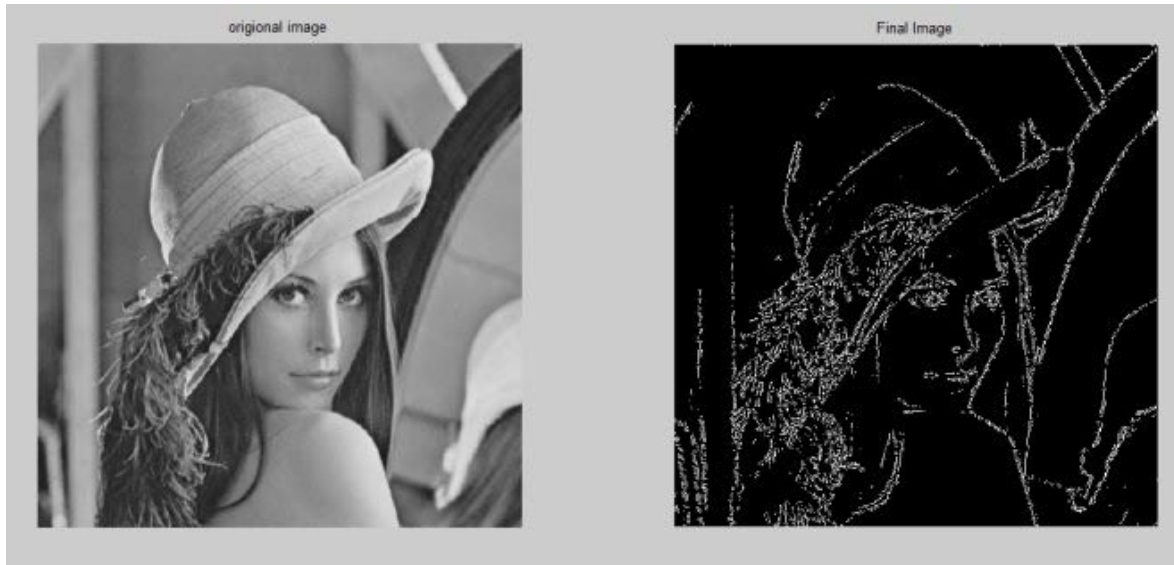


Meetrapport Edge Detection



Menno van der Jagt

Faizal Supriadi

14-02-2020

Inhoudsopgave

Inhoudsopgave	2
Doel	3
Hypothese	3
Werkwijze	4
Snelheid Experiment	4
Geheugen Experiment	9
Resultaten	14
Snelheids Resultaten	14
Geheugen Resultaten	16
Verwerking	18
Snelheids verwerking	18
Geheugen verwerking	19
Conclusie	20
Snelheids Conclusie	20
Geheugen experiment	20
Evaluatie	20

1.1. Doel

Het doel van de experimenten is om te ontdekken welke van de drie edge detection methodes het snelste is, of de hardware invloed heeft en hoeveel geheugen er gebruikt wordt per methode. Hieruit zullen we dan tenslotte proberen te concluderen wat er goed is aan welke methode.

1.2. Hypothese

Op basis van het onderzoek dat wij gedaan hebben om een implementatieplan op te stellen, zijn Sobel en Prewitt de snelste methodes en gebruiken ze ook minder geheugen, dit gaat dan wel weer ten koste van de juistheid van het ontdekken van edges. Verder verwachten we dat de Laplacian methode meer gevoelig is voor de hardware dan de Sobel en Prewitt methode.

1.3. Werkwijze

We zullen hier een duidelijk overzicht maken van hoe we de experimenten hebben uitgevoerd en hoe ze herhaald kunnen worden. We hebben drie experimenten hiervoor uitgevoerd, waarvan er twee erg gerelateerd aan elkaar zijn alleen net iets anders uitgevoerd. We zullen twee experimenten uitvoeren om te bewijzen welke het snelste is.

Snelheid Experiment

Het eerste experiment is een handmatige test waarbij we elke foto tien keer opnieuw laten runnen voor alle drie de methodes. We hebben hiervoor gekozen, omdat het misschien net iets andere resultaten zal opleveren omdat elke keer de applicatie wordt geopend en gesloten. Dit experiment wordt alleen op een computer uitgevoerd, omdat het anders heel veel tijd kost.

Om alle methodes een gelijke kans te geven, maken ze ook allemaal gebruik van dezelfde threshold.

De code ziet er zo uit:

Prewitt:

```
auto start = high_resolution_clock::now();

cv::Mat imageContainer;
HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

cv::Mat prewittx = (cv::Mat_<float>(3, 3) << 1, 0, -1, 1, 0, -1, 1, 0, -1 );
cv::Mat prewitty = (cv::Mat_<float>(3, 3) << 1, 1, 1, 0, 0, 0, -1, -1, -1 );

cv::Mat prewittxResult;
cv::Mat prewittyResult;

filter2D(imageContainer, prewittxResult, -1, prewittx, cv::Point(-1, -1));
filter2D(imageContainer, prewittyResult, -1, prewitty, cv::Point(-1, -1));

IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();

cv::Mat weighted;
cv::addWeighted(prewittxResult, 3, prewittyResult, 3, 0, weighted, -1);

HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);

auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
std::cout << "-----" << duration.count() << std::endl;

return edgeDetectionImage;
```

Sobel:

```
auto start = high_resolution_clock::now();

cv::Mat imageContainer;

HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

cv::Mat sobelx = (cv::Mat_<float>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
cv::Mat sobely = (cv::Mat_<float>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

cv::Mat sobelxResult;
cv::Mat sobelyResult;

filter2D(imageContainer, sobelxResult, -1, sobelx, cv::Point(-1, -1));
filter2D(imageContainer, sobelyResult, -1, sobely, cv::Point(-1, -1));

IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();

cv::Mat weighted;
cv::addWeighted(sobelxResult, 3, sobelyResult, 3, 0, weighted, -1);

HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);

auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
std::cout << "-----" << duration.count() << std::endl;

return edgeDetectionImage;
```

Laplacian:

```
auto start = high_resolution_clock::now();

cv::Mat OverHillOverDale;

HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, OverHillOverDale);

cv::Mat ThoroughBushThoroughBrier = (cv::Mat_<float>(9, 9) << 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);

cv::Mat OverParkOverPale;

filter2D(OverHillOverDale, OverParkOverPale, CV_8U, ThoroughBushThoroughBrier, cv::Point(-1, -1), 0, cv::BORDER_DEFAULT);
IntensityImage* ThoroughFloodThoroughFire = ImageFactory::newIntensityImage();

HereBeDragons::NoWantOfConscienceHoldItThatICall(OverParkOverPale, *ThoroughFloodThoroughFire);

auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
std::cout << "-----" << duration.count() << std::endl;

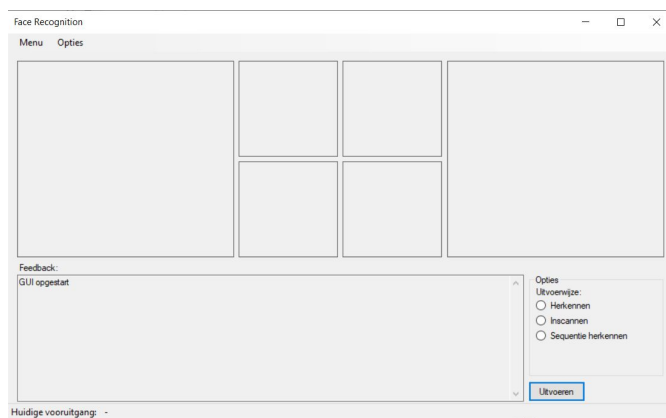
return ThoroughFloodThoroughFire;
```

We maken hiervoor gebruik van de library chrono uit de c++ standaard, deze zal zodra de functie edge detection wordt aangeroepen een start punt opvragen, en aan het einde het volgende punt opvragen en dat dan min elkaar doen om de tijdsduur te meten.

We kunnen nu de applicatie uitvoeren, dit doen we door op start te drukken. Dat is te vinden boven aan het beeld in visual studio.



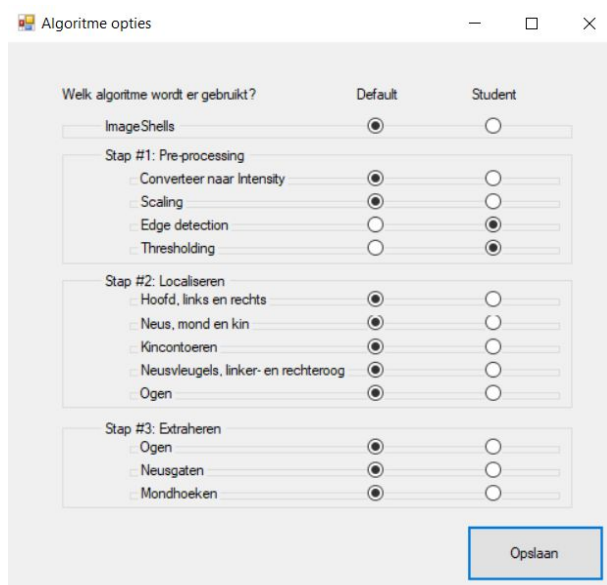
Als we dat gedaan hebben, start de applicatie en krijgen we een scherm die er zo uitziet:



Vervolgens willen we een foto selecteren, dat doen we door op Menu te klikken en vervolgens op “Kies Afbeelding” dat zal daarna nog een scherm openen waar onze testsets in staan

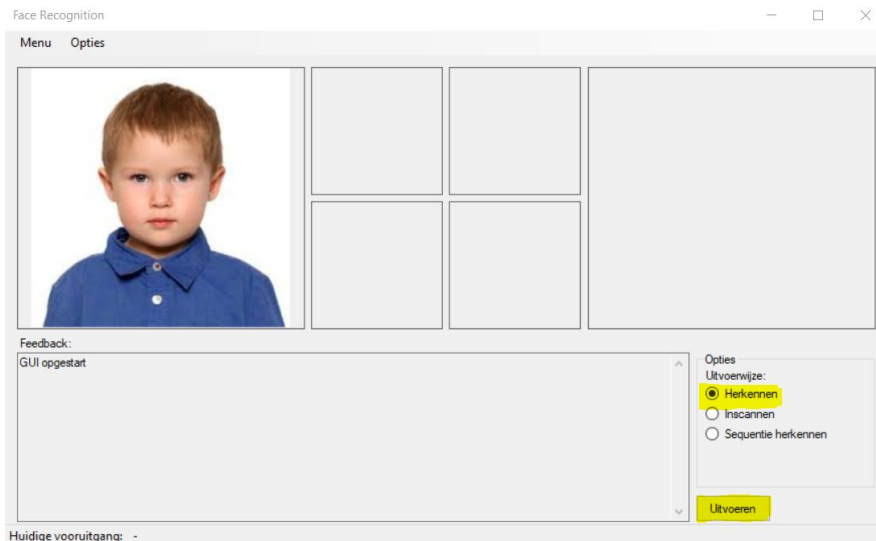


Als we een van de foto's hebben gekozen gaan we naar de opties en kiezen we de algoritme opties

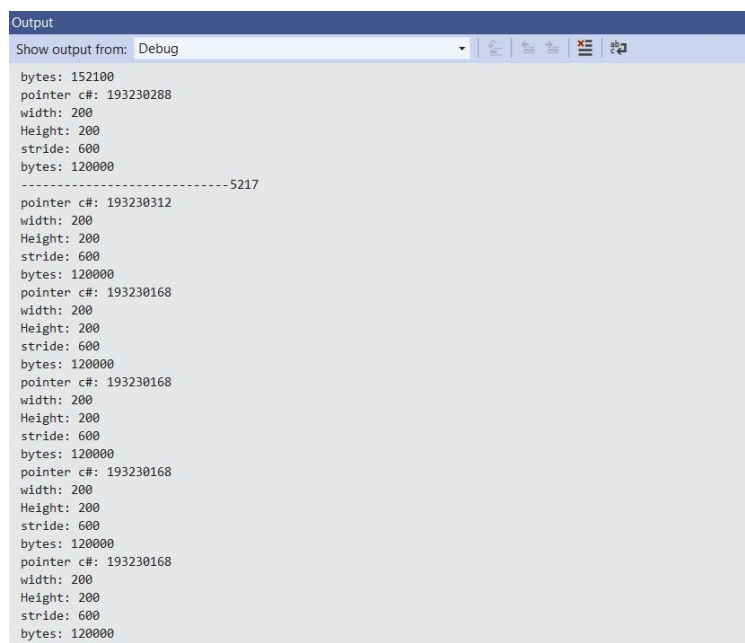


In dat nieuwe scherm zorgen we er voor dat de Thresholding in alle gevallen die van ons is, dus de studenten versie. Verder nemen we de studenten edge detection als we sobel of prewitt willen testen en als we laplacian willen testen nemen we de default. Dat slaan we vervolgens op, daarna hoeven

we alleen nog het vakje herkennen te selecteren en op uitvoeren te drukken rechts onderin het beeld zoals te zien is op de afbeelding hieronder.



Daar komt vervolgens een resultaat uit waar we niet echt heel erg in geïnteresseerd zijn, het is nu de bedoeling dat de applicatie wordt gesloten zodat we de gegevens die we nodig hebben te zien krijgen.



Er zal dan een lijst aan getallen zichtbaar horen te worden in visual studio, zoals hierboven te zien is. We zijn alleen geïnteresseerd in een getal en dat is het getal met de stippellijntjes er voor, Dus in dit geval de 5217 zoals in de afbeelding hierboven staat.

Dit wordt dus handmatig tien keer per foto per methode uitgevoerd, dus 210 keer. Dat zetten we vervolgens in tabellen per foto met een gemiddeld resultaat er onder, zodat we kunnen vergelijken welke foto's het beter doen bij welke methode.

Het tweede experiment zal soortgelijk verlopen alleen hoeven we het geen 210 keer uit te voeren, maar per computer een keer per foto. Hiervoor hebben we een net iets andere code geschreven. Het staat nu in een for loop en zal 1000 keer runnen. Er kan niet vanuit gegaan worden dat de resultaten

hetzelfde zullen zijn als in het eerste experiment, omdat de applicatie niet constant gesloten en geopend wordt, wat mogelijk invloed zou kunnen hebben op het eerste experiment.

De code voor dit experiment ziet er als volgt uit

Prewitt:

```
IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();
microseconds totalDuration = milliseconds(0);

for (int i = 0; i < 1000; i++) {
    auto start = high_resolution_clock::now();

    cv::Mat imageContainer;
    HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

    cv::Mat prewittx = (cv::Mat_<float>(3, 3) << 1, 0, -1, 1, 0, -1, 1, 0, -1);
    cv::Mat prewitty = (cv::Mat_<float>(3, 3) << 1, 1, 0, 0, 0, -1, -1, -1, -1);

    cv::Mat prewittxResult;
    cv::Mat prewittyResult;

    filter2D(imageContainer, prewittxResult, -1, prewittx, cv::Point(-1, -1));
    filter2D(imageContainer, prewittyResult, -1, prewitty, cv::Point(-1, -1));

    cv::Mat weighted;
    cv::addWeighted(prewittxResult, 3, prewittyResult, 3, 0, weighted, -1);

    HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);

    auto stop = high_resolution_clock::now();
    totalDuration += duration_cast<microseconds>(stop - start);
}

std::cout << "-----" << totalDuration.count() / 1000 << std::endl;

return edgeDetectionImage;
```

Sobel:

```
IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();
microseconds totalDuration = milliseconds(0);

for (int i = 0; i < 1000; i++) {
    auto start = high_resolution_clock::now();

    cv::Mat imageContainer;

    HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

    cv::Mat sobelx = (cv::Mat_<float>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
    cv::Mat sobely = (cv::Mat_<float>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

    cv::Mat sobelxResult;
    cv::Mat sobelyResult;

    filter2D(imageContainer, sobelxResult, -1, sobelx, cv::Point(-1, -1));
    filter2D(imageContainer, sobelyResult, -1, sobely, cv::Point(-1, -1));

    cv::Mat weighted;
    cv::addWeighted(sobelxResult, 3, sobelyResult, 3, 0, weighted, -1);

    HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);

    auto stop = high_resolution_clock::now();
    totalDuration += duration_cast<microseconds>(stop - start);
}

std::cout << "-----" << totalDuration.count() / 1000 << std::endl;

return edgeDetectionImage;
```


Laplacian:

```
IntensityImage* ThoroughFloodThoroughFire = ImageFactory::newIntensityImage();
microseconds totalDuration = milliseconds(0);

for (int i = 0; i < 1000; i++) {

    auto start = high_resolution_clock::now();

    cv::Mat OverHillOverDale;

    HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, OverHillOverDale);

    cv::Mat ThoroughBushThoroughBrier = (cv::Mat_<float>(9, 9) << 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1,
        -4, -4, -4, 1, 1, 1, 1, 1, -4, -4, -4, 1, 1, 1, 1, 1,
        -4, -4, -4, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
        1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0);
    cv::Mat OverParkOverPale;

    filter2D(OverHillOverDale, OverParkOverPale, CV_8U, ThoroughBushThoroughBrier, cv::Point(-1, -1), 0, cv::BORDER_DEFAULT);

    HereBeDragons::NoWantOfConscienceHoldItThatICall(OverParkOverPale, *ThoroughFloodThoroughFire);

    auto stop = high_resolution_clock::now();
    totalDuration += duration_cast<microseconds>(stop - start);
}

std::cout << "-----" << totalDuration.count() / 1000 << std::endl;

return ThoroughFloodThoroughFire;
```

Om te kunnen kijken hoeveel invloed de hardware er op heeft hebben we gebruik gemaakt van drie computers met verschillende specificaties. Waarbij computer 1 over het algemeen het sterkste is gevolgd door computer 2 en daarna computer 3.

Computer 1: intel i7-7700HQ cpu met een kloksnelheid van 2.80 GHz, het heeft 16.0 GB geheugen met een snelheid van 2400 MHz en de grafische kaart is een NVIDIA GeForce GTX 1050.

Computer 2: Intel i3-6100 met een kloksnelheid van 3.70 GHz, het heeft 8 GB geheugen met een snelheid van 2133MHz en de grafische kaart is NVIDIA GeForce GTX 1050

Computer 3: intel i3-5005u met een kloksnelheid van 2.00 GHz, het heeft 4.0 GB geheugen met een snelheid van 1600 MHz en de grafische kaart is een intel HD Graphics 5500

Om er ook nog achter te komen hoeveel invloed de hardware op de snelheid van de edge detection methodes heeft zullen we per methode en foto moeten kijken naar wat het resultaat is per computer. Dit resultaat zal het verschil tussen de tijd per computer zijn, vervolgens zullen we dit vergelijken tussen de methodes, waardoor we te zien krijgen welke methode veel beïnvloed wordt door de hardware en welke er minder om geeft.

Geheugen Experiment

Het experiment is een handmatige test waarbij we elke foto tien keer opnieuw laten runnen voor alle drie de methodes. We hebben hiervoor gekozen, omdat het misschien net iets andere resultaten zal opleveren omdat elke keer de applicatie wordt geopend en gesloten. Dit experiment wordt alleen op een computer uitgevoerd, omdat het anders heel veel tijd kost. De hiervoor vernoemde computer 2 wordt hiervoor gebruikt.

Computer 2: Intel i3-6100 met een kloksnelheid van 3.70 GHz, het heeft 8 GB geheugen met een snelheid van 2133MHz en de grafische kaart is NVIDIA GeForce GTX 1050

Om alle methodes een gelijke kans te geven, maken ze ook allemaal gebruik van dezelfde threshold.

De code ziet er zo uit:

Sobel

```
cv::Mat imageContainer;
HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

cv::Mat sobelx = (cv::Mat_<float>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
cv::Mat sobely = (cv::Mat_<float>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

cv::Mat sobelxResult;
cv::Mat sobelyResult;

filter2D(imageContainer, sobelxResult, -1, sobelx, cv::Point(-1, -1));
filter2D(imageContainer, sobelyResult, -1, sobely, cv::Point(-1, -1));

cv::Mat weighted;
cv::addWeighted(sobelxResult, 3, sobelyResult, 3, 0, weighted, -1);

IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();

HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);
```

Prewitt

```
cv::Mat imageContainer;
HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);

cv::Mat prewittx = (cv::Mat_<float>(3, 3) << 1, 0, -1, 1, 0, -1, 1, 0, -1);
cv::Mat prewitty = (cv::Mat_<float>(3, 3) << 1, 1, 1, 0, 0, 0, -1, -1, -1);

cv::Mat prewittxResult;
cv::Mat prewittyResult;

filter2D(imageContainer, prewittxResult, -1, prewittx, cv::Point(-1, -1));
filter2D(imageContainer, prewittyResult, -1, prewitty, cv::Point(-1, -1));

cv::Mat weighted;
cv::addWeighted(prewittxResult, 3, prewittyResult, 3, 0, weighted, -1);

IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();

HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);

return edgeDetectionImage;
```

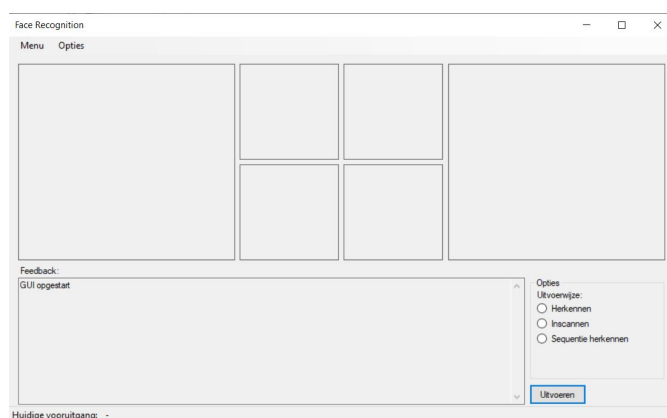
Laplacian

```
cv::Mat imageContainer;  
HereBeDragons::HerLoveForWhoseDearLoveIRiseAndFall(src, imageContainer);  
  
cv::Mat prewittx = (cv::Mat_<float>(3, 3) << 1, 0, -1, 1, 0, -1, 1, 0, -1);  
cv::Mat prewitty = (cv::Mat_<float>(3, 3) << 1, 1, 1, 0, 0, 0, -1, -1, -1);  
  
cv::Mat prewittxResult;  
cv::Mat prewittyResult;  
  
filter2D(imageContainer, prewittxResult, -1, prewittx, cv::Point(-1, -1));  
filter2D(imageContainer, prewittyResult, -1, prewitty, cv::Point(-1, -1));  
  
cv::Mat weighted;  
cv::addWeighted(prewittxResult, 3, prewittyResult, 3, 0, weighted, -1);  
  
IntensityImage* edgeDetectionImage = ImageFactory::newIntensityImage();  
HereBeDragons::NoWantOfConscienceHoldItThatICall(weighted, *edgeDetectionImage);  
  
return edgeDetectionImage;
```

We kunnen nu de applicatie uitvoeren, dit doen we door op start te drukken. Dat is te vinden boven aan het beeld in visual studio.



Als we dat gedaan hebben, start de applicatie en krijgen we een scherm die er zo uit ziet:



Vervolgens willen we een foto selecteren, dat doen we door op Menu te klikken en vervolgens op "Kies Afbeelding" dat zal daarna nog een scherm openen waar onze testsets in staan



Als we een van de foto's hebben gekozen gaan we naar de opties en kiezen we de algoritme opties

Algoritme opties

Welk algoritme wordt er gebruikt?	Default	Student
ImageShells	<input checked="" type="radio"/>	<input type="radio"/>
Stap #1: Pre-processing		
Converteer naar Intensity	<input checked="" type="radio"/>	<input type="radio"/>
Scaling	<input checked="" type="radio"/>	<input type="radio"/>
Edge detection	<input type="radio"/>	<input checked="" type="radio"/>
Thresholding	<input type="radio"/>	<input checked="" type="radio"/>
Stap #2: Localiseren		
Hoofd, links en rechts	<input checked="" type="radio"/>	<input type="radio"/>
Neus, mond en kin	<input checked="" type="radio"/>	<input type="radio"/>
Kincontouren	<input checked="" type="radio"/>	<input type="radio"/>
Neusvleugels, linker- en rechteroog	<input checked="" type="radio"/>	<input type="radio"/>
Ogen	<input checked="" type="radio"/>	<input type="radio"/>
Stap #3: Extraheren		
Ogen	<input checked="" type="radio"/>	<input type="radio"/>
Neusgaten	<input checked="" type="radio"/>	<input type="radio"/>
Mondhoeken	<input checked="" type="radio"/>	<input type="radio"/>

Opslaan

In dat nieuwe schermje zorgen we er voor dat de Thresholding in alle gevallen die van ons is, dus de studenten versie. Verder nemen we de studenten edge detection als we sobel of prewitt willen testen en als we laplacian willen testen nemen we de default. Dat slaan we vervolgens op, daarna hoeven we alleen nog het vakje herkennen te selecteren en op uitvoeren te drukken rechts onderin het beeld zoals te zien is op de afbeelding hieronder.

Face Recognition

Menu Opties

Feedback:

GUI opgestart

Huidige vooruitgang: -

Opties

Uitvoerwijze:

☒ Herkennen

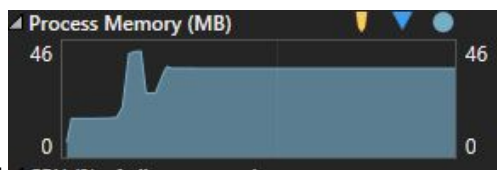
☐ Inscannen

☐ Sequentie herkennen

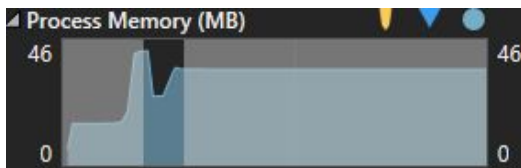
Uitvoeren

Nadat de resultaten zijn gemaakt kijken we naar

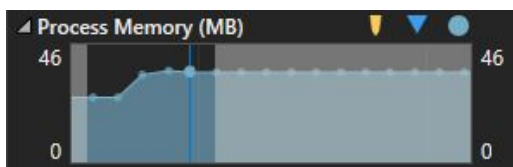
de diagnostic tools en specifiek naar de process memory



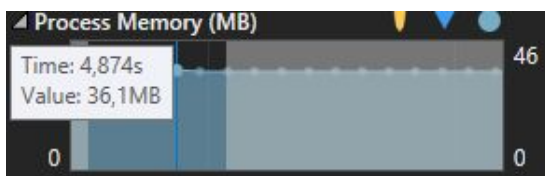
Hier kunnen we specificeren welke regio we willen zien. In ons geval willen we de regio zien waar het herkennen is gestart en geëindigd.




en daar zoomen we in totdat we puntjes zien. Het begin van het herkennen kunnen we uitmaken door de opwaartse heuvel. Hierbij zullen we dus het begin van de heuvel tot twee punten verder kijken om dan te zien wat voor waarden er op ongeveer de helft en op het eind zijn.



Uit deze drie punten kunnen we kijken wat het hoogste value getal is en stoppen dat in een tabel.



Dan drukken we op het restart knopje die in Visual Studio is inbegrepen en doen we dit 10 keer. 

Nadat we dat 10 keer hebben gedaan per foto, zullen we het gemiddelde berekenen en van dat gemiddelde zullen we berekenen wat daarvan het gemiddelde is voor de methoden en in percentages vergelijken.

1.4. Resultaten

Snelheids Resultaten

De snelheids experiment levert deze resultaten op:

Dit is zijn de handmatige resultaten voor de snelheids experiment per foto, uitgevoerd vanaf computer 1:

child-1.png			female-1.png		
Sobel	Prewitt	Laplacian	Sobel	Prewitt	Laplacian
20363	16076	12686	12199	12703	10481
10796	9472	12543	6592	6615	16234
12722	8359	12667	6573	12526	11585
11308	9733	9656	8508	9772	13319
14021	8866	10910	15422	12150	13000
6865	10097	12403	10918	7087	8305
11305	6486	8735	9414	12529	12603
13233	6852	8452	14273	7394	10939
7159	8967	7130	8088	6759	15329
12670	10308	7594	9961	10452	6701
12044,2	9521,6	10277,6	10194,8	9798,7	11849,6

female-2.png			female-3.png		
Sobel	Prewitt	Laplacian	Sobel	Prewitt	Laplacian
3218	8373	4195	8920	7347	6901
4100	4156	2972	6772	10638	7443
3532	3597	6625	8172	15512	13681
3325	5354	2822	11984	12164	6703
3107	2923	2940	12903	6804	7803
7113	3232	2897	12165	12161	6745
2874	3543	5043	11410	14332	6770
4740	3614	3454	12203	18584	8680
3197	2899	2803	18927	8473	6695
7037	5038	3644	7659	6494	9576
4224,3	4272,9	3739,5	11111,5	11250,9	8099,7

male-1.png			male-2.png		
Sobel	Prewitt	Laplacian	Sobel	Prewitt	Laplacian
14248	12494	12012	6315	4688	5810
7027	12123	12585	7132	5321	5348
8741	14888	15919	17076	4896	5081
8152	11923	6764	12172	5174	4992
9381	12318	10366	8462	4410	5041
21143	9961	7800	7787	4893	4960
7030	8174	13215	12259	4721	5003
13353	12594	12178	12007	5302	4899
7358	7062	15530	8701	4921	4660
10138	20152	8019	8064	5012	6169
10657,1	12168,9	11438,8	9997,5	4933,8	5196,3

male-3.png		
Sobel	Prewitt	Laplacian
12266	4425	4923
9552	4727	5059
6864	4856	6234
14084	4687	4925
12132	4710	4891
6485	5067	4858
10031	4767	4886
6506	4461	5810
11969	5350	5209
13425	5070	5219
10331,4	4812	5201,4

De resultaten in deze tabel zijn in microseconden, waarbij de computers dezelfde specificaties hebben als in de werkwijze staat beschreven (computer 1 is het meest krachtige en computer 3 het minste)

Computer:	Computer 1			Computer 2			Computer 3		
Methode:	Sobel	Prewitt	Laplacian	Sobel	Prewitt	Laplacian	Sobel	Prewitt	Laplacian
child-1.png	5502	5383	5555	4889	4866	4785	11578	11991	12466
female-1.png	5321	5165	5618	5004	4856	4839	11403	11882	12289
female-2.png	2167	2175	2314	2138	2245	2038	4914	5190	5338
female-3.png	5167	5153	5614	5018	5188	5141	11403	11800	12381
male-1.png	5371	5146	5539	5186	5361	5261	11569	11672	12502
male-2.png	5088	5177	5578	4841	4893	5259	11691	11669	12407
male-3.png	5128	5135	5530	5029	4584	5245	11536	11891	12090

Geheugen Resultaten

Het geheugen experiment levert deze resultaten op:

De resultaten in deze tabellen zijn in megabytes.

	child-1.png					female-1.png		
	Sobel	Prewitt	Laplacian			Sobel	Prewitt	Laplacian
	42,2	41,1	41,5			45,1	46,1	46,2
	41,5	42,2	40,9			46,1	46,5	46,4
	41,1	41,4	42			46,3	45,8	46,2
	41,9	41,7	41,3			46,3	46,9	46,3
	42,5	42,2	41,7			46,5	46,4	46,1
	41	41,8	42,2			46,1	46,6	46,7
	40,8	41,5	41,4			45,7	46	45,8
	41,3	42,3	40,9			46,6	46,3	46,1
	40,9	41,5	41,7			46,2	45,9	46,8
	41,6	41,7	41,5			45,8	46,1	46,2
Gemiddelde:	41,48	41,74	41,51		Gemiddelde:	46,07	46,26	46,28
	female-2.png					female-3.png		
	Sobel	Prewitt	Laplacian			Sobel	Prewitt	Laplacian
	44,6	44,8	44,8			46,7	45,1	46,3
	44,2	45,2	45,1			47	46,5	46,3
	44,4	44,7	44,8			46,1	46,9	47,1
	44,5	44,7	44,8			45,8	45,2	46,5
	44,3	44,4	44,9			45,8	45,9	46,6
	44	44,6	45			46,2	46,7	46,3
	44,6	45	43,9			45,3	46,7	46,7
	44,6	44,1	44,6			45,8	45,9	47
	44,3	44,4	44,7			45,6	46,6	45,9
	44,6	44,2	44,2			46,2	46,2	46,8
Gemiddelde:	44,41	44,61	44,68		Gemiddelde:	46,05	46,17	46,55

	male-1.png					male-2.png		
	Sobel	Prewitt	Laplacian			Sobel	Prewitt	Laplacian
	46,1	45	45,5			45,5	46,3	45,8
	45,9	46,3	46,9			45,2	46,8	46,3
	45,1	45,9	46			46,2	47,2	46,7
	46,5	45,6	46,9			45,7	46,8	46,1
	46,1	46	46,8			46,4	45,2	46
	46,2	46,7	46,2			46,3	45,9	46,3
	45,1	45,7	47			46,1	45,3	45,9
	45,6	45,9	46,5			46,7	46,3	45,5
	46,2	45,3	46,1			45,5	45	46,6
	46,4	46,2	46,7			46,4	45,3	46,3
Gemiddelde:	45,92	45,86	46,46		Gemiddelde:	46,00	46,01	46,15
	male-3.png							
	Sobel	Prewitt	Laplacian					
	45,9	45,7	45,7					
	45,6	46,3	45,8					
	45,5	46,4	45,8					
	46,1	46	46,9					
	45,9	46,7	45,6					
	45,3	46,2	46					
	46,2	46	46,5					
	45,7	46,5	46,1					
	46,7	45,6	46,8					
	45,4	45,9	46,7					
Gemiddelde:	45,83	46,13	46,19					

1.5. Verwerking

Snelheids Verwerking

Uit de eerste handmatige meting is te zien dat de tijden heel erg variëren en dat de tijden soms zelfs te hoog wordt voor wat het zou horen te zijn. Dit hebben we meerdere malen uitgeprobeerd en elke keer kregen we een ander resultaat, met veel hoge pieken er in. Na een tijdje werd het duidelijk dat het opnieuw opstarten van de computer erg veel invloed had op de snelheid die we per keer kregen. Naarmate we de applicatie meerdere malen hadden geopend en gesloten werd het soms steeds trager. We kunnen met deze cijfers dus weinig concluderen, omdat erg afhankelijk is van op welk moment ze worden uitgevoerd.

De resultaten van het tweede snelheids experiment hebben we omgezet naar een gemiddelde percentage. Deze percentages zijn berekend door het gemiddelde te nemen van de resultaten die horen bij de computers en methodes per afbeelding.

	Sobel sneller dan Laplacian	Prewitt sneller dan Laplacian	Sobel sneller dan Prewitt
child-1.png	3,80991397	2,544964029	1,233556375
female-1.png	4,685198822	3,848787837	0,8054123711
female-2.png	5,109013993	0,8324661811	4,241240916
female-3.png	7,170650361	4,493925297	2,561608301
male-1.png	5,315014011	5,063348212	0,2395371961
male-2.png	7,511563367	6,923041538	0,5504162812
male-3.png	5,402664454	5,807496529	-0,3826119025
Gemiddeld percentage:	5,572002711	4,216289946	1,321308505

Zo is “Sobel sneller dan Laplacian” berekent als volgt in een excel sheet, soortgelijke formules zijn gebruikt voor de andere vergelijkingen.

Waarbij de x de methode is en het cijfer erachter aangeeft welke computer het is, datzelfde geldt voor de y waarbij de y de andere methode is en het cijfer erachter aangeeft welke computer het is. Dus in het geval van “Sobel sneller dan Laplacian” stelt de x de Laplacian methode voor en y de Sobel methode voor.

$$(((x1+x2+x3)/3) / ((y1+y2+y3)/3) * 100) - 100$$

Het gemiddelde percentage is heel simpel berekend door alle zeven percentages die onder elkaar staan bij elkaar op te tellen en te delen door zeven.

Om vervolgens te kunnen kijken wat het verschil tussen de computers zijn hebben we besloten om nog een tabel te maken, die duidelijker laat zien wat het verschil in percentage tussen de methodes per computer is.

Computer:	Computer 1		Computer 2		Computer 3	
vergelijking:	Sob/Lap	Prew/Lap	Sob/Lap	Prew/Lap	Sob/Lap	Prew/Lap
child-1.png	0,9632860778	3,195244288	-2,127224381	-1,664611591	7,669718432	3,961304312
female-1.png	5,581657583	8,770571152	-3,29736211	-0,3500823723	7,769885118	3,425349268
female-2.png	6,783571758	6,390804598	-4,677268475	-9,220489978	8,628408628	2,851637765
female-3.png	8,651054771	8,946244906	2,451175767	-0,9059367772	8,576690345	4,923728814
male-1.png	3,127909142	7,636999611	1,446201311	-1,865323634	8,064655545	7,111034955
male-2.png	9,630503145	7,745798725	8,634579632	7,480073574	6,124369173	6,324449396
male-3.png	7,839313573	7,692307692	4,295088487	14,41972077	4,802357836	1,673534606
gemiddeld:	6,082470864	7,196852996	0,9607414615	1,127621427	7,376583582	4,324434159

Hiervoor hebben we een vergelijkbare berekening gedaan als bij de vorige tabel alleen in dit geval per computer, dus de berekening ziet er als volgt uit:

$$(x / y) * 100 - 100$$

Waarbij de x laplacian moet voorstellen en y sobel of prewitt is, we hebben besloten om in dit geval sobel en prewitt niet met elkaar te vergelijken, omdat de ze beide op dezelfde manier werken alleen een net iets andere kernel.

Geheugen Verwerking

	Sobel	Prewitt	Laplacian
Gemiddelde	45,10857143	45,25428571	45,40285714

$$(a + b + c + d + e + f + g) / 7$$

Waar de letters het gemiddelde waarde per foto betekenen voor elke methode.

Memory Vergelijkingen	Sobel minder dan Prewitt	Sobel minder dan Laplace	Prewitt minder Laplacian
Procent minder	0,3230301495	0,6523942235	0,3283035545

$$(x / y) * 100 - 100$$

Waar x Prewitt of Laplacian kan zijn en y Sobel of Laplacian kan zijn. Met deze tabel kan worden gezien hoeveel minder geheugen de ene methode gebruikt in vergelijking met de andere methode.

1.6. Conclusie

Snelheids Conclusie

Uit de verwerkingen van 1.5 kunnen we nu dus de methodes gaan vergelijken in snelheid. Zo zien we dat Sobel gemiddeld 5,57% (afgerond) sneller is dan Laplacian en dat Prewitt 4,22% (afgerond) sneller is dan Laplacian. Dit laat duidelijk zien dat Laplacian slomer is dan Sobel en Prewitt. Verder kunnen we zien dat Sobel een klein beetje sneller is dan Prewitt, alleen is dit heel erg afhankelijk van twee foto's die ervoor hebben gezorgd dat er een groter verschil is tussen Sobel en Prewitt, we kunnen dus niet echt concluderen of Sobel ook echt sneller is dan Prewitt hiervoor zou het experiment meerdere malen moeten worden uitgevoerd om een specifiek resultaat te krijgen.

Geheugen Conclusie

Uit de verwerkingen van 1.5 kunnen we nu dus de methodes gaan vergelijken in geheugen. Bij Sobel zien we dat het gemiddeld 0.32% (afgerond) minder geheugen gebruikt in vergelijking met Prewitt en dat het 0.65% (afgerond) minder geheugen gebruikt dan Laplacian. Prewitt is dan wel 0,33% (afgerond) sneller dan Laplacian. Met deze gegevens kan worden veronderstelt dat de Laplacian methode meer geheugen gebruikt dan bij de andere methoden en dat Sobel het minst geheugen gebruikt in vergelijking met Prewitt en Laplacian.

1.7. Evaluatie

Onze doel is dat de edge detection methode sneller en efficiënter wordt. Met daarbij onze hypothese, Prewitt en Sobel zijn sneller dan de Laplacian methode.

Uit de conclusie kunnen we trekken dat Prewitt 4,22% en Sobel 5,572002711% sneller zijn dan de Laplacian methode. Dit betekent dat ons doel over snelheid gelukt is..

Ons andere doel, efficiëntie, is ook gelukt. Wij hebben in de conclusie gevonden dat de Sobel en Prewitt methode minder geheugen gebruiken dan de Laplacian(default) methode. Sobel neemt 0,65% minder geheugen en Prewitt 0.33% minder geheugen in vergelijken met Laplacian. Daardoor zal, wanneer de Sobel en Prewitt methode worden gebruikt in plaats van de Laplacian, het programma efficiënter verlopen.

We adviseren wel om nog verder onderzoek te doen naar de mogelijkheden om de snelheid verschillen te kunnen testen tussen de verschillende methodes, aangezien de metingen erg gevarieerd kunnen zijn met onze test. Maar dit geldt ook voor het geheugen onderzoek. Een verdere onderzoek hiervoor zou kunnen gaan over hoe het onderzoek geautomatiseerd kan worden gemaakt, zodat er sneller meer data kan worden gemaakt.