1. Implementatieplan RGBtoInstensity

Jeremy ruizenaar 09-03-17

1.1. Doel onderzoek

Het doel van deze implementatie is het omzetten van een RGBimage naar een Intensityimage. Dit moet efficiënt en snel kunnen gebeuren. Verder moet dit algoritme bruikbaar zijn binnen het vision project geleverd door de HU.

1.2. Bruikbare algoritmes

Er zijn verschillende bruikbare algoritmes mogelijk op dit probleem op te lossen:

1. Averaging

Bij averaging wordt er per pixel een linear gemiddelde genomen tussen de drie kleur kanalen

2. Luminance correction

Bij luminance correction worden per pixel de drie kleur kanalen gewogen naar gevoeligheid voor het menselijk zicht. De coefficienten waarmee de kleur kanalen gewogen worden zijn 0.3*R + 0,59*g + 0,11*b, deze formule heeft als uitkomst die intensiteit.

3. Desaturation

Bij desaturation wordt per pixel een RGB waarde omgezet in een HSL waarde. De hue wordt gezet op een willekeurige kleur en de saturation wordt op nul gezet. De lightness kan berekend worden door het volgende algoritme lightness =(max (R, G, B) + min (R, G, B)) / 2

4. Decomposition

Bij decomposition kan er onderscheid gemaakt worden tussen maximum en minimum decomposition. Bij maximum decomposition wordt per pixel de intensity gelijk aan max (R, G, B), en bij minimum decomposition wordt de intensity gelijk aan min (R, G, B).

5. Single color channel

Bij single color channel wordt de intensiteit per pixel gelijk aan de waarde van één van de drie kleur kanalen.

1.3. Keuze

Er is gekozen voor de single-color-channel algoritme. Dit algoritme zou het snelst moeten zijn omdat alleen de instructie (gray = pixel.g) uitgevoerd word.

ook is er gekozen voor het average algoritme omdat dit ook een simpel algoritme is. hierbij wordt een lineair gemiddelde genomen tussen de RGB waardes.

Verder komt het luminance-correction algortime aan de orde. Dit is een algoritme wat ook snel zou moeten werken maar voor een cpu iets complexer is. Dit vanwege meerdere floating-point operations per instructie.

1.4. Implementatie

Single color channel algoritme

```
for (int x = 0; x < image.getWidth(); x++) {
	for (int y = 0; y < image.getHeight(); y++) {
		RGB pixel = image.getPixel(x, y);
		int grayValue = pixel.g;
		IntensityImg->setPixel(x*image.getHeight() +y,Intensity(grayValue));
	}
}
```

Luminance correction algoritme

```
for (int x = 0; x < image.getWidth(); x++) {
	for (int y = 0; y < image.getHeight(); y++) {
		RGB pixel = image.getPixel(x, y);
		int grayValue = (0.3*pixel.r + 0.59*pixel.g + 0.11*pixel.b);
		IntensityImg->setPixel(x*image.getHeight() +y,Intensity(grayValue));
	}
}
```

average algoritme

```
for (int x = 0; x < image.getWidth(); x++) {
	for (int y = 0; y < image.getHeight(); y++) {
		RGB pixel = image.getPixel(x, y);
		int grayValue = (pixel.r + pixel.g + pixel.b) / 3;
		IntensityImg->setPixel(x*image.getHeight() +y,Intensity(grayValue));
	}
}
```

1.5. Evaluatie

De tijd die het algoritme in beslag neemt om een image om te zetten zal gemeten worden over 100000 iteraties. Verder wordt de kwaliteit van de image vergeleken met het menselijk oog en zal er gekeken worden of de afbeelding bruikbaar is in vervolg stappen van het face-recognition process. Ook zal er een histogram van de afbeeldingen bijgevoegd zijn.