# Implementatieplan Jip en Tim

## Namen en datum

Teamlid 1: Jip Galema

Teamlid 2: Tim IJntema

Datum: 17-2-2017

## Doel

**Kwestie:**

We hebben gezichtsherkennings implementatie gekregen voor het vak vision. Deze is echter niet snel/efficiënt dan gewenst. Om deze sneller en efficiënt te maken hebben we gekozen om een deel van dit proces te verbeteren, wat te maken heeft met het inlezen van een plaatje en dit omzetten in een plaatje bestaande uit grijstinten

**Ideale situatie:**

Een snelle, efficiënt en werkende inlees functionaliteit voor een plaatje daarnaast ook een goede omzetting van een rgb-waarde naar een grayscale waarde.

## Methoden

Je geeft hieraan welke methoden er zijn, wat de verschillende tussen de methodes zijn, en wat de voor en nadelen hiervan zijn

Er zijn verschillende manieren om een image op te kunnen staan:

* Een 1D array

**Over het algemeen werkt een 1D array het snelste maar moet wel een vaste waarde hebben wat met verschillende grootte voor images wel een ingewikkeld iets wordt**

* Een 2D array

**Een 2D array is veel langzamer dan een 1D array, dit is echter voor een programmeur makkelijker te gebruiken**

* Een vector

**Een vector is bijna hetzelfde als een 1D array, het heeft echter een extra voordeel namelijk: een vaste grootte hoeft niet gegeven te worden. En daarnaast is het bijna net zo snel als de standaard manier**

Bronnen:

<http://stackoverflow.com/questions/17259877/1d-or-2d-array-whats-faster>

<http://lemire.me/blog/2012/06/20/do-not-waste-time-with-stl-vectors/>

Er zijn ook verschillende manieren om een image op te slaan.

* Op de heap

**De heap is een stuk langzamer dan de stack, maar geeft wel de ruimte om de grootte van de array eventueel te vergroten**

* Op de stack

**De stack is een stuk sneller dan de heap, de stack zit namelijk in de CPU-cache. Een nadeel is echter dat je met de stack vastzit aan een grootte. Dit betekent dat de array niet groter of kleiner worden gemaakt als dit nodig is.**

Bronnen:

<http://stackoverflow.com/questions/5836309/stack-memory-vs-heap-memory>

Wij hebben online 5 methodes gevonden voor het opzetten van een rgb-waarde naar een grijstint:

* Het gemiddelde nemen van de 3 kleuren

**Is een snelle methode die niet veel geheugen kost. Wordt geïmplementeerd als:**

**Gray = (Red+ Green + Blue) /3**

**Deze optie is goed toe te passen met een look-up tabel en is dus makkelijk dynamisch te maken.**

**De nadelen hebben vooral met de type grijstinten te maken. Deze optie is namelijk niet zo goed in het weergeven van verschillende soorten grijs en de helderheid van deze tinten.**

* correctie voor het menselijk oog (“Luminance”)

**Deze methode maakt gebruik van het feit dat mensen bepaalde kleuren sterker zien. De formule hiervoor is:**

**Gray = (Red \* 0.3 + Green \* 0.59 + Blue \* 0.11)**

**Dit kost wat meer reken operaties maar is nog steeds best wel efficiënt**

* Desaturation

**Zorgt voor een donker grijsachtige tint. Maar hierbij wordt het HSL color space gebruikt, er zou dus een soort omzettig moeten plaatsvinden (hier zullen we nog wat research voor moeten doen) maar dit kan voor extra rekentijd zorgen voor het programma.**

* Decomposition, Het minimum of maximum van de de rgb waardes pakken

**Pak of de minimale waarde van de rgb en gebruik die als grijswaarde (snap dit nog niet helemaal)**

* Enkele kleur als grayscale

**De simpele manier om de grijstinten te bepalen is door 1 kleur rgb waarde te nemen en die te gebruiken als grijswaarde**

Bronnen:

<http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/>

## Keuze

Voor de imageshell hebben we besloten om in eerste instantie een vector te gebruiken. Dit heeft als voordeel dat de array goed gealloceerd en gedeblokkeerd wordt. Dit is ook een veilige optie (weinig kans op geheugen dat overschreven wordt)

Voor het opzetten van RGB-waardes hebben we in eerste instantie gekozen voor “Luminance”. Maar we gaan een aantal logische omzettingen testen voor efficiëntie en de hoeveelheid geheugen die ze innemen. Hierop zullen we nog een keer kijken op we de goede keuze hebben gemaakt.

## Implementatie

Voor de implementatie van het project hebben we op dit moment voor veiligheid gekozen. Dit betekent dat we met een vector werken, hierdoor weten we zeker dat deallocatie en allocatie van ruimte goed gaat.

Voor de conversie hebben we besloten om dit in eerste instantie niet dynamisch te programmeren. We zullen een operator= gebruiken om een rgb object om te kunnen zetten naar een grijstint object. Deze operator= zal bestaan uit een for-loop die voor elke pixel zijn nieuwe grijswaarde berekent.

## Evaluatie

De volgende testen zouden we graag uit willen voeren:

* Welke grijstinten omzetting is het handigste (kost het minste tijd en geheugen)
* Hoe groot is het verschil tussen een vector of een array op de heap (tijd en geheugen gebruik).

Om onze implementatie te bewijzen:

* Tijd meten van methode in de implementatie en eigengemaakte methode
* Plaatjes onderling vergelijken voor preciezere herkenning van gezichtseigenschappen