Practicum Vision: Face Recognition

Project Handleiding

Auteurs:
Rolf Smit
Alexander Hustinx

Versie:
1.0

Cursus:
Vision (TCTI-V2VISN1-13)

Inhoud

1 Inleiding

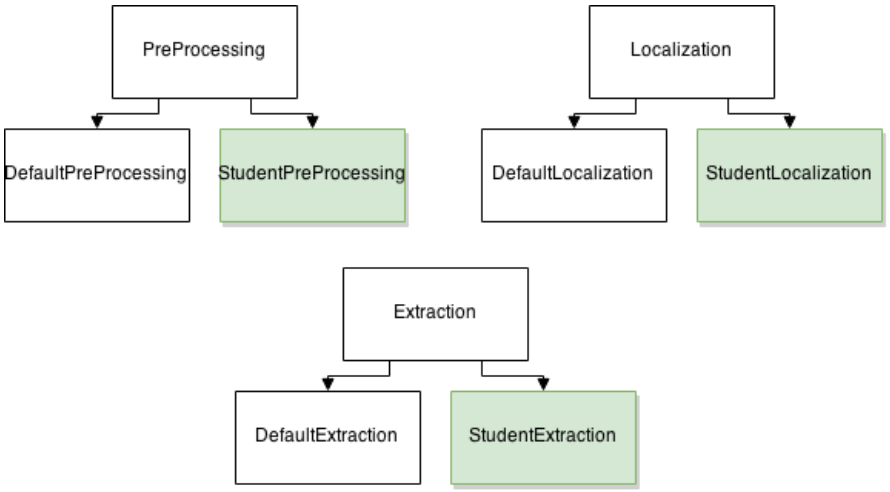
In deze handleiding worden een aantal zaken omtrent het face recognition project uitgelegd. Er zal onder meer uitgelegd worden wat de in- en output van bepaalde methodes is, op welke manier afbeeldingen geladen en opgeslagen kunnen worden en hoe de globale opbouw van het project er uit ziet. Wat niet in de handleiding opgenomen is zijn zaken zoals het installeren van Visual Studio en het programmeren in C++.

2 Project opbouw

Het *face recognition* practicum project bestaat uit een tweetal sub-projecten waarvan er slechts één echt van belang is voor het practicum, namelijk het DLL project. Het andere project, de GUI, is enkel van belang voor uitgebreid testen of het gemakkelijk switchen tussen *face recognition* stappen die je zelf geprogrammeerd hebt en die standaard mee geleverd zijn. Om een globaal idee te krijgen hoe dit project er precies uit ziet en waarom volgt hier een beknopte beschrijving:

Face recognition is een complex vision probleem. Een werkend *face recognition* programma schrijven zonder op de schouders van reuzen te staan zou veel te veel tijd in beslag nemen en past niet in één practicum. Toch is er voor gekozen een *face recognition* practicum te doen omdat de algoritmen, denkstappen en technieken erg interessant en leerzaam zijn. Om *face recognition* niet te groot en te ingewikkeld te maken is het opgedeeld in een aantal kleine stappen. Van deze stappen gaat de student er een aantal zelf programmeren. Vervolgens kan de student een werkend *face recognition* programma bouwen door de stappen die zelf geprogrammeerd zijn aan te vullen met de standaard mee geleverde stappen.

Om het bovenste gemakkelijk te realiseren is er een abstracte laag in het project gebouwd. Elke stap heeft dus een eigen abstracte basis klasse of methode. Voor elke van deze abstracte klassen of methoden is een werkende implementatie gegeven en een lege implementatie die door de student ingevuld kan worden. Het figuur hieronder illustreert deze abstractie voor de drie hoofdstappen van face recognition.



Naast de abstractie in de face recognition klassen is er ook abstractie aangebracht in de image klasse die gebruikt wordt. Er moet namelijk ook een eigen implementatie van een image klassen worden geschreven.

3 DLL en GUI

Het DLL project dankt zijn naam aan het feit dat het project gecompileerd kan worden naar een DLL bestand, dit DLL bestand kan door de GUI gebruikt worden (wanneer geplaats in dezelfde map) om de algoritmen uit te voeren. **Dit betekent niet dat je constant een DLL compileert en je code test in de GUI!** In de opties van Visual Studio kan namelijk geswitcht worden tussen het compileren van een DLL en een executabel. Het DLL project heeft namelijk gewoon een main entry point waar vanuit code gemakkelijk te testen is (eventueel zelfs met behulp van een *heuse* debugger).

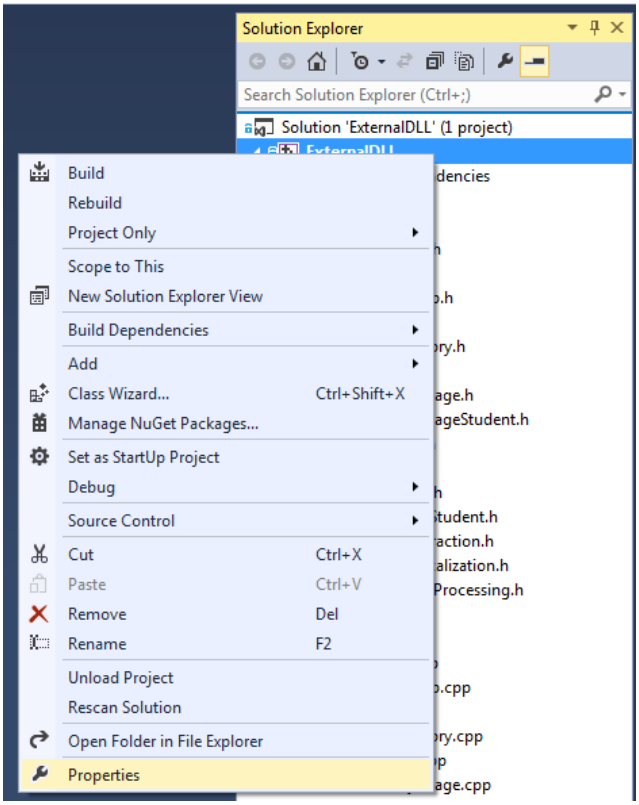
Er kan als volgt geswitcht worden tussen DLL en executable.

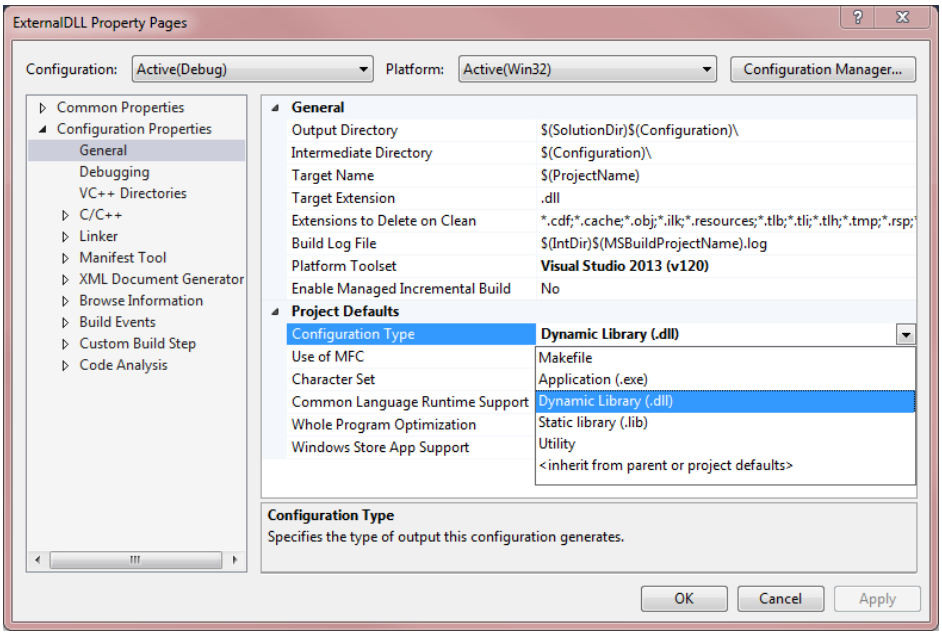
1. **Rechtermuisknop op het project in de Solution Explorer;**

2. **Klik op Properties;**

3. **Linker venster:**
   Open het kopje Configuration Properties en klik op General;

4. **Rechter venster:**
   Onder het kopje Project Defaults staat een optie Configuration Type. Zet deze optie op DLL of executable.





4 Heavy lifting

Bij de cursus Vision draait het allemaal om algoritmen, algoritmen waarmee data uit een afbeelding gehaald kan wordt of algoritmen die afbeeldingen manipuleren. Waar het vooral niet om draait is bijvoorbeeld het laten zien van een afbeelding in een GUI of het laden en opslaan van afbeeldingen. Om er voor te zorgen dat jullie geen dagen kwijt zijn aan het schrijven van een PNG-decoder en encoder zijn er in het DLL project klassen en methoden opgenomen die dit voor jullie doen. Hier onder volgt een lijst.

- **ImageIO**
  Klasse met verschillende functies voor het laden, opslaan en het tonen van afbeeldingen.
- **Image**
  Abstracte klasse als basis voor alle image klassen deze klasse heeft zelf geen enkele informatie over pixels en kent enkel de breedte en hoogte van de afbeelding.
- **RGBImage**
  Abstracte klasse die de Image klasse *inherit* en een aantal abstracte methoden toevoegt voor het aanpassen van pixels in het RGB kleurmodel.
- **IntensityImage**
  Abstracte klasse die de Image klasse *inherit* en een aantal abstracte methoden toevoegt voor het aanpassen van pixels in het intensity/*grayscale* kleurmodel.
- **ImageFactory**
  Bevat methoden om een nieuwe afbeeldingen te maken. Deze klasse is cruciaal om te gebruiken wanneer je wilt kunnen switchen van je eigen image klasse naar de standaard image klasse. De ImageFactory returnt bij het maken van een afbeelding altijd de base klasse (abstracte klasse). De standaard *face recognition* stappen gebruiken de ImageFactory bijvoorbeeld zodat een andere image klasse gemakkelijk getest kan worden zonder overal de code aan te passen.
- **Main**
  Main entry point, bevat een test setup om alle stappen achterelkaar uit te kunnen voeren
- **Point2D<type>**
  Klasse voor het werken met punten in twee dimensies: x en y.
- **Feature**
  Een feature is een punt of een reeks punten die een gezicht identificeren. Bij de lokalisatie stappen van het *face recognition* proces is het de bedoeling dat er Feature's worden gevonden worden.
- **FeatureMap**
  Een FeatureMap wordt in bijna elke *face recogniton* stap meegestuurd, deze map bevat alle features die tot nu toe gevonden zijn. Stappen die nieuwe features vinden moeten deze toevoegen aan de FeatureMap.

5 Features

Bij *face recognition* wordt er gezocht naar *facial features* die vervolgens door middel van berekening omgezet kunnen worden naar *facial parameters*. Een *facial feature* of feature is een punt, of een reeks van punten die een locatie of meerdere locaties op het gezicht beschrijven.

5.1 Feature

In de klasse `Feature` zijn een aantal constanten opgenomen die alle features identificeren. Een `Feature` object bestaat uit een van de constanten en een aantal punten.

Hieronder volgt een tabel van alle beschikbare feature constanten:

Feature constante	Punten	Uitleg
`FEATURE_HEAD_TOP`	1	Bovenkant van het hoofd horizontaal gecentreerd.
`FEATURE_HEAD_LEFT_SIDE`	1	Punt dat horizontaal tegen de linkerkant van het hoofd ligt en verticaal ergens tussen de ogen en het neus einde.
`FEATURE_HEAD_RIGHT_SIDE`	1	Punt dat horizontaal tegen de rechterkant van het hoofd ligt en verticaal ergens tussen de ogen en het neus einde.
`FEATURE_NOSE_BOTTOM`	1	Onderkant van de neus, horizontaal gecentreerd tussen de zijkanten van het hoofd.
`FEATURE_MOUTH_TOP`	1	Bovenkant van de mond, horizontaal gecentreerd tussen de zijkanten van het hoofd.
`FEATURE_MOUTH_BOTTOM`	1	Onderkant van de mond, horizontaal gecentreerd tussen de zijkanten van het hoofd.
`FEATURE_MOUTH_CENTER`	1	Het midden van de mond, verticaal gecentreerd tussen FEATURE_MOUTH_TOP en FEATURE_MOUTH_BOTTOM en horizontaal gecentreerd tussen de zijkanten van het hoofd.
`FEATURE_CHIN`	1	Onderkant van de kin, horizontaal gecentreerd tussen de zijkanten van het hoofd.
`FEATURE_CHIN_CONTOUR`	2-18	Punten van de contoeren van de kin. De punten beginnen aan de linkerkant en eindigen aan de rechterkant van het hoofd. Voor elke kant van het hoofd moeten er 9 punten gevonden worden. In een uiterst geval mogen er punten missen tot minimaal 1 punt per kant van het hoofd. Deze punten dienen om de 10 graden met het midden van de mond als midden van de cirkel gevonden te worden.
`FEATURE_EYE_LEFT_RECT`	2	Een set van twee punten die samen een vierkant om het linker oog vormen. Het

Feature constante	Punten	Uitleg
		eerste punt is de linker bovenhoek van het vierkant, het tweede punt de rechter onderhoek.
FEATURE_EYE_RIGHT_RECT	2	Een set van twee punten die samen een vierkant om het rechter oog vormen. Het eerste punt is de linker bovenhoek van het vierkant, het tweede punt de rechter onderhoek.
FEATURE_NOSE_END_LEFT	1	Een punt dat het meest linker punt van de linker neusvleugel aangeeft.
FEATURE_NOSE_END_RIGHT	1	Een punt dat het meest rechter punt van de linker neusvleugel aangeeft.
FEATURE_NOSTRIL_LEFT	1	Midden van het linker neusgat.
FEATURE_NOSTRIL_RIGHT	1	Midden van het rechter neusgat.
FEATURE_MOUTH_CORNER_LEFT	1	De linker mondhoek, een mondhoek is altijd verticaal uitgelijnd op het punt waar de boven en onderlip elkaar raken.
FEAURE_MOUTH_CORNER_RIGHT	1	De rechter mondhoek, een mondhoek is altijd verticaal uitgelijnd op het punt waar de boven en onderlip elkaar raken.
FEATUER_HEAD_LEFT_NOSE_BOTTOM	1	De linker zijkant van het hoofd ter hoogte van de onderkant van de neus. (Horizontaal zoveel mogelijk afgerond naar de binnenkant van het gezicht.)
FEATURE_HEAD_RIGHT_NOSE_BOTTOM	1	De rechter zijkant van het hoofd ter hoogte van de onderkant van de neus. (Horizontaal zoveel mogelijk afgerond naar de binnenkant van het gezicht.)
FEATURE_HEAD_LEFT_NOSE_MIDDLE	1	Niet benodigd (wordt door de standaard code in de *post-processing* uitgevoerd). Dit punt ligt op de lijn die het midden vormt tussen de ogen en de onderkant van de neus. Op deze lijn markeert dit punt de linkerkant van het hoofd. (Horizontaal zoveel mogelijk afgerond naar de binnenkant van het gezicht.)
FEATURE_HEAD_RIGHT_NOSE_MIDDLE	1	Niet benodigd (wordt door de standaard code in de *post-processing* uitgevoerd). Dit punt ligt op de lijn die het midden vormt tussen de ogen en de onderkant van de neus. Op deze lijn markeert dit punt de rechterkant van het hoofd. (Horizontaal zoveel mogelijk afgerond naar de binnenkant van het gezicht.)

5.2 FeatureMap

Features worden tussen de verschillende *face recognition* stappen uitgewisseld doormiddel van een `FeatureMap`. Een `FeatureMap` is niet meer dan een *hash-map* waarin een `Feature` gestopt kan worden doormiddel van een `Feature` constante. Hieronder volgt een lijst van de belangrijkste methoden.

```
bool hasFeature(const int featureId) const;
void putFeature(Feature &feature);
Feature & getFeature(const int featureId);
```

De methode `hasFeature()` kan gebruikt worden om te controleren of een bepaalde `Feature` aanwezig is in de `FeatureMap`. De methoden `putFeature()` en `getFeature()` kunnen gebruikt worden om een `Feature` aan de `FeatureMap` toe te voegen, bij te werken of om een Feature te verkrijgen. Wanneer er een `featureId` benodigd is wordt hiermee een van de `Feature` constanten bedoelt.

6 Image klassen

Om het gemakkelijk te maken afbeeldingen te laden, opslaan en te tonen worden er een aantal klassen meegeleverd. In dit hoofdstuk wordt kort uitgelegd hoe deze klassen werken.

6.1 Flags & instellingen

In het project zijn er een 3 tal *flags* of instellingen die het gedrag van de code kunnen veranderen. Deze worden bijvoorbeeld gebruikt om te bepalen welke code (student of default) uitgevoerd moet worden.

Opslaan afbeeldingen

Omdat de geschreven code zowel in een GUI (via DLL) als in een *standalone executable* uitgevoerd kan worden, is het niet altijd wenselijk dat (DLL) code afbeeldingen opslaat. Daarom zit er in de ImageIO klasse een *flag* (`ImageIO::isInDebugMode`) die aangeeft of afbeeldingen "echt" opgeslagen moeten worden. Deze boolean *flag* staat standaard op false wat betekend dat afbeeldingen niet opgeslagen worden.

Naast een *flag* om te bepalen of een afbeelding opgeslagen moet worden is er ook een *flag* (`ImageIO::debugFolder`) die bepaald wat het basis path van een afbeelding is. Dit is handig wanneer de code uitgevoerd wordt op een andere computer. De *flag* voor het basis path wordt enkel gebruikt wanneer de methode `ImageIO::getDebugFileName()` gebruikt wordt.

ImageFactory

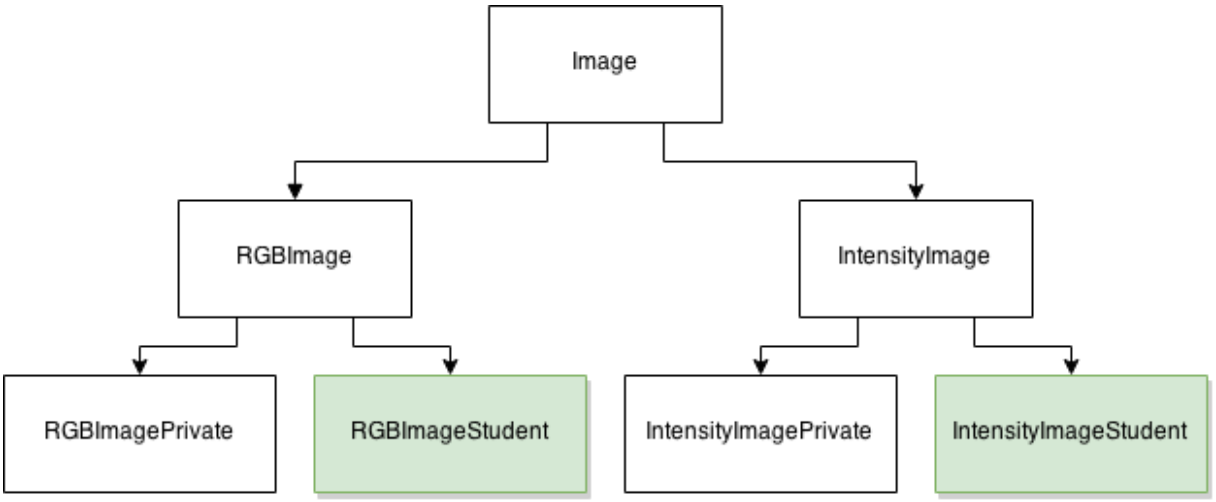
Om te kunnen switchen tussen de default image implementatie en die van de student is er een speciale `ImageFactory` klasse die bepaald welke type (default of student) afbeelding er wordt gealloceert. Doormiddel van de methode `ImageFactory::setImplementation()` kan de implementatie gekozen worden. Er zijn standaard twee mogelijke implementaties: `ImageFactory::DEFAULT` en `ImageFactory::STUDENT`.

Voorbeeld code

```
#include "ImageIO.h"
#include "ImageFactory.h"

int main(int argc, char * argv[]){

	ImageIO::debugFolder = "C:\\Users\\Student\\SomeFolder";
	ImageIO::isInDebugMode = true;

	//Set the current image implementation to default
	ImageFactory::setImplementation(ImageFactory::DEFAULT);

	//Create an empty image and load
	RGBImage * image = ImageFactory::newRGBImage();
	if(!ImageIO::loadImage("C:\\test.jpg", *image)){
		std::cout << "Image load error!" << std::endl;
		return 0;
	}

	//Save the image using the debug path
	ImageIO::saveRGBImage(*image,
		ImageIO::getDebugFileName("debug.png"));
}
```

6.2 Typen

In het project zijn twee soorten afbeeldingen aanwezig namelijk `IntensityImage` en `RGBImage`, beide typen zijn sub klassen van de klasse Image. De Image klasse kent enkel de breedte en hoogte van de afbeelding. De `IntensityImage` en `RGBImage` klasse zijn op hun beurt beide abstract en kennen enkel abstracte methoden voor het lezen en schrijven van pixels in het type `RGB` of `Intensity`. De echte implementatie van de twee soorten afbeeldingen vindt plaats in de `RGBImagePrivate` en `IntensityImagePrivate` klasse. Naast de 'Private' sub klassen van de *RGB* en *Intensity* Image zijn er ook 'Student' sub klassen van de *RGB* en *Intensity* Image. Deze 'Student' klassen zijn helemaal leeg en de implementatie van de verschillende abstracte methoden moeten nog geschreven worden.



Het includen van de ImageIO.h en ImageFactory.h headers is in de meeste gevallen genoeg om gebruik te kunnen maken van de verschillende image klassen.

Belangrijk

Om te kunnen switchten tussen de default image typen en de student image typen wordt aangeraden altijd een nieuwe instantie van een afbeelding te verkrijgen door gebruik te maken van de ImageFactory.

RGB & Intensity

Een `RGBImage` bestaat in het project uit `RGB` pixels. Dit pixel type is gedefinieerd als een *struct* die uit 3 unsigned char's bestaat. Een `IntensityImage` bestaat op zijn beurt uit `Intensity` pixels, een Intensity pixel is in het project gedefinieerd als een typedef unsigned char. Beide pixel typen zijn te vinden in de PixelType.h header file.

6.3 Input & output

De ImageIO klasse kan gebruikt worden voor het laden en opslaan van afbeeldingen. Voor beide image types (Intensity en RGB) zijn er aparte methoden.

6.3.1 Show

De methode `ImageIO::showImage()` kan gebruikt worden om een afbeelding te tonen, deze functie wordt alleen uitgevoerd wanneer de `ImageIO::isInDebugMode` *flag* op `true` staat. De functie accepteerd zowel `IntensityImage` als `RGBImage`.

Voorbeeld:

```
RGBImage * image = ImageFactory::newRGBImage();
if(!ImageIO::loadImage("C:\\test.jpg", *image)){
	std::cout << "Image load error!" << std::endl;
} else {
	ImageIO::showImage(*image);
}
```

6.3.2 Laden

De functie `ImageIO::loadImage()` kan gebruikt worden voor het laden van afbeeldingen. Afbeeldingen worden altijd geladen in het RGB type (grijs/intensity conversie moet zelf geïmplementeerd worden).

De volgende bestands typen worden ondersteunt:

- Windows bitmap (bmp)
- Portable image formats (pbm, pgm, ppm)
- Sun raster (sr, ras)
- JPEG (jpeg, jpg, jpe)
- JPEG 2000 (jp2)
- TIFF files (tiff, tif)
- portable network graphics (png)
- OpenEXR

Voorbeeld:

```
RGBImage * image = ImageFactory::newRGBImage();
if(!ImageIO::loadImage("C:\\test.jpg", *image)){
	std::cout << "Image load error!" << std::endl;
} else {
	std::cout << "Image successfully loaded!" << std::endl;
}
```

6.3.3 Opslaan

De functie `ImageIO::saveRGBImage()` en `ImageIO::loadRGBImage()` kunnen gebruikt worden voor het opslaan van afbeeldingen. Opslaan gebeurt alleen wanneer de `ImageIO::isInDebugMode` *flag* op `true` staat. Aangeraden wordt gebruikt te maken van de `ImageIO::getDebugFileName()` functie voor de bestands naam zodat het pad later gemakkelijk aangepast kan worden. De extensie van de bestandsnaam wordt gebruikt om te bepalen hoe de afbeelding opgeslagen moet worden.

De volgende bestands typen worden ondersteunt:

- JPEG (jpeg, jpg, jpe)
- portable network graphics (png)
- Portable image formats (pbm, pgm, ppm)

Voorbeeld:

```
ImageIO::debugFolder = "C:\\Users\\Student\\SomeFolder";
ImageIO::isInDebugMode = true;

//Save the image using the debug path
ImageIO::saveRGBImage(*image, ImageIO::getDebugFileName("debug.png"));
```

7 Face recognition stappen

De verschillende *face recognition* stappen zijn opgedeeld in drie hoofdstappen met elk een aantal sub-stappen. Elke hoofdstap heeft een eigen abstracte klasse en een standaard (default) implementatie.

Hoofdstappen en sub-stappen:

1. Pre-processing:
    1. Kleur naar grijs;
    2. Schalen;
    3. Edge-detection;
    4. Thresholding.
2. Localization:
    1. Zoeken naar bovenkant en zijkanten hoofd;
    2. Zoeken naar onderkant neus, mond en kin;
    3. Zoeken naar kin contouren;
    4. Zoeken naar neus vleugels, zijkanten hoofd op neus hoogte;
    5. Zoeken naar initiële positie ogen.
3. Extraction:
    - Zoeken naar exacte oog locaties;
    - Zoeken naar exacte locaties van de neus gaten;
    - Zoeken naar exacte locaties van de mond hoeken.

Bij de extraction is de volgorde van uitvoering niet meer van belang omdat de verschillende stappen geen gegevens van elkaar verwachten.

In werkelijkheid zijn er nog 2 extra hoofdstappen, namelijk: Post-processing en Representation. Deze twee stappen zijn echter niet van belang voor vision en daarom weg gelaten uit de code die studenten moeten maken.

7.1 Pre-processing

De *pre-processing* stappen zijn de stappen van het *face recognition* programma waar nog niet gezocht wordt naar informatie maar waar enkel de ingevoerde afbeelding wordt getransformeerd naar een 'bruikbaar formaat'.

7.1.1 Conversie naar grijswaarde (IntensityImage)

Functie	`IntensityImage * stepToIntensityImage(const RGBImage &image);`
Input	Kleuren afbeelding (RGBImage)
Output	Grijswaarden afbeelding (IntensityImage)

7.1.2 Schalen van afbeelding

Functie	`IntensityImage * stepScaleImage(const IntensityImage &image);`
Input	Grijswaaden afbeelding van stap 1 (IntensityImage)
Output	Verkleinde grijswaarden afbeelding (IntensityImage) Er is geen specifiek format waarnaar de afbeelding verkleint moet worden. Er wordt echter gekeken naar de hoeveelheid pixels die een afbeelding heeft, dit moeten er ongeveer 40.000 (200 x 200) zijn. Wanneer een afbeelding meer dan 40.000 pixels heeft moet de afbeelding verkleind worden (zonder de aspect ratio te veranderen) naar het formaat dat het dichts in de buurt van 40.0000 pixels komt. Wanneer een afbeelding kleiner is dan 40.000 pixels moet er een kopie van het origineel als resultaat gegeven worden.

7.1.3 Edge-detection

Functie	`IntensityImage * stepEdgeDetection(const IntensityImage &image);`
Input	Verkleinde afbeelding van stap 2
Output	Edge-detected afbeelding (IntensityImage) Zwart= achtergrond Wit tinten = lijnen

7.1.4 Thresholding

Functie	`IntensityImage * stepThresholding(const IntensityImage &image);`
Input	Edge-detected afbeelding van stap 3
Output	Thresholded en inverse binaire afbeelding. Puur zwart (0) = lijnen Puur wit(255) = achtergrond

7.2 Localization

De lokalisatie stappen worden allemaal uitgevoerd aan de hand van een binair (*thresholded*) *edge detected* image. Het bestand waarin gewerkt wordt is: StudentLocalization.cpp, in dit bestand zijn een 5-tal lege methoden gedefinieerd die elk een eigen invoer en uitvoer hebben. Alle methoden hebben gemeen dat wanneer alle features gevonden zijn de return waarde `true` is en anders `false`.

7.2.1 Bovenkant en zijkanten hoofd

Functie	`bool stepFindHead(const IntensityImage &image, FeatureMap &features);`
Input	-
Output	FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE

7.2.2 Onderkant neus, mond en kin

Functie	`bool stepFindNoseMouthAndChin(const IntensityImage &image, FeatureMap &features);`
Input	FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE
Output	FEATURE_NOSE_BOTTOM FEATURE_MOUTH_TOP FEATURE_MOUTH_CENTER FEATURE_MOUTH_BOTTOM FEATURE_CHIN

7.2.3 Kin contouren

Functie	`bool stepFindChinContours(const IntensityImage &image, FeatureMap &features);`
Input	FEATURE_NOSE_BOTTOM FEATURE_MOUTH_CENTER FEATURE_CHIN
Output	FEATURE_CHIN_COUNTOUR

7.2.4 Neusvleugels en zijkanten hoofd op neus hoogte

Functie	`bool stepFindNoseEndsAndHeadSides(const IntensityImage &image, FeatureMap &features);`
Input	FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE FEATURE_NOSE_BOTTOM FEATURE_MOUTH_CENTER FEATURE_CHIN
Output	FEATURE_NOSE_END_LEFT FEATURE_NOSE_END_RIGHT FEATURE_HEAD_LEFT_NOSE_BOTTOM FEATURE_HEAD_RIGHT_NOSE_BOTTOM

7.2.5 Initiële positie ogen

Functie	`bool stepFindInitialEyes(const IntensityImage &image, FeatureMap &features);`
Input	FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE FEATURE_NOSE_BOTTOM FEATURE_MOUTH_CENTER FEATURE_CHIN FEATURE_NOSE_END_LEFT FEATURE_NOSE_END_RIGHT FEATURE_HEAD_LEFT_NOSE_BOTTOM FEATURE_HEAD_RIGHT_NOSE_BOTTOM
Output	FEATURE_EYE_LEFT_RECT FEATURE_EYE_RIGHT_RECT

7.3 Extraction

De extractie stappen worden in tegenstelling tot de lokalisatie stappen niet allemaal op een binair *edge detected* image uitgevoerd. Het bestand waarin gewerkt wordt aan de extractie stappen is: StudentExtraction.cpp in dit bestand zijn 3 legen methoden gedefinieerd die elke een eigen invoer en uitvoer hebben. Alle methoden hebben gemeen dat wanneer alle features gevonden worden de methode `true` returned en wanneer dit niet het geval is `false`.

7.3.1 Exacte locaties ogen

Functie	`bool stepExtractEyes(const IntensityImage &image, FeatureMap &features);`
Input	Binair edge-detected image FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE FEATURE_EYE_LEFT_RECT FEATURE_EYE_RIGHT_RECT FEATURE_NOSE_END_LEFT FEATURE_NOSE_END_RIGHT FEATURE_NOSE_BOTTOM FEATURE_HEAD_LEFT_NOSE_BOTTOM FEATURE_HEAD_RIGHT_NOSE_BOTTOM
Output (vervang input)	FEATURE_EYE_LEFT_RECT FEATURE_EYE_RIGHT_RECT

7.3.2 Exacte locaties neus gaten

Functie	`bool stepExtractNose(const IntensityImage &image, FeatureMap &features);`
Input	Grayscale image FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE FEATURE_NOSE_END_LEFT FEATURE_NOSE_END_RIGHT FEATURE_NOSE_BOTTOM
Output	FEATURE_NOSTRIL_LEFT FEATURE_NOSTRIL_RIGHT

7.3.3 Exacte locaties mondhoeken

Functie	`bool stepExtractMouth(const IntensityImage &image, FeatureMap &features);`
Input	Grayscale image FEATURE_HEAD_TOP FEATURE_HEAD_LEFT_SIDE FEATURE_HEAD_RIGHT_SIDE FEATURE_MOUTH_BOTTOM FEATURE_MOUTH_TOP FEATURE_MOUTH_CENTER FEATURE_HEAD_LEFT_NOSE_BOTTOM FEATURE_HEAD_RIGHT_NOSE_BOTTOM
Output	FEATURE_MOUTH_CORNER_LEFT FEATURE_MOUTH_CORNER_RIGHT