View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositorio Universidad de Zaragoza

BACHELORARBEIT

Human Recognition and Identification

Identification of Persons in the Social Context based on Image Processing

Submitted to the
Institute for Computer Technology,
Faculty of Electrical Engineering and Information Technology,
TU Wien
in partial fulfilment of the requirements for the degree of
Bachelor of Electrical Engineering and Information Technology

under supervision of

Dr. Nima TaheriNejad
TU Wien
Institute of Computer Technology

and

Dr. Carlos Sagüés Blázquiz
University of Zaragoza
Institute of Computer Technology

by

Patricia Javierre del Río
Matr.Nr. 1528043
Gumpendorferstrasse 39
1060, Wien

13.July.2016 ___________________________

Kurzfassung

Die bestehende Bachelorarbeit hat als Ziel den therapeutischen Roboter TUK mit einer Software für die Erkennung von Menschen zu versehen; sodass er die Leute in seiner Umgebung identifizieren kann. Auβerdem, es ist beabsichtigt, dass er die Person, die mit ihm interagiert, erkennen kann, um damit sein Verhalten anzupassen.

Erstens, die theoretischen Konzepten, die während der Entwicklung der Software benutzt worden sind, werden vorgestellt. Verschiedene Arten von Bildverarbeitungstechniken als auch Klassifizierungalgoritmen werden erklärt. Zweitens, die Implementierung word eingezeigt, um ein allgemeines Sicht des System zu geben.

Die erworbene Ergebnisse werden dargestellt, dazu werden auch einige Probleme analysiert. Abschlieβend, eininge Leitungen, um die Probleme zu lösen werden vorgeschlagt. Auf diese Weise wird die Richtung für weitere Entwicklungen unterbreitet.

Abstract

The aim of this Bachelor's Thesis is to provide the therapeutic robot TUK with an identification software, so that it is able to recognize people in its environment. Moreover, it is intended that TUK can identify who it is interacting with, and thus adapt its behaviour depending on the situation.

In the first part, the theoretical principles that have been used during the development of the software are presented. Several kinds of image processing techniques as well as classification algorithms are explained. In the second part, the implementation is shown step by step in order to give an overview of the whole system.

Finally, the results obtained during several tests are presented and discussed. In conclusion, several guidelines for tackling some of the challenges are proposed, setting a possible way for further work.

Acknowledgements

First of all, I have to acknowledge my supervisor Dr. Nima TaheriNejad for all his advice and support during the whole project. The most important persons in this adventure have been Mum and Dad not only for their unmeasurable support and encouragement, but also because they gave me the wonderful opportunity to develop my Bachelor Thesis in the TU Wien. I would also like to thank all my friends in Spain, especially Marina and Ignacio, who have been with me from the very beginning until the very end, and have trusted me when I didn't.

I would also like to acknowledge my colleague Elena, and, of course, the person who became my family during all this experience, Maria del Carmen, because without her support, it would have been impossible to bring this project to an end. Thank you.

Table of contents

Abbreviations

ANN	Artificial Neural Network
GUFD	Glasgow Unfamiliar Face Database
HOG	Histogram Oriented Gradients
HSV	Hue-Saturation-Value
IDE	Integrated Development Environment
LOG	Laplacian of Gaussian
LBP	Local Binary Patterns
LBPH	Local Binary Patterns Histogram
MIT	Massachusetts Institute of Tehcnology
OpenCV	Open Source Computer Vision Library
RGB	Red-Green-Blue
ROI	Region Of Interest
SFC	Social Face Classification Dataset
SNR	Signal to Noise Ratio
SVM	Support vector Machine
TUK	Technische Universität Kiwi
USC	University of Southern California
VIT	Visually Interactive Toy

1. Introduction

This Bachelor Thesis is a part of the project *Technische Universität Kiwi (TUK).* TUK is conceived as a Visually Interactive Toy (VIT), this is a visually interactive robot that understands and reacts to human presence and visual communication messages. Moreover, TUK is a therapeutic robot thought to help people interact with autistic kids. TUK has to be able to recognize people in the room and realize if the autistic kid is there or not and who is him or her.

1.1 Motivation

People suffering from Autism Spectrum Disorder have high difficulties to communicate with their relatives as well as with people in their environment, which makes their relationships really complicated. With this in mind, TUK is thought as a tool to help autist kids and their families to understand each other and, as a result of it, improve their communication and make their everyday life easier.

TUK could also be very helpful for people specialised in education for autistic kids, as it could be an interesting tool to understand how they feel or what they want to express when they are not able to do it by themselves.

Hence, it follows that TUK is not only an interesting project because of the broad engineering progress that could bring by using image processing techniques to recognize people but also because of its social perspective, as the main purpose of an engineer is to contribute to make a better society with the acquired knowledge.

1.2 Problem Statement

As already stated, TUK is conceived as a visually interactive toy; therefore it has to be able to identify who it is interacting with. A possible way to achieve this goal is presented in this paper using face features that can characterise a person under different conditions.

Based on personal experience, the information that we save about a person the first time we meet him or her is mainly the eyes' colour, the hair colour and the skin tone. These features are going to be used in the present project to identify people in the environment of TUK and therefore make its performance more efficient.

The main idea is to use the present software to identify who is TUK interacting with and thus adapt its behaviour depending on this condition. This will make TUK useful during the therapies.

1.3 Task Setting

The main task of this work is to implement a classifier that can make TUK able to recognize people in its surroundings using features that vary significantly among different people, thus making the classification more accurate.

For this goal, two main steps are defined. First of all, the required features of the given human are extracted using image processing techniques, in the present case, eye colour, skin colour and hair colour, in addition to a face recognition algorithm. Secondly, these features are used to classify the given person using a predefined database.

1.4 Methodology

The IDE Visual Studio 2015 Community together with OpenCV, an open source computer vision library, are going to be used to implement several classifiers in a C++ interface. The hair and the skin are going to be classified into several categories using a Naïve-Bayesian Classifier, whereas a Neural Network is going to be used to categorize the colour of the eye. Finally, a template matching algorithm is used to extract person that is classified from a database.

2. State of the Art and Related Work

Human recognition and identification is a very up-to-date topic, as being accurate could have numerous benefits in the performance of artificial intelligence. As stated for this project, the main purpose of the investigations is to be able to adapt robots' behaviour according to the person they are interacting with, thus accuracy can make them work more efficiently.

Within this field, face recognition has aroused a significant interest, fueled by potencial applications as well as by algorithmic techniques and inexpensive computers having enough computational power to run these algorithms. The majority of papers report an outstanding result, usually more than 95%, on limited-size databases (usually less than 50 individuals) [1].

2.1 Face Recognition

Face recognition error rates have decreased over the last twenty years by three orders of magnitude when recognizing frontal faces in still images taken under controlled conditions [2]. Besides, many systems for the application of border-control and smart biometric identification have been implemented [3]; however, these systems have shown to be sensitive to various factors, such as lighting, expression, occlusion and aging; thus their performance is deteriorated when trying to recognize people in uncontrolled environments [3].

Cutting-edge face verification methods use hand-crafted features, that are often combined to improve performance [4]. Deep neural networks have also been applied in face detection, face alignment and face verification. In an unconstrained environment, Local Binary Patterns (LPB) used as input showed an improvement when combined with traditional methods [5].

Metric methods are also often used in face verification, often cooperating with task-specific objectives [6]. Currently, the most successful system that uses a large data set of labelled faces employs a clever transfer learning technique which adapts a Joint Bayesian model learned on a dataset containing 99773 images from 2995 different subjects [3].

2.2 Big Data and Deep Learning

In recent years, plenty of photos have been gathered by search engines and uploaded to social networks, which include a variety of under-constrained material, such as objects, faces and scenes. Due to this large volume of data and the increase in computational resources, more powerful statistical methods can be implemented.

This has led to the result that the robustness of vision system to several important variations, such as non-rigid deformations, clutter, occlusion and illumination, has improved [3]. These are every-day problems of many computer vision applications. Whereas conventional learning methods as Support

Vector Machines, Principal Component Analysis and Linear Discriminant Analysis, have limited capacity with large volumes of data, deep neural networks have shown better scaling properties [3].

Recently, deep and large neural networks have been the focus of attention due to their impressive results. They have been applied to large amounts of training data and scalable computation resources such as thousands of CPU cores and GPU's have become available [3]. It was shown by Krizhevsky et al. that very large deep convolutional networks trained by standard backpropagation (see section 3.5.2) can achieve excellent accuracy when trained on a large dataset [7] .

2.3 DeepFace

Using all this ideas as an inspiration and with the goal in mind of improving the accuracy of the already existent recognition systems, the University of Tel Aviv (Israel) in collaboration with the Facebook AI Research Menlo Park in California (USA) have developed *DeepFace*, a system that has closed the gap to human-level performance [3]. It is based on the conventional four stages of modern face recognition: detect, align, represent and classify.

They present a system that has closed the majority of the remaining gap in the most popular benchmark in unconstrained face recognition and is now at the brink of human level accuracy. The error rates and accuracy of the system is shown in Table 1.

Network	**Error (SFC)**	**Accuracy**
DeepFace-align2D	9,50%	94,3%
DeepFace-gradient	8,90%	95,82%
Deepface-Siamese	NA	96.17%

Table 1. The performance of various individual DeepFace netwoks and the Siamese Network [3].

It is trained on a large set of faces acquired from a population vastly different than the one used to construct the evaluation benchmarks, and it is able to surpass existing systems with only very minimal adaptation.

Furthermore, the system produces an extremely compact face representation. It differs from the majority of contributions in the field in that it uses the deep learning (DL) framework, instead of well-engineered features. DL is especially suitable for dealing with large training sets, with many recent successes in diverse domains such as vision, speech and language modelling. It has been proved , that with faces the success of the learned net in capturing facial appearance in a robust manner is highly dependent on a very rapid 3D alignment step.

The architecture is based on the assumption that once the alignment is completed, the location of each facial region is fixed at the pixel level. Therefore, it is possible to learn from the raw pixel RGB values, without any need to apply several layers of convolutions. To summarize, the following contributions are taken:

- The development of an effective deep neural network (DNN) architecture and a learning method that leverage a very large labelled dataset of faces in order to obtain a face representation that generalizes well to other databases.

- An effective facial alignment system based on explicit 3D modelling of faces.
- Advance the state of the art significantly reaching near human performance and decreasing the error rate more than 50% [3].

2.4 Face Recognition Algorithms

The Massachusetts Institute of Technology (MIT) has developed an algorithm based on principal component analysis [1]. In this technique, a set of reference faces is used to compute a set of eigenfaces, defined as the eigenvectors produced by the analysis. As a result of it, the face is represented in the image as its projection onto the eigenvectors. The goal of the algorithm is to identificate faces by comparing their projections in eigenspace, computing the eigenvectors from a subset of the images in the database [8].

The Computational Laboratory of the Rockefeller University has developed an algortith based on factorial learning and local feature analysis. Local feature analysis consists of a sparsely distributed coding of correlated local features. The result is a local low-dimensional compact representation of the face [9].

Finally, the Computational and Biological Vision Laboratory of the University of Southern California (USC) has also developed a face recognition algorithm. It is based on the use of the dinamic-link-architecture paradigm, which projects an image into a set of Garbor Jets. These are sets of Gabor wavelets with different scales and orientations, all centered at the same pixel. They are located in the vertex of a planar graph, which is a geometric model of the face. As a result, the face is represented as the coefficients derived from projecting the image onto the Garbor jets and the distances between vertices in the graph. Th similarity between faces is determined by comparing the Garbor jet coefficients and the graphs [10].

It is important to consider also the AdaBoost classifier [11], used with Haar and Local Binary Patterns (LPB). These Haar-like features are evaluated using a new image representation that generates a large set of features; afterwards, the AbaBoost algorithm is used in order to reduce the degenerative tree of boosted classifiers. Besides, only rectangular Haar-like features are used providing a speed increase [12].Using the Local Binary operator, based on the threshold operation, each face can be considered as a composition of micro-patterns which can be decetected using that operator [13].

Support Vector Machine (SVM) is used with Histogram Oriented Gradients (HOG) features for face detection. These gradients should be calculated at the finest available scale in the current pyramid layer; besides, strong local normalization is essential for good results. Th idea is to perform the training step in a difference space that explicitily captures the dissimilarity between two facial images [14].The results of testing using different datasets are shown on the next table

Dataset	**Detection**		
	AdaBoost		**SVM**
	Haar	**LBP**	**HOG**
1	99.31%	95.22%	92.68%
2	98.33%	98.96%	94.10%
3	98.31%	69.83%	87.89%

Table 2. Results of AbaBoost and SVM [1].

3. Model and Concepts

In this section, the concepts used to implement the recognition algorithm are going to be presented in a theoretical way. First of all, Image processing techniques are used to avoid as much as possible the influence of variable lighting conditions. Secondly, several techniques of classification will be used in order to model the colour of skin, eyes and hair, as well as an algorithm for face recognition based on Local Binary Patterns Histograms. Also, several algorithms will be explained in order to detect the iris, the hair and drop skin from face.

3.1 Image Processing

Before starting to find important features in an image, it is important to apply an appropriate filter so that the lighting conditions affect the accuracy of the system as little as possible. To this end, noise will be eliminated using a Gaussian Filter.

3.1.1 Gaussian Blur

The Gaussian Blur, also called Gaussian Smoothing, is a type of image-blurring filter that calculates the transformation to apply to each pixel in the image using a Gaussian function, whose equation is shown in Eq.1,

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\frac{-x^2}{2\sigma^2}} \qquad \textbf{(1)}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis and σ^2 is the variance of the Gaussian filter; it is assumed that the distribution has a mean of zero, as shown in Figure 1 [15].

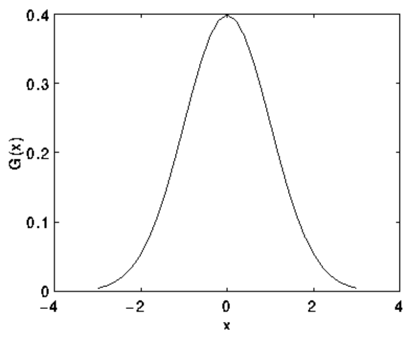


Figure 1. 1-D Gaussian distribution witth mean zero and σ^2=1 [15].

The isotropic Gaussian function in two dimensions is shown in Equation 2 [16] and has the form shown in Figure 2 [17].

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \textbf{(2)}$$

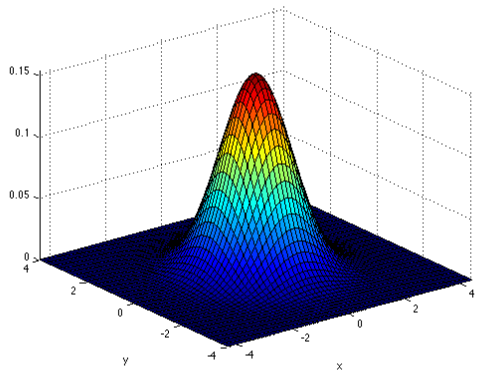


Figure 2. 2-D Gaussian distribution with mean zero.

Gaussian smoothing uses the two-dimensional distribution as a 'point-spread' distribution, achieved by means of convolution. Since the image is stored as a collection of discrete pixels, a discrete approximation of the Gaussian function is needed before performing the convolution.

Ideally, a Gaussian distribution is non-zero at every point, what would require an infinitely large convolution kernel, but in practice, it is considered effectively zero at a distance larger than about three standard deviations (3σ) from the mean. Therefore, this is the point where the kernel can be truncated.

On the other hand, in order to have a suitable convolution kernel, it is appropriate to integrate the value of the Gaussian function over the whole pixel matrix, which is done by summing the Gaussian every 0.001 increments. As the integrals are not integers, re-escalation is needed in order to have a value of 1 at the corners of the kernel. In the end, 273 is the addition of all of the kernel's values.

$$\frac{1}{273}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

Figure 3. Discrete approximation of Gaussian function with $\sigma^2=1$ [15].

Once a suitable kernel has been calculated, the Gaussian smoothing is performed using standard convolution methods. As the two-dimensional isotropic Gaussian is separable into x and y components, the convolution can be performed by convolving first a one-dimensional Gaussian in the x direction and then convolving with another Gaussian in the y direction .

Gaussian filtering results in a blurred image that preserves boundaries and edges, but eliminates noise; therefore it is the most appropriate filter for the present purpose, as the desired output is the input image with no noise respecting possible useful edges [15].

3.1.2 Laplacian of Gaussian (LoG)

The Laplacian of Gaussian is a two-dimensional isotropic measure of the second spatial derivative of an image. It highlights regions of rapid intensity changes and is used for edge detection. The Laplacian is applied to an image that has first been smoothed with a Gaussian Smoothing Filter to reduce its sensitivity to noise (See section 3.1.1).

The Laplacian of an image with pixel intensity values I(x,y) can be calculated using a convolution filter and is given by:

$$L(x,y)=\frac{\partial^2 I}{\partial x^2}+\frac{\partial^2 I}{\partial y^2} \tag{3}$$

The input image is represented as a set of discrete pixels; therefore, it is necessary to find a convolution kernel that can approximate the second derivatives given in the definition of the Laplacian.

$$\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

Figure 4. Two commonly used discrete approximations to the Laplacian filter [18].

Using these kernels, the Laplacian can be calculated using standard convolution methods [18]. Due to the associative property of the convolution, it is possible to first convolve the Gaussian smoothing filter with the Laplacian filter and then convolve this hybrid filter with the image to achieve the required results [19].

The two-dimensional Laplacian of Gaussian function centred on zero and with Gaussian standard deviation (σ) is given by the following equation .

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \textbf{(4)}$$

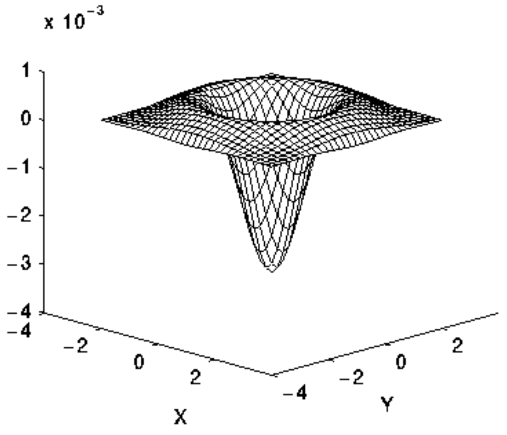


Figure 5. The two-dimensional Laplacian of Gaussian (LoG) function. (X and Y axes are marked in standard deviation units) [18].

An example of a discrete kernel that approximates this function (for Gaussian σ =1.4) is shown in Figure 5 [18].

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 6. Discrete approximation of LoG function with σ =1.4.

3.2 Edge Detection

Edge detection is the process of locating an edge in an image, which is a very important step in order to understand image features. Edges contain meaningful features and give significant information. Edge detection reduces the image size and filters out information that may be considered as less important, this way maintaining the important structural properties of an image.

Images contain some amount of redundant information that can be removed when edges are detected. Since edges often appear at image locations representing object boundaries, edge detection is used in image segmentation when images are divided into areas corresponding to different objects.

3.2.1 Canny Edge Detection

Requirements

- Good detection – The probability of failing to mark real edge points as well as the probability of falsely marking non-edge points should be low. As these probabilities are monotonically decreasing functions of the output signal-to-noise ratio, this leads to maximise the signal-to-noise ratio.
- Good localization - The points analyzed as edge points should be as close as possible to the centre of the true edge. In consequence, the marked out edges should be as close to the real edges as possible.
- Minimal response – Only one response to a certain edge. Hence, it follows that an edge should be marked only once and image noise should not create false edges.

Algorithm

This method was first designed by John F. Canny, who used the calculus of variations, a technique which finds the function that optimises a given function. The optimal function in Canny's detector is described using the sum of four exponential terms, although it can be approximated by the first derivative of a Gaussian.

Considering the signal-to-noise ratio and localization, we assume the impulse response to the filter to be f (x) and the edge to be G(x). It is considered that the edge is centered at x=0, thus the response of the filter to this edge at its center H_g is given by the convolution integral

$$H_G = \int_{-W}^{+W} G(-x)f(x)dx \tag{5}$$

assuming that the filter has a finite impulse defined by [-W, +W]. The root-mean-squared response to the noise n(x) only , will be

$$H_n = n_0 \left[\int_{-W}^{+W} f^2(x)dx\right]^{\frac{1}{2}} \tag{6}$$

where n_0^2 is the mean-squared noise amplitude per unit length. The Signal to Noise Ratio (SNR) is defined as:

$$SNR = \frac{\left| \int_{-W}^{+W} G(-x) f(x) dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f^2(x) dx}} \quad \textbf{(7)}$$

Attending to the location, the reciprocal of the root-mean-squared distance of the marked edge from the centre of the edge is used. Considering that the edges are marked at the local maxima of the function $f(x)$, the first derivative of the response at these points will be zero. Besides, as the edges are centred at x=0 there should be a local maximum in the response at x=0 in the absence of noise.

Let $H_n\,(x)$ be the response to the noise and $H_G\,(x)$ be the response to the edge. Considering that there is a local maximum in the total response at $x = x_0$, then:

$$H'_n\,(x_0) + H'_G\,(x_0) = 0 \quad \textbf{(8)}$$

The Taylor expansion of $H'_G\,(x_0)$ about the origins gives

$$H'_G(x_0) = H'_G(0) + H''_G(0) x_0 + O\,(x_0^2) \quad \textbf{(9)}$$

Assuming that the response of the filter in the absence of noise ($H'_G(0)$) has a local maximum at the origin, the first term in the expansion may be ignored. The displacement x_0 of the actual maximum is assumed to be small so that quadratic and higher terms are not considered.

If the edge $G(x)$ is either symmetric or asymmetric, all terms in x_0 disappear. Suppose $G(x)$ is asymmetric and express $f(x)$ as a sum of a symmetric component and an asymmetric component. On one hand the convolution of the symmetric component with $G(x)$ contributes nothing to the numerator of the SNR but, on the other hand, it contributes to the noise component in the denominator.

Thus, if $f(x)$ has any symmetric component, its SNR will be worse than a purely asymmetric filter. A dual argument holds for symmetric edges, so that if the edge $G(x)$ is symmetric or antisymmetric, the filter $f(x)$ will follow suit. The final result of this, is that the response $H_G\,(x)$ is always symmetric and that its derivatives of odd orders are zero at the origin.

From Equations (8) and (9), it follows that:

$$H_G(0)'' x_0 \approx -H'_n(x_0) \quad \textbf{(10)}$$

Now, $H'_n\ (x_0)$ is a Gaussian random quantity whose variance is the mean-squared value of $H'_n\ (x_0)$, and is given by

$$E[H'_n\ (x_0)^2] = n_0^2\left[\int_{-W}^{+W} f'^2(x)dx\right] \tag{11}$$

Where $E[y]$ is the expectation value of y. After combining this result with (10) and substituting for $H''_G(0)$ gives

$$E[x_0^2]\ \approx\ \frac{n_0^2\left[\int_{-W}^{+W} f'^2(x)dx\right]}{\left[\int_{-W}^{+W} G'(-x)f'(x)dx\right]} = \delta x_0^2 \tag{12}$$

Where δx_0 is an approximation to the standard deviation x_0. The localization is defined as the reciprocal of δx_0.

$$Localization = \frac{\left|\int_{-W}^{+W} G'(x)f'(x)dx\right|}{n_0\sqrt{\int_{-W}^{+W} f'^2(x)dx}} \tag{13}$$

Equations (7) and (13) are the mathematical forms for the first two criteria, thus the design problem leads to the maximization of both of these. For this goal, the product of (7) and (13) is maximized, which simplifies the analysis for step edges. Firstly, the product of the criteria for arbitrary edges will be maximized.

$$\frac{\left|\int_{-W}^{+W} G(-x)f(x)dx\right|\left|\int_{-W}^{+W} G(-x)f'(x)dx\right|}{n_0\sqrt{\int_{-W}^{+W} f'^2(x)dx}\ n_0\sqrt{\int_{-W}^{+W} f^2(x)dx}} \tag{14}$$

At this point, the multiple responses will be eliminated. As stated before, the edges will be marked at the local maxima in the response of a linear filter applied to the image. The detection criterion measures the effectiveness of the filter for discrimination between signal and noise at the centre of an edge, but this does not take into account the behaviour of the filter nearby the edge centre.

The first two criteria can be maximized using the Schwarz inequality [20], thus SNR is bounded above by

$$n_0^{-1}\sqrt{\int_{-W}^{+W} G^2(x)dx} \tag{15}$$

and Localization by

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \textbf{(16)}$$

Both bounds are achieved and the product of SNR and location is maximized when $f(x) = G(-x)$ in $[-W,+W]$. These maxima are so close together that is not possible to distinguish the response to the step from noise. It is needed to add to the criteria the requirement that the function f will not have many responses to a single step edge in the vicinity of the step. The number of peaks in the response needs to be limited so that there is a low probability of declaring more than one edge. Ideally, it is wanted to make the distance between peaks in the noise response approximate the width of the response of the operator width ; the average distance between zero-crossings of a function g to Gaussian noise is

$$x_{ave} = \pi \left(\frac{-R\ (0)}{R\ ''(0)} \right)^{\frac{1}{2}} \quad \textbf{(17)}$$

Where $R(\tau)$ is the correlation function of g . In this case, the mean zero-crossing spacing for the function f' is been looked for.

Assuming that

$$R(0) = \int_{-\infty}^{+\infty} g^2(x)dx \quad \textbf{(18)}$$

And that

$$R''(0) = \int_{-\infty}^{+\infty} g'^2(x)dx \quad \textbf{(19)}$$

The mean distance between zero-crossings of f' will be

$$x_{zc}(f) = \pi \left(\frac{\int_{-\infty}^{+\infty} f'^2(x)dx}{\int_{-\infty}^{+\infty} f''^2(x)dx} \right)^{\frac{1}{2}} \quad \textbf{(20)}$$

The distance between adjacent maxima in the noise response of f, denoted by x_{max} , will be twice x_{zc}. This distance is set to be some fraction k of the operator width.

$$x_{max} = 2x_{zc}(f) = kW \quad \textbf{(21)}$$

This is a natural form of constraint as the response of the filter will be concentrated in a region of width $2W$, and the expected number of noise maxima in the region N_n where

$$N_n = \frac{2W}{x_{max}} = \frac{2}{k} \tag{22}$$

By fixing k, the number of noise maxima that could lead to a false response is also fixed. The inter-maximum spacing (17) scales with the operator width. Thus, we first define an operator f_w which is the result of stretching f by a factor of w, $f_w = f\left(\frac{x}{w}\right)$. Then after substituting into (17) it is found that the inter-maximum spacing for f_w is $x_{zc}(f_w) = wx_{zc}(f)$.

If some function f satisfies the maximum response constraint (18) for fixed k , then the function f_w will also satisfy it, assuming W scales with w . For any fixed k, the multiple responses do not depend on the spatial scaling of f.

The Canny Edge Detection Algorithm is adaptable to various environments and its parameters allow to use it for recognition of edges of differing characteristics depending on the particular requirements of a given implementation. [20]

3.3 Haar Cascade Classifier

This kind of classifiers has been developed based on the algorithm that Viola and Jones introduced to detect objects rapidly and accurately within an image. Nevertheless, the area of the image analysed to detect representative features needs to be regionalized to the location showing the highest probability of containing them. This regionalization decreases the number of false positives and increases the speed because of the reduction of the examined area [21].

3.3.1 Haar-Like Features

Haar classifier object detection is based on the Haar-like features. These features use the change in contrast values between adjacent rectangular groups of pixels instead of the intensity values of a pixel. Using the contrast variances between those groups is possible to find relative light and dark areas. These features can be scaled by increasing or decreasing the size of the pixel group, which allows features to be used to detect objects of various sizes [22].

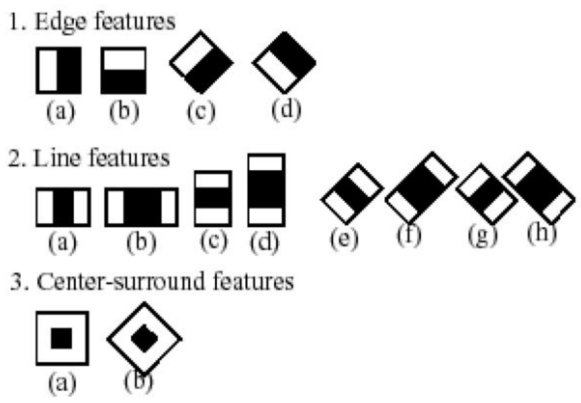


Figure 7. Common Haar Features [23].

The rectangular features of an image are determined using an intermediate representation of it, called integral image [21] . The integral image is an array that contains the sums of pixels' intensity values located directly to the left and above the pixel at location (x,y) inclusive.

If A[x,y] is the original image and AI [x,y] is the integral image, the integral image is estimated as:

$$AI[x, y] = \sum_{x' \le x, y' \le y} A(x', y') \tag{23}$$

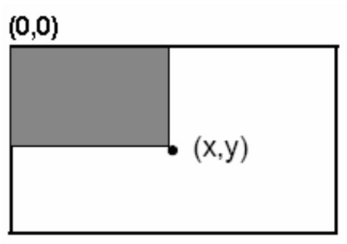


Figure 8. Summed area of integral image [23].

The features rotated by forty-five degrees need another intermediate representation called the rotated integral image or rotated sum auxiliary image. In this case, the rotated integral image is calculated using the sum of the pixels' intensity values that are located at a forty-five degrees angle to the left and above for the x value and below for the y value.

If A[x,y] is the original image and AR [x,y] is the rotated integral image then the integral image is estimated as:

$$AI[x, y] = \sum_{x' \le x, y' \le x - |y - y'|} A(x', y') \tag{24}$$

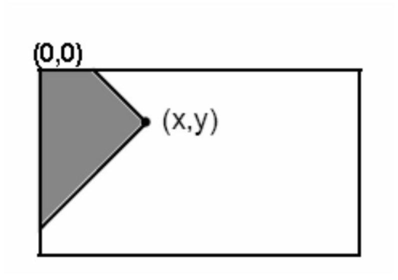


Figure 9. Summed area of rotated integral image [23].

Both integral image arrays can be computed in two steps, one for each integral array. Using the pertinent integral image and taking the difference between six to eight array elements forming two or three connected rectangles, it is possible to estimate a feature of any scale. Hence, it follows, that calculating a feature is very fast and efficient. As a consequence, calculating features of various sizes requires the same effort as a feature of only a few pixels. The effort needed to find various sizes of the same object is similar to the work needed to detect objects of similar sizes [23].

3.3.2 Classifiers Cascaded

A cascade classifier is a particular case of ensemble learning based on the concatenation of several classifiers using all information collected from the output of a given classifier as additional information for the next classifier in the cascade. Cascade classifiers are considered as multistage systems [24].

Calculating a feature is extremely efficient and fast, but a 24 x 24 sub-image contains 180,000 features. Therefore, calculating all of them is totally impractical. Nevertheless, only a tiny fraction of those features are needed to determine if a sub-image potentially contains the desired object. In order to eliminate as many features as possible, only a few of the features that define an object are used when analysing sub-images. Thus, in order to decrease running time as well as increase accuracy,around 50 % of the sub-images that do not contain any of the obect's features will be eliminates. This process is iterated using a high number of features to analyse the sub-image each time.

The cascading of the classifiers allows only the sub-images with the highest probability to be analyzed for all Haar-features that distinguishes an object. Besides, the accuracy of the classifier can also be changed. Viola and Jones were able to achieve a 95% accuracy [23].

3.4 Naïve Bayesian Classifier

A classifier is defined as a function that relates input feature vectors $x \in X$ to output class labels $y \in \{1, \ldots, C\}$ where X is the feature space. It is assumed that the feature space is represented as a vector of real numbers or binary bits, but in general, it is possible to combine discrete and continuous features. It is also assumed that the class labels are unordered and mutually exclusive. As the goal is to learn the classifier from a labelled set of N input-output pairs, $(x_n, y_n), n = 1{:}N$, it is considered a supervised learning process.

The Naive Bayesian Classifier is a probabilistic classifier, this is a method that returns $p(y|x)$. The idea is to learn the class-conditional density $p(y|x)$ for each value of y, and learn class probabilities $p(y)$. Moreover, we can compute the posterior using the Bayes Rule [25].

Given a class variable y and a dependent feature vector x_1 through x_n Bayes' Rule gives the following relationship [26]

$$P(y, x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n|y)}{P(x_1, \ldots, x_n)} \tag{25}$$

This is considered as a generative model, as it provides a method to generate the feature vectors x for each possible class y [25]. Using the Naïve's independence assumption, that states that every feature x_i is conditionally independent of every other feature x_j for $j \neq i$, given the category y. This means that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y) \tag{26}$$

For all i the relationship is simplified to

$$P(y|x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \ldots, x_n)} \tag{27}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, it can be used the following classification rule

$$P(y|x_1, \ldots, x_n) \propto P(y)\ \prod_{i=1}^{n} P(x_i|y)$$

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y) \tag{28}$$

This function is called discriminant function and directly maps inputs to outputs [26].

3.4.1 Gaussian Naïve Bayes

If used data are continuous, it is common to assume that the values associated with each class are distributed according to a Gaussian distribution. Supposing that the training data contain a continuous attribute x ; we first segment the data by the class, and then compute the mean and variance of x in each class. The probability distribution of v given a class y , $p(x = v|y)$, can be computed using the equation for a Normal Distribution.

$$p(x = v|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(v-\mu_y)^2}{2\sigma_y^2}} \tag{29}$$

where μ_y is the mean of the values in x associated with class y and σ_y^2 the variance of the values in x associated with class y [27].

3.5 Neural Networks

Artificial Neural Networks (ANNs) in machine learning and cognitive science are models inspired by biological neural networks used to estimate functions that can depend on a large number of inputs and are generally unknown [28].

Neural networks have recently emerged as an important tool for classification, as the multiple research activities have shown that neural networks are a promising alternative to various conventional classification methods [29].

A reason is that neural networks are data driven self-adaptive methods that can adjust themselves to the data without any explicit specification of functional or distributional form for the underlying model. On the other hand, they are universal functional estimators since they are able to estimate any function with arbitrary accuracy. Given that any classification procedure looks for a functional relationship between the group membership and the attributes of the object, the accurate identification of the underlying function is essential.

Besides, the non-linear condition of neural networks makes them flexible in modelling real world complex relationships. Eventually, neural networks are able to estimate the posterior probabilities, the basis for establishing the classification rule [29].

3.5.1 Multilayer Perceptrons (MLPs)

A Multilayer Perceptron is defined as a network of simple neurons called perceptrons, in which the computation is performed using a set of many simple units with weighted connections between them. The concept of a single perceptron was introduced by Rosenblatt in 1958 [30]: the perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights. Afterwards the output passes through some non-linear activation function.

The mathematical expression of the performance can be written as

$$y = \varphi\left(\sum_{i=1}^{n} \omega_i x_i + b\right) = \varphi(w^T x + b) \tag{30}$$

where w defines the vector of weights, x is the vector of inputs, b is the bias and φ is the activation function. Multilayer perceptrons allow a neural network to perform arbitrary mappings.

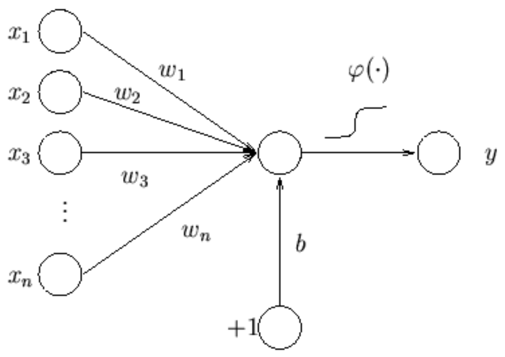


Figure 10. Signal-flow graph of a perceptron **[31]**.

A single layer perceptron is not very useful as its ability for mapping is very limited. Regardless of the activation function, it is only able to represent an oriented ridge-like function. Nevertheless, the perceptrons can be used to build a larger and much more practical structure. A multilayer perceptron network is formed by a set of source nodes forming the input layer, one or more hidden layers of computation junctions, and an output layer of nodes. The propagation of the input signal is done layer by layer.

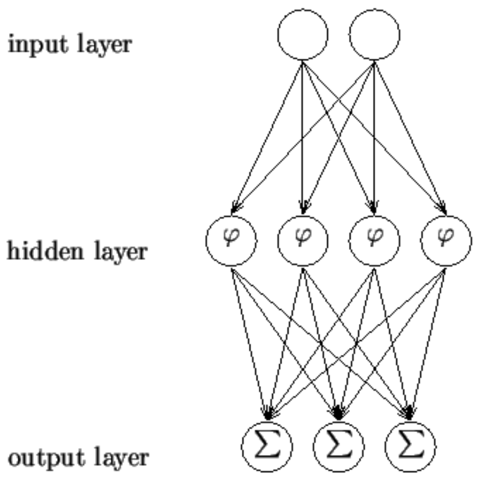


Figure 11. Signal-flow graph of a MLP **[31]**.

The input layer accepts the data vector or pattern, afterwards the hidden layer(s) accept the output of the previous ones, weight them and pass through a non-linear activation function. Finally, the output layer takes the output from the final hidden layer, weighs it and pass through an output non-linear to produce the target values [31].

Possible activation functions

The original Rosenblatt's perceptron used a Heaviside step function [30]. However, nowadays the activation function is often chosen to be either a sigmoid or the hyperbolic tangent. These two functions are related by:

$$\frac{\tanh(x)+1}{2}=\frac{1}{1+e^{-x}} \tag{31}$$

They are used because they are mathematically convenient as well as close to linear near the origin,because they saturate quickly when getting away from the origin. These properties allow multilayer perceptron networks to model strongly and face nonlinear mappings [31].

3.5.2 Supervised Learning of Neural Networks: backpropagation.

Neural networks use some kind of learning rule that determines the connections weights in order to minimize the error between the neural network output and the desired output [32]. The Backpropagation algorithm, an abbreviation for "backward propagation of errors", is a method of training artificial neural networks used in cooperation with an optimization method such as gradient descent [33]. The backpropagation algorithm looks for the minimum of the error function in weights using the gradient descent. This method has two main steps, the forward pass and the backward pass.

Firstly, during the forward pass the predicted outputs corresponding to the given inputs are evaluated using the following equation

$$x=f(s)=B\varphi(As+a)+b \tag{32}$$

where s is a vector of inputs and x a vector of outputs. A is the matrix of weights of the first layer and a is the bias vector of the first layer. B and b are the weight matrix and the bias vector of the second layer, and φ denotes an elementwise nonlinearity.

Secondly, in the backward path, partial derivatives of the cost function taking into account the different parameters are guided through the network. The idea is to adapt afterwards the network weights with any gradient-based algorithm. In the end, the whole process is iterated until the weights converge [31] [32].

3.6 Hough Circle Transform

Hough transform is a method used for finding simple forms in an image, such as circles or lines. The original Hough transform was a line transform, which allows detecting straight lines in a binary image fast [34]. Nowadays, this method has been generalised to other cases than just simple lines.

3.6.1 Hough line transform

The very first principle of a Hough transform is that any point in an image can be part of some set of possible lines. If a line is parametrized using a slope a and an intersection b , then a point in the original image is transformed to a locus of points in the *(a,b)* plane consisting of all the lines passing through that point. If every non-zero pixel in the input image is converted into a set of points in the output image and all contributions are added; then they will appear as local maxima in the output image. As all contributions of each point are being summed, the plane *(a,b)* is commonly called the accumulator plane.

Sometimes, the slope-intersection form is not really the best way to represent all lines passing through a point because of the considerably different density of lines as a function of the slope, as well as the fact that the range of possible slopes goes from -∞ to +∞. This is the reason why.the preferred parametrization represents each line as a point in polar coordinates (ρ,θ), with the represented line being the line passing through the indicated point but perpendicular to the radial from the origin to that point . The equation representing a line is [34]:

$$\rho = x\, cos\theta + y\, sin\theta \tag{33}$$

3.6.2 Hough Circle transform

The Hough circle transform works in a similar way to the Hough line transform. The main difference is that, in this case, the accumulator plane has to be replaced with an accumulator volume with three dimensions: one for x, one for y, and another for the circle radius r. This leads to a far greater memory and slower speed.

Nevertheless, this problem can be eliminated using the Hough Gradient Method. First,an edge detection algorithm is applied to the image. Next, the local gradient of every non-zero point in the image is computed. Using this gradient, every point along the line is added to the accumulator. At the same time, we save the location of every non-zero pixel. The candidate centers are then selected from those points in the accumulator that are above a given threshold as well as larger than all of their immediate neighbours. These candidate centers are ordered descending according to their accumulator values so that the centers with the most supporting pixels appear first.

Next, for each center, all of the non-zero pixels are ordered according to their distance from the center. Working out from the smallest distance to the maximum radius, the best-supported radius by the non-zero pixels is selected. A center is only considered if it has sufficient support from the nonzero pixels on the edge and if there is a given distance from any previously selected center.

Besides, this method of implementing the algorithm allows it to run much faster and helps to solve the problem of the otherwise sparse population of a three-dimensional accumulator, which would bring noise and instability.

Nonetheless, this algorithm has also some disadvantages. First of all, the entire set of nonzero pixels in the edge image is considered for every candidate center; this means that if the accumulator threshold is chosen too small, the runtime of the algorithm will be too large.

Secondly, as only one circle is selected for every center, if there are concentric circles then only one of them will be detected. Finally, as all the centers are considered in ascending order according to the accumulator value and as long as the new centers are not kept if they are too close to previously accepted centers, there is a bias toward keeping the larger circles when multiple circles are concentric or approximately concentric [34].

3.7 Local Binary Patterns Histogram (LHBP)

The main purpose of the Local Binary Pattern (LBP) operator was originally the description of textures. This operator assigns a label to every pixel of an image by thresholding the 3 x 3 neighbourhood of each pixel with the center pixel value; in the end, it returns a binary number. Then, the histogram of each label is extracted and used to analyse the texture. The following figure shows the basic LBP operator.

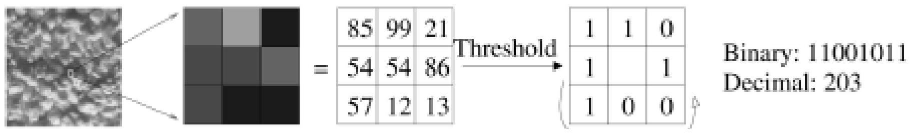


Figure 12. The basic LBP operator.

However, the operator was not that useful, as it only allowed to work with neighbourhoods of 3x3 pixels. In order to make the operator more practical, it was thought to use it for defining textures of neighbpurhoods of different sizes; therefore it was later extended to use neighbourhoods of different sizes. If the local neighbourhood is defined as a set of sampling points spaced on a circle centered at the pixel to be labelled, any radius and number of sampling points can be used. If a sampling point does not coincide with the center of a pixel, bilinear interpolation is used.

In addition, a local binary pattern is called uniform if the binary pattern contains at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is considered circular. When computing the LBP histogram, uniform patterns are used so that the histogram has a separate bin for every uniform pattern and all non-uniform patterns are assigned to a single bin [35].

3.7.1 Local Binary Patterns for Face Description

The formal description of the basic LBP operator is the following

$$LBP\,(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c) \tag{34}$$

where (x_c, y_c) is a central pixel with intensity i_c and i_n is the intensity of the neighbour pixel. S is the sign function defined as:

$$(x) = \begin{cases} 1, & x \geq 0 \\ 0, & else \end{cases} \tag{35}$$

By aligning an arbitrary number of neighbours on a circle of variable radius, the following neighbours can be obtained.

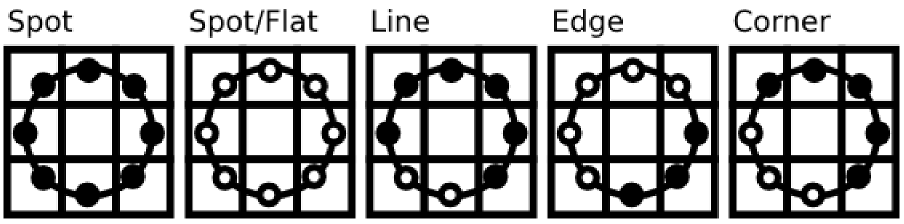


Figure 13. Possible Neighbours using the extended LBP operator [35].

Let a sample point be (x_c, y_c) then the position of the neighbour (x_p, y_p) $p \in P$ can be calculated as follows

$$\begin{aligned} x_p &= x_c + R\cos\left(\frac{2\pi p}{p}\right) \\ y_p &= y_c - R\sin\left(\frac{2\pi p}{p}\right) \end{aligned} \tag{36}$$

where R is the radius of the circle and P is the number of sample points. If a point's coordinate on the circle doesn't correspond to image coordinates, the point is interpolated using a bilinear interpolation.

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \tag{37}$$

The LBP operator is robust against monotonic grey scale transformations [36]. The method builds a global description of the face by computing several local descriptors achieved by means of the local binary opereator. Therefore, the facial image is divided into local regions, in which several independent texture descriptors are used. They are then concatenated into a global description of the face. Afterwards, the histogram of each region is extracted.

The spatially enhanced feature vector is then obtained by concatenating the local histograms. These histograms are called Local Binary Patterns Histograms [35].

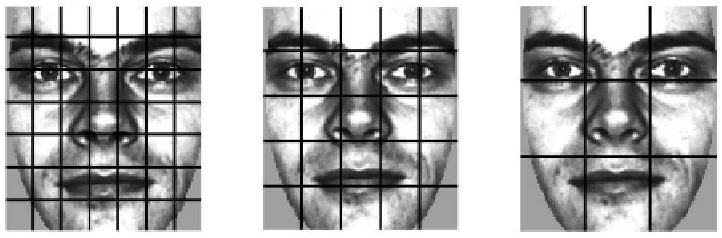


Figure 14. Face divided into 7 x7, 5x5 and 3x3 rectangular regions [35].

4. Implementation

In this section, the implementation of the system will be explained using the concepts explained in the previous section. In order to implement the code, the IDE Visual Studio Community 2015 has been used, in cooperation with the Intel Open Computer Vision Library (OpenCV) using the C++ programming language.

An overview of the implemented system is shown in the following figure.

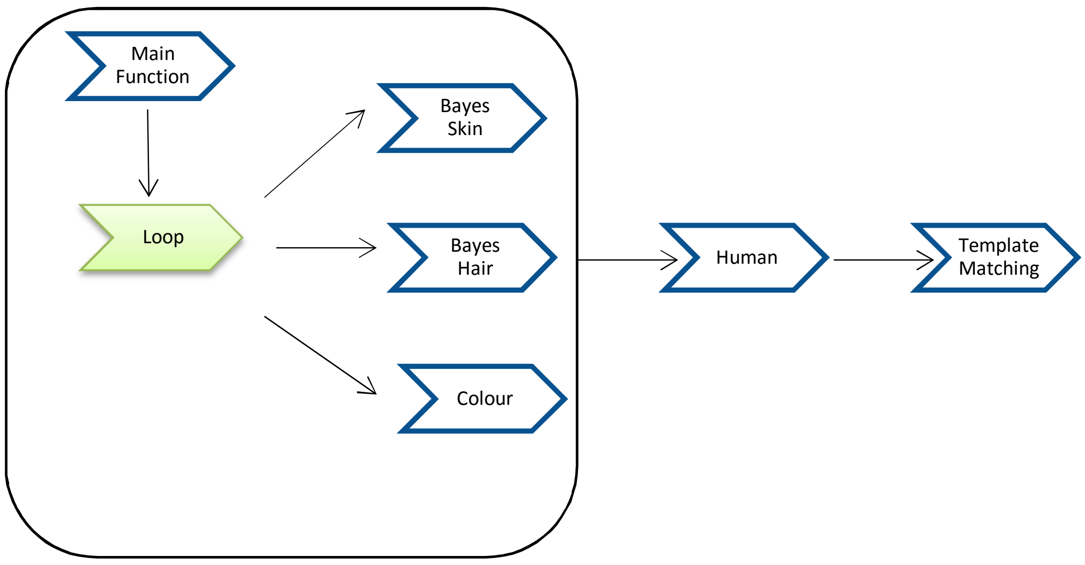


Figure 15. Overview of the implemented system.

The system consists of two classes and four scripts, where the used methods are codified. The main function contains a loop, which allows the program to run continuously. Thus, all the the methods programmed in the other scripts are used here; this is, the two Bayesians classifiers, the neural networks and the methods created in order to extract features from face.

On one hand, the class colour, is used to create objects that allow to save the properties of the analysed colours. On the other hand, the class human provides a way to save characteristics of the analysed persons.Finally, the template matching algorithm use all the extracted information to compare with the existing database in order to perform classification.

4.1 Cascade Classifiers for face and eye detection

Detecting facial features, such as the mouth, eyes, and nose require that Haar cascade classifiers are first trained. In order to train the classifiers the AbaBoost algorithm, as well as, the Haar Feature algorithm must be implemented. Nevertheless, OpenCV already contains a function that implements the Haar Feature algorithm (see Annex A) [37].

To train the classifiers, two set of images are needed. One set contains an image or scene that does not contain the object, in this case, a facial feature, which is going to be detected; this set of images is called negative images. The other set of images, namely positive images, contain one or more instances of the object. The location of the objects within the positive images is specified by image name, the upper left pixel and the height and width of the object. For training facial features, 5.000 negative images with at least one megapixel resolution are used containing everyday objects, like paperclips, and natural scenery, like photographs of forest and mountains.

In order to produce the most robust facial feature detection possible, the original positive set of images needs to be representative of the variance between different people including race, gender and age [23].

The way to use these algorithms in OpenCV is the following. The first step is loading the classifiers form an XML file; there is one file for the eyes called “haarcascade_eye_tree_eyeglasses.xml” and two files for the face; the first one for the frontal face is called “lbpcascade_frontalface.xml” (as local binary pattern are used in the classification) and the second one for the profile face is called “haarcascade_profileface.xml) [34].

The second step is using the following function:

> ***“void detectMultiScale (constMat& image, vector <Rect>& objects, double scaleFactor =1.1, int minNeighbours = 3, int flags =0, Size minSize =Size(),Size maxSize = Size())”***

It is important that the input image is in grayscale; therefore, it is needed to convert it to that colour space before calling the function. Besides, the parameters *minSize* and *maxSize* , define the minimum and maximum possible size of the detected objects. These parameters are set as *Size (30,30)* for both the face and the eyes; as practice showed it is working efficiently with this definition.

The described function returns a vector of rectangles containing the detected object, that are used to create a region of interest (ROI) to crop the objects from the image. These rectangles are defined using two opposite vertexes [38].

Finally, after calling the function and crop both ROIs from the image, one of the eyes and the other of the face, we save them in RGB colour space, so that we can use them in further steps, for the eye colour modelling and the skin colour modelling.

4.2 Face Recognition using Local Binary Patterns Histograms (LBPH)

In this section, the way to perform the face recognition will be explained. Local Binary Patterns Histograms are used (see section 3.7), which are already implemented in OpenCV. The idea is to have a database of faces, a given number of people, from three to five pictures of everyone in different conditions: with and without glasses, with tied-back hair and without it, smiling and serious, front pictures and tilted ones [i].This database will be used to train the classifier, using the function already provided by OpenCV. The images are extracted from the Glasgow Unfamiliar Face Database (GUFD) database of faces and have a size of 350 x 500 pixels [39].

For this goal, the paths of the used images are saved in a CSV file, and every subject is assigned an integer number, called label, which identifies him or her. In order to train the classifier, a vector of images, extracted from the CSV file, and a vector of labels, also present in the mentioned file, are needed.

Next step is using the classifier, for this purpose OpenCV has also an implemented function, that requires as an argument the image of the face we are about to classify. This method returns an integer that corresponds to the predicted label, this is, the face of the database that better fits the sample image [36].

This is the first step of the classification, as in the following sections the colours of the eyes, skin and hair will be analysed in order to get better results (see Annex A for code).

4.3 Eye Colour Recognition

The algorithm for the modelling of the eye colour has two phases. First of all, a Canny edge detection as well as a circular hough transform are applied in order to detect the iris and crop it from the image. Secondly, a classifier based on neural networks, to categorise the colour of the detected iris in one of the following groups: brown, blue or green, is implemented.

4.3.1 Phase 1: Canny Edge Detection and Circular Hough Transform

Image Acquisition

Image acquisition is considered the most critical step, as all the subsequent stages depend highly on the image quality [40]. The GUFD Database was used, which contain 3D images of subjects with different characteristics [39]. The pictures have a resolution of 350 x 500 pixels and the format is jpg.

Image Manipulation

The image of the eye obtained using the Haar Cascade Classifier is first converted to grey scale in order to facilitate the manipulation of the images during further steps [40]. Afterwards, a Gaussian Blur filter is applied in order to reduce noise. This filter also helps to make the detection as accurate as possible.

Iris Localization

Firstly, we use the Canny function to detect edges in the image. This function is defined in OpenCV as:

> ***"void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int aperture=3, bool L2gradient = false)"***

It is set a low threshold (*threshold1*) of 0 and an upper one (*threshold2*) of 10. The parameter *aperture* defines the aperture size for the *Sobel()* operator, while *L2gradient* is a flag that indicates whether a more accurate $L_2\ norm = \sqrt{\left({dI}/{dx}\right)^2 + \left({dx}/{dy}\right)^2}$ should be used.

Then, the function HoughCircles is called to detect the circles included in the eye. In OpenCV, the function has the following form:

> ***"void HoughCircles (InputArray image, OutputArray circles, int method, double dp, double minDist, double param1=100, double param2 = 100, int minRadius=0, int maxRadius =0)"***

The first parameter is the image in which the detection is performed, in this case the image of the eye. The parameter *method* refers to the algorithm applied to detect the circles, in this case the Hough Gradient is used (see section 3.6.2), which can be called using the keyword *CV_HOUGH_GRADIENT.*

The parameter *dp* is defined as the inverse ratio of the accumulator resolution to the image. In this case, it is defined as 1, what means that the accumulator has the same resolution as the input image.

Besides, *minDist* is the minimum distance between the centers of the detected circles, it has to be defined so that it is neither too small, as multiple neighbour circles could be falsely detected, nor too large, as some important circles could be missed.

Consequently, it is defined as 20, so that there has to be a minimum distance of 20 pixels between one circle and another. The idea is that there is only one circle detected per eye, therefore the minimum distance is set large.

Following with the explanation of the used function, *param 1* and *param2* are two method-specific parameters. The first parameter is the higher threshold of the two passed to the *Canny()* edge detector, being the lower one which is twice smaller. The second parameter is the accumulator threshold for the circles center at the detection stage. They are set as 50 and 30, respectively.

Finally, the two latest parameters are the minimum and maximum radius considered for the detected circles; they are set as 0 and 25, respectively. On top of that, this function provides us with a vector of found circles (parameter *circles*) whose elements are encoded as a 3-element floating-point vector$(x, y, radius)$ [41].

Afterwards, this output vector of circles is used to draw a circle around the eye, creating a region of interest (ROI), which is used as a mask to crop the iris from the image. Once the iris is detected,it is time to use the classifier to categorise its colour (see Annex A for code).

4.3.2 Phase 2: Neural Network for Colour Classification

In order two classify the eye's colour, a Neural Network has been implemented. Its implementation has to stages: training and testing.

Training the Neural Network

Neural Networks in OpenCV are implemented to be trained using matrixes. Two matrixes are needed, the matrix of images and the matrix of labels. First, the matrix of labels is built so that it has the same number of rows as images and the same number of columns as pixels in every image. For the current purpose, 72 images with size 30 x 30 pixels in RGB colour space are used. The idea is to put every image in a row of the images matrix, by first reshaping it to one single row.

Secondly, the matrix of labels is built to have the same number of rows as images used for training purposes and the same number of columns as categories. In this case, three categories are defined: green, blue and brown, with 24 images for each class. The idea is to put a 1 in the column that corresponds to the class of the image beeing in the same row in the matrix of images. It is important that the type of both matrices is defined as type CV_32F (matrix of floats) or CV_32S (matrix of integers). In the training algorithm used, the labels matrix is defined as CV_32S and the images matrix is defined as CV_32F.

Nevertheless, a neural network requires more parameters for training: the activation functions, the layers and the method used for the training. First, five layers have been defined, the first one with same number of neurons as the number of pixels in every image, this is 900.

The following layers have 400,100, 20 and 3 neurons, respectively. Secondly, the activation function used is the sigmoid and the training method used is the backpropagation (see section 3.5.2) [42].

The advantage of this kind of classification is that the neural network can be trained only once, as the train function returns a matrix with the weights of all neurons, which can be used later in the algorithm implemented for the prediction. This is the reason why the training is done outside the main loop, before the program starts to work in an infinite loop (see Annex A).

Testing the Neural Network

In the testing step, images containing irises of several colours are used. First, the images are resized to 30 x 30 pixels and reshaped to one single row. The goal is to include them in a matrix with as many rows as images used to test the neural network (12) and as many columns as pixels every image contains (900), whose type is set as CV_32F.

The function used to predict requires the matrix of images as input and returns a matrix that has the same number of rows as the input matrix and as many columns as classes, in this case, three. This matrix has the same form as the labels' matrix explained in the training; it has a number 1 in the column of the class that corresponds to the image placed in the same row of the images' matrix.

Finally, this output matrix is compared with the labels' matrix constructed for the test samples, extracting some conclusions that will be explained in section 5.

4.4 Skin Colour Recognition

In this section, the algorithm used to extract the colour of the skin will be presented. First, a detection of skin based on image segmentation is implemented, then the exact parameters that define the colour are extracted. The objective is to use a Naïves-Bayesian classifier to categorise the skin in two classes: light and dark.

4.4.1 Skin Detection

The image of the face obtained using the Cascade Classifier is used as input for the algorithm. First, the image is blurred to eliminate noise, since it can end in a false detection of contours. Next step is converting the image to HSV colour space.

The idea is to find the regions of the image whose colour is between the range defined for skin in the HSV colour space. This range involves the colours between (0,10,60) and (20,150,255). Once we have these regions, the idea is to construct a mask that can crop the skin regions from the image.

With this in mind, the contours contained in the imaged are detected. Nevertheless, trying to draw all these contours in the image led to a wrong detection of the regions. In order to solve this problem, the biggest contour is detected and used as a parameter in the function called to draw the contours. [43]

The function used to detect the contours is the following:

> **"void findContours(InputOutputArray image,OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())"**

where the input parameter is the image of the face after extracting the regions with a colour contained in the range established for skin. The parameter *hierarchy* is an optional output vector that contains information about the image topology. In this case the parameter *mode* is defined as CV_RETR_TREE, which retrieves all of the contours and reconstructs a full hierarchy of nested contours. Besides, the method used is CV_CHAIN_APPROX_SIMPLE that compresses horizontal, vertical and diagonal segments and leaves only their end points.

That function constructs an output vector of contours, each of which is stored as a vector of points. This vector of contours is used to find the biggest contour in the image, the one that is going to be drawn in the output image, namely mask.

The function used to draw the contours in the mask is:

> **"void drawContours(InputOutputArray image,InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArrayhierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())"**

where the input image is an empty image with the same size as the image of the face extracted using the cascade classification. The second parameter is the vector of contours extracted from the previous step. Finally, the parameter *contourIdx* is a parameter indicating the index of a contour to draw, in this case, the index of the biggest contour (see Annex B) [44].

Now that we have the mask, it can be applied to the original image to have only the regions of the image that include skin.

4.4.2 Skin Colour: Naïves-Bayesian Classifier

Once, there is an image that only contains the skin regions, it is time to extract the parameters that define colour. This is achieved by first splitting the image in the three HSV channels, and calculating the mean and standard value of the intensity of every channel. These data are saved in an object of class colour, that contains six arguments, two per channel (mean and standard deviation). These are the features used in the Bayesian classifier.

Training the Bayesian classifier

The database used to train the classifier contains 58 images of faces in 3D extracted from the GUFD database [39], 29 people with light skin and 29 people with dark skin. First of all, the skin is dropped from the images as explained in the previous section and then the colour parameters of the skin are extracted and saved in a CSV file so that they can be used more than once.

As a second step, the mean and standard deviation is calculated for every feature and each class, and saved in a CSV file that is afterwards used in the main program for the prediction. In the end there are 4 CSV files, two per class, one containing the mean of each feature and the other the standard deviation (See Annex F)

Using the Bayesian Classifier

The explained classifier is used to categorise the skin into two classes, either light or dark. However, it doesn't give an absolute result, since it gives the probability of the sample skin to belong to light and dark. In order to estimate the probability the Gaussian formula of the Naïves Bayes Theorem is used, assuming the likelihood of the extracted features to be Gaussian (See section 3.4.1).

After applying the Bayes Theorem, we will get the probability of the skin to belong to the class dark as well as the probability of the class light. The class the skin belongs to, will be the one that shows the higher probability (see Annex C).

4.5 Hair ColourRecognition

In order to model the colour of the hair, an algorithm to search for the end of the face is used, as the hair is assumed to be on the top of the hair. Then once the hair is detected, some representative features of the colour are extracted to use a Bayesian classifier and categorize it in dark, blonde, red and white.

4.5.1 Hair detection algorithm

The very first condition we have to assume before applying the algorithm is that the faces are disposed to frontal view. This algorithm consists of several steps: face detection, eye detection, skin colour modelling and head hair colour modelling. As the three first steps are already described in previous sections, the main purpose of this section is to explain the last step.

Hair is assumed to be present at one or more of three principal locations adjacent to facial skin , this is, right, middle and left sides of the upper face, in this case, is assumed to be on the upper side of the face. The initial upper area is automatically set based on the location of the detected face and eyes, from now on it is considered as seed.

The idea is to iterate the algorithm, putting the area each step in an upper location until the colour histogram is different enough from the seed. Afterwards, the previous skin colour model is used to separate the skin pixels from the non-skin ones. Finally, the non-skin pixels are considered hair and some representative features are extracted in order to do the classification [45]. The performance of the algorithm is shown in the following flow chart:

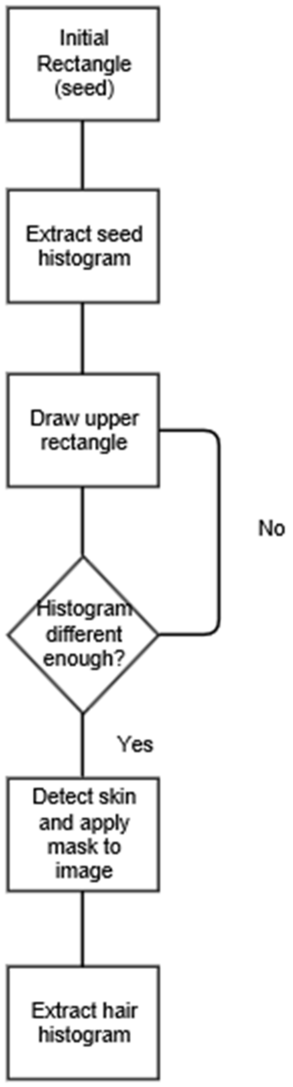


Figure 16. Flowchart for hair detection.

It is important to clarify, how the comparison between the histograms is done. OpenCV provides a function that does it, using the chi-squared distance [46]. This method implements bin-to-bin distance for comparison using the following formula

$$X^2(P,Q) = \frac{1}{2}\sum_i \frac{(P_i - Q_i)}{(P_i + Q_i)} \tag{38}$$

where P is the histogram of the seed rectangle and Q are the subsequent ones. If the histograms differ in a distance equal or higher than $X^2 = 0.1$ then we consider they are different enough and the rectangle contains hair [47].

4.5.2 Hair Colour: Naïve-Bayesian Classification

Once we have detected the pixels of the face that are skin, the next step is to isolate the ones that belong to the hair. To this end, the idea is to apply an XOR (exclusive or) operator using the original image as input and the image that only contains the skin pixels. The reason is that as the definition of the XOR states, the output is true only when the inputs differ so that the output image will only show the pixels that are not skin, this is, the hair pixels. It is important to clarify at this point that it is assumed that, as the rectangle has a sufficient small dimention to consider that everything in it that is not skin is hair.

Secondly, the features that define the hair colour are needed in order to do the classification. In this case, the image containing only the hair pixels is split into the three channels of the RGB colour space. Afterwards, the mean and standard deviation of each channel are calculated and used as the features that will take part during the classification.

Training the Bayesian Classifier

Here, the goal is to categorize the hair in dark, blonde, red and white. With this in mind, 120 different people out of the GNUD Database are chosen, 40 for each class [39]. Afterwards, the algorithm for hair detection is applied to everyone and the features that define colours are extracted (mean and standard deviation of each of the three HSV channels). These features are saved in a CSV so that they can be used in further occasions.

The last step of the training algorithm is to compute the mean and standard deviation for each feature corresponding to each class. After calculating them, they are saved in a CSV file. In the end, there are eight CSV files, two per class, one containing the means of the features and the other containing the standard deviation (See Annex G).

Using the Bayesian Classifier

The explained procedure allows the classifier to be trained only once, which decreases the runtime. The classification is done using the Bayes Rule and the Gaussian formula (see equation 26). The means and standard deviations are taken from the CSV file. In the end, we don’t get an absolute result, but the probability of the sample hair to belong to each class. Therefore, to obtain only one class, it is assumed that the sample belongs to the class showing the higher probability (See Annex C).

4.6 Template Matching for overall classification

Once all the data have been extracted, the las phase is to analyse them as a group. For this goal, a class Human with several arguments has been implemented, as well as a template matching algorithm based on a decision rule, which is the last step of the recognition of people.

4.6.1 Class Human

After the analysis of hair, skin, eye and the whole face, the colours and appearance are codified using integers. These data are saved using the class Human, which is defined by 4 arguments: the label returned by the LBPH face recognizer, the class the hair belongs to, the class the eyes belongs to and the class the skin belongs to.

Therefore, there is another database, in which each of the faces saved in the database for face recognition, has those four parameters that define it. The database is organised as the following picture shows.

LABEL	**EYES**	**SKIN**	**HAIR**
5	3	2	3
6	3	1	2

Table 3. Database for overall classification.

The first box corresponds to the label assigned in the database for the LBPH, the second one is the colour of the eyes and the third and fourth ones, are the skin and hair colours, respectively . In Figure 15, we can see that both subjects have blue eyes (class 3). Subject 5 is light skinned whereas subject 6 has dark skin. Finally, Subject 5 is red-haired while subject 6 is blonde.

This database is used to perform a template matching algorithm for the overall classification. The following table shows the code used to classify colours for eye, skin and hair.

LABEL	**1**	**2**	**3**	**4**
EYE COLOUR	Brown	Green	Blue	-
SKIN COLOUR	Dark	Light	-	-
HAIR COLOUR	Brown	Blonde	Red	White

Table 4. Codification of colours.

4.6.2 Template Matching

The overall recognition is done using a template matching algorithm, this is, using the database explained in section 4.6.1; the sample person is compared with every person saved in the database, feature per feature. Afterwards, the following decision tree is used to perform the classification

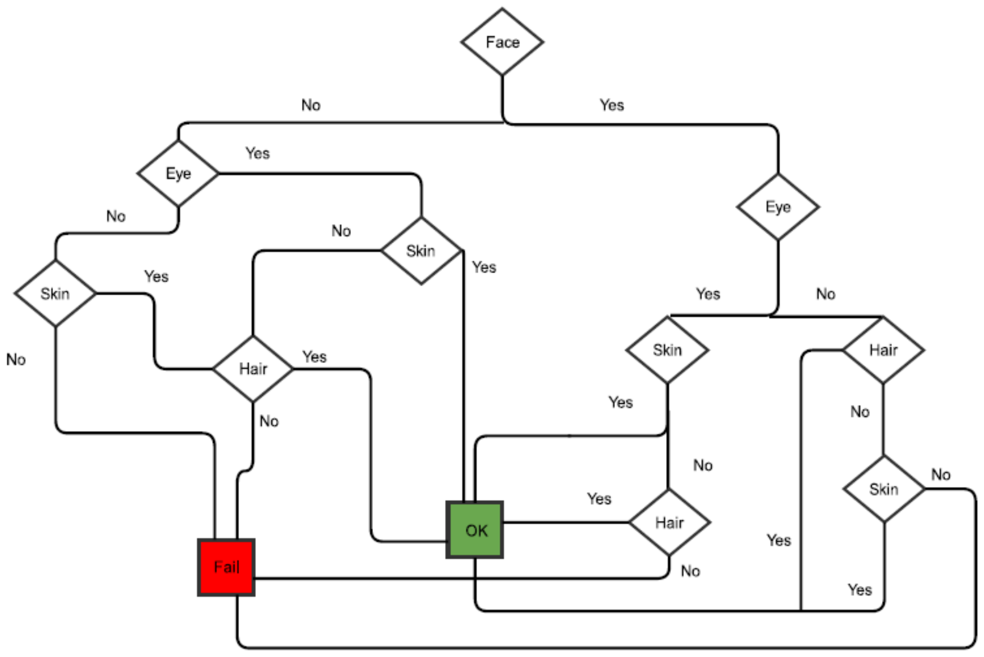


Figure 17. Decision tree usied for the overall classification.

After applying the decision tree, two situations can happen. If the sample person matches with the saved one, then the algorithm ends and the program returns the label assigned in the database. On the other hand, if the sample doesn’t match, the algorithm continues with the following person in the database.

If any matching has happened, then the algorithm ends and includes a picture of the sample person in the database, and also the values for eyes, skin and hair in the CSV file.

5. Simulation and Results

In this section, the obtained results with the implemented software will be presented. It will be done step by step, going through the four used classifiers; ending by analysing the results of the overall classification.

5.1 Skin Colour Modelling

The first step is the detection of the face using the Local Binary Patterns Cascade Classifier, which returns an image as shown in the following figure.

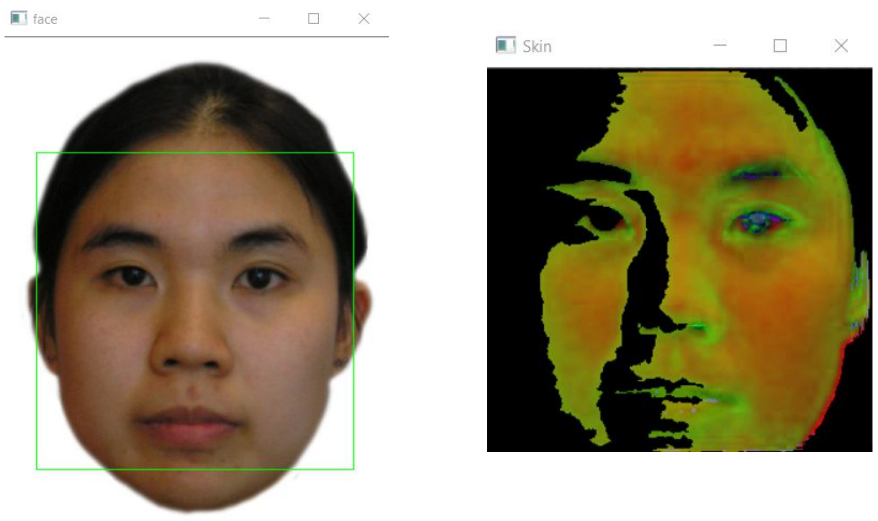


Figure 18. Detected Face (left) and cropped skin from face (right)

The next step is extracting the features of the skin i.e., mean and standard deviation from every channel out of the HSV colour space. The values for this sample are shown in the following figure, where the two first columns correspond to the hue, the following two are the saturation and the last ones the value; mean and standard deviation respectively:

RED CHANNEL		**GREEN CHANNEL**		**BLUE CHANNEL**	
Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**
7.76577	12.3239	66.89	59.3743	76.6395	68.357

Table 5. Skin parameters for the sample shown in Fig.20.

The last step is to apply the Bayes rule and perform the classification. The results obtained for the given sample are shown in the following picture, we can see how the classification is absolute for this sample, as the class "dark" shows a probability of 100%. Therefore, the sample is assigned a 1 for the skin colour.

DARK	100%
LIGHT	0%

Table 6. Classification results for skin.

In order to test the performance of the skin, classification has been done using 58 subjects out of the GNUFD database, 29 with light skin and 29 with dark skin. The result was that for each class, 23 out of 29 subjects were correctly categorized, while 6 were put in the other class.

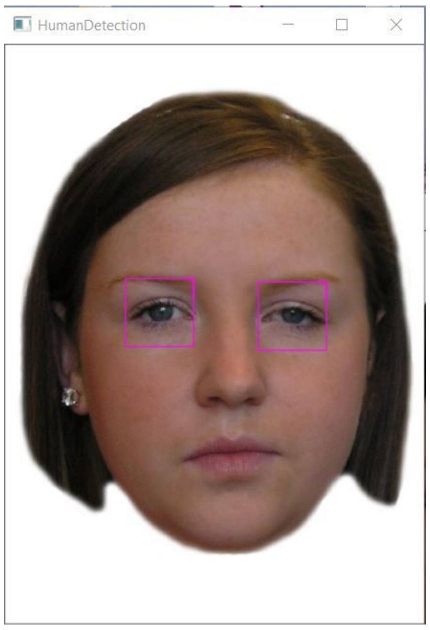
The success is of 79% for each class. These results are shown in the following table

Class	**Correct**	**Wrong**	**Success**
Dark	23	6	79%
Light	23	6	79%
Total	46	12	79%

Table 7. Analysis of success in skin classification.

5.2 Hair Colour Recognition

The first step is to detect the eyes, as the initial location of the rectangles used for hair detection will be set accordingly. Then the algorithm for hair detection is applied, obtaining an image of the upper side of the head that contains hair. As the last step, the XOR function is applied, to crop only the hair from the image.



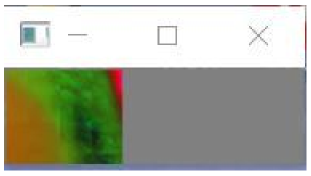


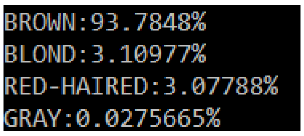
Figure 19. Detection of eyes (left) and hair cropped (right).

The features for colour classification are extracted from the left image, first splitting it and then calculating the mean and the standard deviation of every channel in the RGB colour space. The extracted data for the given sample are shown in the following figure, where the two first columns correspond to the red channel, the following two to the green one and the last ones the blue one; mean and standard deviation respectively:

RED CHANNEL		**GREEN CHANNEL**		**BLUE CHANNEL**	
Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**
161.594	29.4096	57.6469	20.0929	95.2024	58.8316

Table 8. Extracted features for hair classification.

Finally, the Bayes Rule is applied and the probability of the given hair sample to belong to each class is obtained as shown in figure 23.



```
BROWN:93.7848%
BLOND:3.10977%
RED-HAIRED:3.07788%
GRAY:0.0275665%
```

Figure 20. Results of hair classification for the given sample.

As we can see in Figure 23, the given sample is classified as brown haired with a 93 % of probability. As a result of it, the assigned label for the hair will be 1.

In order to test the hair, 146 subjects out of the GNU were used: 40 brown haired, 40 blonde, 28 red haired and 38 grey-haired,considered as a member of the class white. The results obtained are shown in the following similarity matrix:

	BROWN	BLONDE	RED	WHITE
BROWN	25	3	5	7
BLONDE	4	22	11	3
RED	2	6	19	1
WHITE	7	5	1	22

Table 9. Similarity matrix for hair colour recognition.

From the upper table, it can be said that 25 out of 40 brown haired people were properly classified, while 3 were classified as blonde, 5 as red-haired and 7 as grey-haired. For the blonde class, there is an interesting situation; 22 out of 40 people were detected as blonde, while 11 were detected as red-haired. This fact can be justified as the features for a blonde hair are very close to the ones for blonde hair in a lot of cases. This is why it is sometimes for us hard to define whether someone is blonde or red haired.

For red-haired, the situation is not as extreme as for blonde haired, because 19 out of 28 samples are correctly classified, while only 6 out of 28 are classified as blonde. Finally, considering grey-haired, 22 out of 38 people are correctly classified, while 7 are classified as brown, as sometimes the hair is more grey than white and the parameters are closer to the ones for brown haired.

As a conclusion, we can say that the implemented classifier has an accuracy of 63% for brown haired, 55 % for blonde people, 68 % for red haired and 58% for grey-haired. Hence, it follows that the overall accuracy is 60%.

5.3 Eye ColourRecognition

As for hair colour modelling, the first step is to detect the eyes using the Haar Cascade Classifier. To follow, the iris is detected using a Hough circle transform and then a neural network is used to classify the colour.

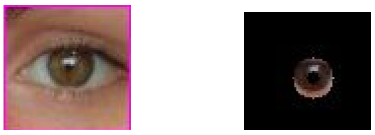


Figure 21. Detected eye(left) and cropped iris (right).

In the case of the example shown in Fig.24, the obtained label will be 1 as the colour is brown, i.e the class showing the higher probability.

In order to test the classification, 12 samples out of the GUFD database [39] were taken; 4 out of each eye colour, brown, green and blue. The results obtained are shown in the following table:

Samples	BROWN	GREEN	BLUE
1	0.799	0.0034	0.1014
2	0.1325	0.02861	0.0184
3	0.7601	0.0081	0.09059
4	0.763	0.0083	0.9235
5	0.1012	0.1103	0.7274
6	0.0026	0.8139	0.0076
7	0.0663	0.4877	0.015
8	0.0663	0.4877	0.016
9	0.837	0.00165	0.01129
10	0.00636	0.4266	0.4838
11	0.1544	0.293	0.5895
12	0.155	0.2932	0.5902

Table 10. Results of testing eye colour classification.

In the previous table, each row of the matrix corresponds to one analysed sample, while each column corresponds to one class; brown, green and blue respectively. The given numbers are the probability of each sample belonging to each class. The first four samples are brown; as shown in the table three of them are classified as brown, while one of them is classified as green.

The next four samples are green; three of them are properly classified, while the first one is classified as blue. Finally, the last four samples are blue, and they are correctly classified, except for the first one that is classified as brown.

The incorrectly classified samples are shown in the following figure:

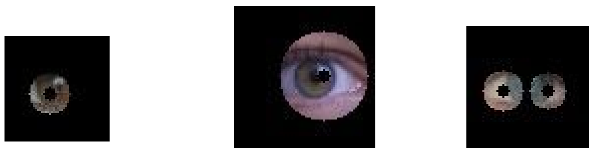


Figure 22. Incorrectly classified samples.

It is shown, that the last sample is a bad one, as the iris was not properly detected, so that the picture is darker than if it would have been properly done, therefore it is classified as brown. The other two samples are also incorrectly classified; to give a reason, it can be said that the features are probably not being extracted well as the images contain much more than only the iris. As a conclusion, it can be stated that as 3 out 4 samples are being properly classified, the success is 75%.

5.4 Overall classification

Once eye, skin, hair and face have been classified,it is time to perform the overall classification to find out who is in front of the camera. For this goal, a template matching algorithm is used as explained in section 4.6. A database is needed in order to compare the sample person with the saved ones and see if anyone matches or not.

5.4.1 Controlled conditions

First, the test is done under controlled conditions i.e., using samples of the GUFD database that were not used for training. Therefore, six persons were selected out of the not used samples for training; two with brown eyes, two with green eyes and two with blue eyes. They are shown in the following figure:

ç

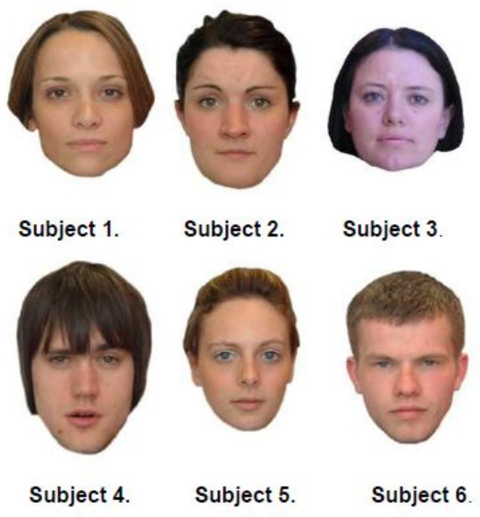


Figure 23. Database under controlled conditions.

The expected results are shown in the following table:

FACE	EYE	SKIN	HAIR
2	1	2	2
2	2	2	1
6	2	1	1
4	2	2	1
2	3	2	3
2	3	1	2

Table 11. Expected results for overall classification.

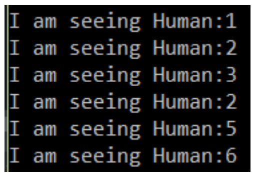
After running the software, the output results are the following:

FACE	EYE	SKIN	HAIR
1	1	2	2
2	1	2	1
3	2	1	1
4	2	2	1
5	3	2	1
6	3	2	2

Table 12. Results for the overall classification.

From this test, we can come to several conclusions. First of all, that the success of the face recognition algorithm is not very good. However, it can be used to give precision to the overall classification as one more parameter means one more condition. Secondly, the eye classification works accurately, excluding the second sample that is categorized as green; nevertheless, the eyes of the second sample are light brown, so that the parameters can be closed to the ones of a green eye.

In the case of the hair, the fifth sample is analysed as red although at first sight it would be categorized as brown. Nevertheless, after a careful observation, it can be stated that the actual colour is more copper-coloured than brown; this can be the reason why it is classified as red.The final template matching algorithm shows the following results:



```
I am seeing Human:1
I am seeing Human:2
I am seeing Human:3
I am seeing Human:2
I am seeing Human:5
I am seeing Human:6
```

Figure 24. Final Results for the overall classification.

These results are obtained after applying the template matching algorithm. It works so that, when two parameters of one of the saved people match the sample, it stops comparing and returns the label of the corresponding saved person.

For the first sample, the colour of the eyes as well as skin and hair matches with the first person, so it stops comparing and returns label 1. In the second case, face recognition, skin and hair match, but not eyes as they are categorized as green. The third sample is well classified as eyes, skin and hair match with the third person of the database. A mistake can be seen in the fourth sample person as colour of skin and hair matches with the second sample person and as a result of it the software returns a false label. If we look at the pictures of both subjects, attending to colours, they look very similar.

Finally, for the last two subjects the classification works well, as the colour of the eyes is very well categorized and colour of hair and skin allows distinguishing from the others. Hence, it follows that the software has a success of 83.3% under controlled conditions.

5.4.2 Uncontrolled conditions

In this section, the performance of the software under uncontrolled conditions will be tested. This means that pictures of random people under random conditions will be used in order to see if the software is suitable to be used in an uncontrolled environment.

To this end, six persons were included in the database. For the girls, the difference between the database and the testing was that during the testing they had their hair tied back, while for the database they didn't. In case of boys, the difference was made with the glasses; they had glasses during testing but not in the database. These pictures were taken from 6 random people, two of them (subjects 4 and 6) have different lightning conditions.

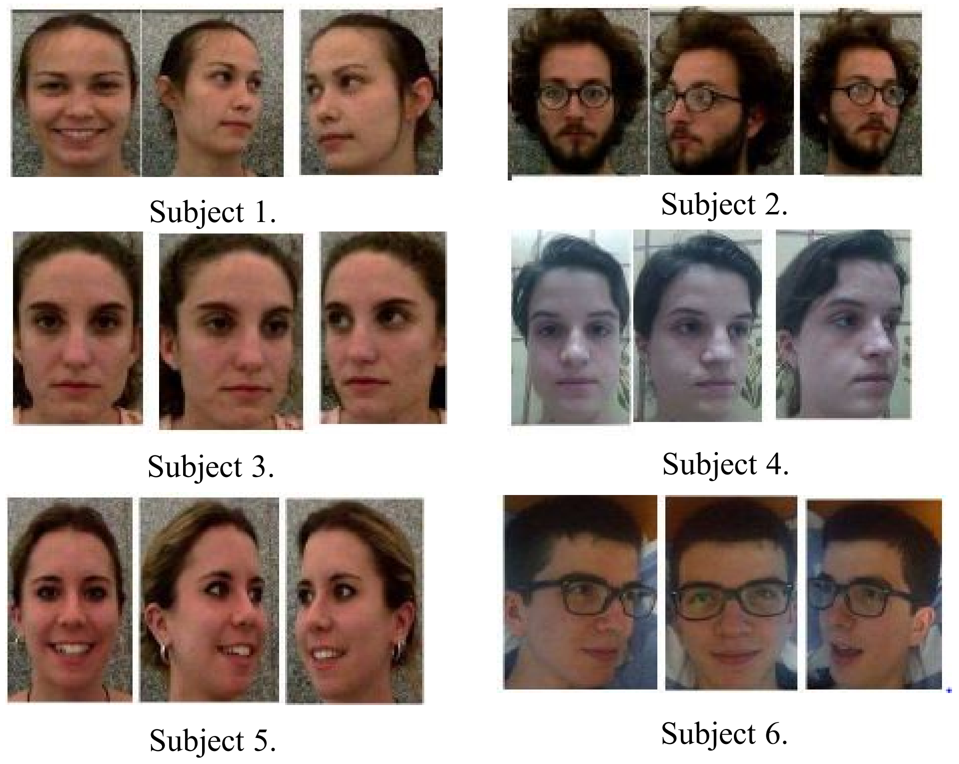


Figure 25. Database used to test under uncontrolled conditions.

5.

FACE	EYE	SKIN	HAIR	
5	x	1	3	Subject 1
x	3	1	2	
5	3	1	2	
2	x	2	3	Subject 3
x	1	2	2	
x	1	2	4	Subject 4
x	3	2	4	
x	x	2	4	
x	1	2	3	Subject 5
2	1	2	3	
5	1	2	3	

Table 13. Results for uncontrolled lighting conditions.

Analysing the previous table, it is shown that glasses bring confusion to the software, as brown eyes are analysed as green; and also to the skin is analysed as dark. For hair, it is working usually properly, however, it gets confused when the colour is more copper than brown and also when the light conditions make it to appear lighter, once is equivocally classified as white.

Besides, the eye colour classification is wrong when using pictures with the eyes turned to the left, while it goes well with front and right pictures. Skin colour classification is showed to be independent of lightning conditions as subjects are light skinned.

The external code used for face recognition also gets confused when coming to uncontrolled conditions. The main reason is that it is based on histogram features, and light conditions can change histograms; so that it works poorly in the mentioned conditions.

For the overall classification, the software doesn't recognise subject 1 with glasses when he is not front oriented, as well as for subject 3 when she is turned to the left. However, the shape of the hair doesn't affect the classification; as subject 5 is well recognized regardless of the hair's shape. To give some statistics, we can say that results showed that 65 % of times people were well classified although the shape of the hair changed. Besides, results showed that only 25 % of times the lighting conditions have an influence on the classification; moreover, it only concerns hair colour modelling. As a consequence, the effect on the overall classficiation is nearly imperceptible.

Attending to the overall classification, there is one interesting situation, as subject 5 is confused with subject 2 when she has the hair tied-back. As a conclusion, we can say that the classification is going wrong with peoples of the same colour features features, as we are only using colour parameters (eye colour, skin colour and hair colour); this means that if there is a person in the database with brown eyes, light skin and brown hair, it would classify all people with this features as him or her. This is thought to be solved using the face recognition algorithm with the Local Binary Patterns Histograms, but it is not working as expected. Thus as a solution, it would increase the accuracy to use not only colour parameters but also "shape" parameters,such as , for example, the distance between eyes, the shape of the face or the ratio between the length of the arm and the leg to distinguish between people.

6. Conclusion

This Bachelor Thesis is part of in the project TUK (Technische Universität Kiwi), whose purpose is to develop a therapeutic robot that could give autistic people help during their therapies; it is mostly thought for children but the idea can also be useful for autistic adults. This could provide them with the opportunity to be able to communicate with people in their environment.

The main goal of the present Thesis was to provide a way to make TUK able to recognise people in their environment, thus being able to know who is it working with and adapt its behaviour. The idea was considering mostly face parameters, as it is the human part that most defines a person and the one that we first see when we meet someone.

As results show, the success is high under controlled conditions, what means that working always under the same light, i.e the same room, would make TUK able to know who it is talking to. However, problems come when trying to work under uncontrolled conditions. Thus, further works can be oriented this way.

As results showed, the light affects the classification, as only parameters involving colour have been used.Therefore, the accuracy can be increased by using not only a higher number of parameters but also, human features independent of light, such as the distance between eyes, the arm to leg ratio or the shape of the hair.

Literature

[1] P. J. Phillips, H. Wechsler, J. Huang, Rauss and P. J., "The FERET database and evaluation procedure for face-recognition algorithms," *Image and Vision Computing,* vol. 16, no. 5, pp. 295-306, 1998.

[2] P. J. Phillips, J. R. Beveridge, B. A. G. G. Draper, A. J. O'Toole, D. S. Bolme, J. Dunlop, Y. M. Lui, H. Sahibzada and S. Weimer, "An introduction to the good, the bad, & the ugly face recognition challenge problem," in *IEEE International Conference on Automatic Face & Gesture*, Santa Barbara, 2011.

[3] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace:Closing the Gap to Human-Level Performance in Face Verification.," in *Procedings of the IEEE Conference in Computer Vision and Pattern Recognition*, Columbus, 2014.

[4] X. Cao, D. Wipf, F. Wem, G. Duan and J. Sun, "A Practical Transfer Learning Algorithm for Face Verification," in *IEEE International Conference on Computer Vision*, Sydney, Australia, 2013.

[5] G. B. Huang, H. Lee and E. Learned-Miller, "Learning Hierarchical Representations for FAce Verification with Convolutional Deep Belief Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence Rhode Island Convention Center Providence, RI, USA, 2012.

[6] O. M. Parkhi, K. Simonyan, A. Vedaldi and A. Zisserman, "A Compact and Discriminative Face Track Descriptor," in *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014.

[7] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems,* pp. 1097-1105, 2012.

[8] B. Moghaddam and A. P. Pentland, "Face recognition using view-based and modular eigenspaces," in *SPIE's 1994 International Symposium on Optics, Imaging and Instrumentation*, 1994.

[9] J. J. Atick and A. N. Reidlich, "Convergent algorithm for sensory receptive field development," *Neural Computation,* vol. 5, no. 1, pp. 45-60, 1993.

[10] M. Lades, J. C. Vorbruggen, J. Buhmann, J. Lange, C. von der Malsburg, R. P. Wurtz and W. Konen, “Disortion invariant object recognition inthe dinamic link architecture,” *IEEE Transactions on Computers,* vol. 42, no. 3, pp. 300-311, 1993.

[11] K. Talete, S. Kadam and A. Tikare, Efficient Face Detection using AdaBoost, 2012.

[12] T. Mita, T. Kaneki and O. Hori, “Joint Haar-like features for face detection,” in *Tenth IEEE International Conference on Computer Vision*, 2005.

[13] T. Ahohen, A. Hadid and M. Pietiäinen, “Face Recognition with Local Binary Patterns,” in *Europan Conference on Computer Vision*, Prague, 2004.

[14] I. Kukenys and B. McCane, “Support Vector Machines for Human Face Detection,” in *Proceedings of the new Zealand Computer Science Research Student Conference*, New Zealand, 2008.

[15] R. Fisher, S. Perkins, A. Walker and E. Wolfart, “ Gaussian Smoothing, ” 2000. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. [Accessed April 2016].

[16] G. Deng and L. Cahil, “An adaptative Gaussian filter for noise reduction and edge detection,” in *Nuclear Science Symposium and Medical Imaging Conference*, San Francisco, 1993.

[17] pacarro2, “3D Convolution of the Gaussian Kernel,” 1 December 2011. [Online]. Available: https://imagineatness.wordpress.com/2011/12/01/3d-convolution-and-the-gaussian-kernel/. [Accessed April 2016].

[18] R. Fisher, s. Perkins, A. Walker and E. Wolfart, “Laplacian of Gaussian,” 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm. [Accessed April 2016].

[19] M. Juneja and S. Parvinder, “Performance Evaluation of Edge detection techniques for images in spacial domain,” *International Journal of Computer Theory and Engineering,* vol. 1, no. 5, p. 614, 2009.

[20] R. Debosmit, “Edge Detection in Digital Image Processing,” 2013.

[21] P. Viola and M. Jones, “Rapid Object Detection using Boosted Cascade of simple Features,” in *Procedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.

[22] Intel Corporation, Open Computer Vision Library Reference Manual, USA, 2000.

[23] P. I. Wilson and D. J. Fernández, “Facial Feature Detection Using Haar Classifiers,” *Journal of Computing Sciences in Colleges,* vol. 21, no. 4, pp. 127-133, 2006.

[24] J. Gama and P. Barnzil, “Cascade Generalization,” *Machine Learning,* pp. 315-343, 2000.

[25] K. P. Murphy, *Naive Bayes classifier,* 2006.

[26] s.-l. developers, “Naive Bayes,” scikit-learn, 2010-2014. [Online]. Available: http://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed June 2016].

[27] D. J. Hand and K. Yu, “Idiot's Bayes- Not so stupid after all?,” *International Statistical Review,* vol. 69, no. 3, pp. 385-398, 2001.

[28] D. J. MacKay, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003.

[29] G. P. Zhang, “Neural Networks for Classification: A Survey,” *IEEE Transactions on Systems, Man and Cybernetics,* vol. 30, no. 4, pp. 451-462, 200.

[30] R. F., “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological Review 65,* vol. 65, no. 6, pp. 386-408, 1958.

[31] A. Honkela, “Multi Layer Perceptrons,” 30 05 2001. [Online]. Available: https://www.hiit.fi/u/ahonkela/dippa/node41.html#eq:basicmlp. [Accessed June 2016].

[32] V. G. Gudise and G. K. Venayagamoorthy, “Comparison of Particle Swarm Optimization and Backpropagation as Training,” in *Swarm Intelligence Symposium*, Indianapolis, 2003.

[33] D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning representations by back-propagating errors,” *Letters to nature,* vol. 5, no. 3, p. 1, 1986.

[34] G. Bradski and A. Kaehler, Learning OpenCV, O'Reilly Media, Inc., 2008.

[35] T. Ahonen, A. Hadid and M. Pietikäinen, “Face Description with Local Binary Patterns : Application to Face Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 28, no. 12, pp. 2037-2040, 2006.

[36] OpenCV Development Team, “Face Recognition with OpenCV,” OpenCV, 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#id22. [Accessed April 2016].

[37] OpenCv Development Team, “Cascade Classification: Haar Feature-based Cascade Classifier for Object Detection,” 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html. [Accessed April 2016].

[38] OpenCV Development Team, “Cascade Classification,” OpenCV, 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html. [Accessed April 2016].

[39] University of York, “York FaceVar Lab,” [Online]. Available: http://www.facevar.com/downloads/gufd. [Accessed June 2016].

[40] N. Singh, D. Gandhi and K. Pal Singh, “Iris Recognition using a Canny Edge Detection and a Hough Circle Transform,” *International Journal of Advances in Engineering and Technology,*

vol. 1, no. 2, pp. 221-228, 2011.

[41] OpenCV Development Team, “Image Processing : Feature Detection,” OpenCV, 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny. [Accessed April 2016].

[42] OpenCV Development Team, “Neural Networks,” OpenCV, 2014. [Online]. Available: http://docs.opencv.org/3.0-beta/modules/ml/doc/neural_networks.html. [Accessed June 2016].

[43] V. Oliveira and A. Conci, “Skin Detection using HSV color space,” *H. Pedrini, & J. Marques de Carvalho, Workshops of Sibgrapi,* pp. 1-2, 2009.

[44] OpenCV Development Team, “Structural Analysis and Shape Descriptors,” 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=drawcontours#drawcontours. [Accessed May 2016].

[45] Y. Yacob and L. S. Davis, “Detection and Analysis of Hair,” *IEEE Transactions on Patterns Analysis and Machine Intelligence,* vol. 28, no. 7, pp. 1164-1169, 2008.

[46] OpenCV Development Team, “OpenCV Documentation (Histograms),” 2011. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html?highlight=comparehist#comparehist. [Accessed March 2016].

[47] O. Pele and M. Werman, “The Quadratic -Chi Histogram Distance Family,” in *European Conference on computer vision*, Crete (Greece), 2010.

Declaration

Hereby, I declare, this present work has been drawn up without inadmissible aid of third parties and without usage of other than mentioned resources. Further sources or indirectly appropriated data and concepts are identified by stating the source.

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

This work has not been presented to other examination procedures, neither nationally, nor in foreign countries, in the same or in a similar form.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

[illegible]

Vienna, 12th. August.2016 Patricia Javierre

Annex A

The present annex contains the code of the main file, where the loop is included. Moreover, the trainig of the neural network is included in this file outside the loop: detection of eyes and face, as well as iris. This is the file where the Bayesian classifiers and the neural networks are used. Besides, here can also be found the function where the intensity values of the channels of the colour spaces are calculated, as well as the method used to detetect hair. In the system, it is called Visual.cpp.

```
#pragma comment(lib, "MSCOREE.lib")


#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include "opencv2/features2d/features2d.hpp"
#include <opencv2/ml.hpp>
#include "colour.h"
#include "Detection.h"
#include "Human.h"
#include "Class.h"
#include "BayesCEye.h"
#include "BayesHair.h"
#include "BayesCSkin.h"




#using <mscorlib.dll>
#using <System.dll>
#using <System.Data.dll>
#using <System.Xml.dll>
#using <System.Xml.Linq.dll>
#using <System.Runtime.Serialization.dll>
```

```
//used namspaces
using namespace std;
using namespace cv; //opencv
using namespace cv::face;//facerecognitionLHBP
using namespace System;
using namespace System::IO; //StreamWriters
using namespace ml;





//Declarations
void detectandDisplay(Mat frame);
colour RGBHistogram(Mat image);
vector <Mat> DrawRectangle(Rect rect, Mat frame, int where);
colour detectHair(vector <Mat> Rectangles, Mat frame, Rect input);
void read_csv(string& filename, vector <Mat>& images, vector <int>& labels, char separator = ';');
void read_csvTEST(string& filename, vector <string>& images, vector <int>& labels, char separator = ';');
Boolean NonZeroColour(colour colour);
void Write2CSV(vector <int> test);
void Write2CSV2(vector <Human> test);
void SaveIris(string path); // Only used to extract iris from every image of the database to train the neural network

string face_cascadeclas = "lbpcascade_frontalface.xml";
string eyes_cascadeclas = "haarcascade_eye_tree_eyeglasses.xml";
string upperbody_cascadeclas = "haarcascade_upperbody.xml";
string lowerbody_cascadeclas = "haarcascade_lowerbody.xml";
string profileface_cascadeclas = "haarcascade_profileface.xml";
string fullbody_cascadeclas = "haarcascade_fullbody.xml";
string fn_csv = "FacesDatabase.csv";//FacesDataBase.csv
string Test = "DataTestNN.csv"; //DataTestNN.csv
CascadeClassifier face_cascade,eyes_cascade, profileface_cascade;
Detection myskin,myhair;
vector<Mat> images;
vector<string> imagesTest2,imagesNN, imagesTest;
vector<int> labelsTest, LabelsNN, LabelsTest, TestEye, TestHair, TestSkin,labels;
vector <Human> TestGeneral;
Ptr<ANN_MLP> nnetwork = ANN_MLP::create();

/*The main function contains the loop where the funtion runs and the Neural Network is trained outside that loop*/

int main(int argc, const char ** argv) {
		/*Read the csv file containing the database for the Face Recognition algorithm (LHBP)*/
		try {
				read_csv(fn_csv, images, labels);
		}
		catch (cv::Exception& e) {
				cerr <<  e.msg << endl;
				//exit(1);
		}
		cout<< images.size()<< endl;
		for (int i = 0; i < images.size(); i++) {
				if (images[i].empty()) {
						cout << "not DB face" << i << endl;
				}
```

```
	}

	/*Loading cascade Classifiers*/
	eyes_cascade.load(eyes_cascadeclas);
	face_cascade.load(face_cascadeclas);
	profileface_cascade.load(profileface_cascadeclas);
	/*Testing if the classifiers have been load correctly*/
	if (!face_cascade.load(face_cascadeclas)) {
		cout << "Error loading face classifier" << endl;
		return -3;
	}
	if (!eyes_cascade.load(eyes_cascadeclas)) {
		cout << "Error loading eyes classifier" << endl;
		return -2;
	}
	if (!profileface_cascade.load(profileface_cascadeclas)) {
		cout << "Error loading profile face classifier" << endl;
		return -6;
	}
	/*Training the Neural Network for eye colour classification*/
	string path = "IrisTest.csv"; //CSV file containing the training database for the neural Network.
	read_csvTEST(path, imagesNN, LabelsNN, ';');
	Mat LabelsMat = Mat::zeros(72, 3, CV_32F);// Matrix of labels (0 in the column of the label for the corre-
sponding image)
	Mat ImagesMat = Mat::zeros(0, 0, CV_32S);// Every row contains an image (iris)
	cout << images.size() << endl;
	for (int i = 0; i < imagesNN.size(); i++) {
		Size size(30, 30);
		Mat image = imread(imagesNN[i]);
		Boolean Empty = image.empty();
		cout << Empty << endl;
		imshow("Try", image);
		Mat imageHSV;
		cvtColor(image, imageHSV, CV_BGR2HSV);
		Mat ImgRE;
		resize(imageHSV, ImgRE, size);
		imshow("resized", ImgRE);
		vector <Mat> HSV;
		split(ImgRE, HSV);
		Mat reshaped;
		cout << ImagesMat.size() << endl;
		reshaped = HSV[2].reshape(1, 1);
		ImagesMat.push_back(reshaped);
		cout << ImagesMat.size() << endl;
		cout << i << ';' << LabelsNN[i] << endl;
		LabelsMat.at<float>(Point(LabelsNN[i] - 1, i)) = 1.0;
	}
	int layers_d[] = { 900,400,100,20,3 };
	Mat layers = Mat(1, 5, CV_32S);
	layers.at<int>(0, 0) = layers_d[0];
	layers.at<int>(0, 1) = layers_d[1];
	layers.at<int>(0, 2) = layers_d[2];
	layers.at<int>(0, 3) = layers_d[3];
	layers.at<int>(0, 4) = layers_d[4];
	Mat Img32;
	ImagesMat.convertTo(Img32, CV_32F);
	nnetwork->setTrainMethod(cv::ml::ANN_MLP::BACKPROP);
	nnetwork->setLayerSizes(layers);
	nnetwork->setActivationFunction(ANN_MLP::SIGMOID_SYM, 1, 1);
	Mat weights(1, ImagesMat.rows, CV_32F, Scalar::all(1));
	Ptr<TrainData> tdata = TrainData::create(Img32, ROW_SAMPLE,
		LabelsMat);
	Mat output;
	try {
		int iterations = nnetwork->train(tdata);
		cout << "Number of iterations :" << iterations << endl;
	}
	catch (cv::Exception& e) {
		cout << e.what() << endl;
	}

		/*Second step is reading the video stream*/
		/*VideoCapture capture(0);//0=default video camera.
		if (!capture.isOpened()) {
			cout << "cannot open the video cam" << endl;
	}*/
		read_csvTEST(Test, imagesTest2, labelsTest);
		cout << imagesTest2.size() << endl;
			for (int i=0; i<imagesTest2.size();i++){
				try{
					Mat frame= imread(imagesTest2[i]);
					if (frame.empty()) {
						cout << "No captured frame" << endl;
					}

					detectandDisplay(frame);
					if (waitKey(30) == 27) {
						//Write.close();
						break; } //if ESC then stop debugging.
					}
				catch (cv::Exception& e) {
					cout << e.what() << endl;

				}
				}

			//Write2CSV2(TestGeneral);
	}

void detectandDisplay(Mat frame) {
	vector <Rect> faces,eyes,profileface;
	vector <Mat>  RectanglesForHair;
	Mat frame_gray;
	size_t i;


	Ptr <FaceRecognizer> model = createLBPHFaceRecognizer();
	model->train(images, labels);

	cvtColor(frame, frame_gray, CV_BGR2GRAY, 0); // convert image to grayscale.
	equalizeHist(frame_gray, frame_gray); //equalize histogram of gray scale.

	/*Applying object detection in image.*/
	face_cascade.detectMultiScale(frame_gray, faces, 1.1, 3, 0 | CV_HAAR_DO_CANNY_PRUNING,
cvSize(30, 30));
	eyes_cascade.detectMultiScale(frame_gray, eyes, 1.1, 3, 0 | CV_HAAR_DO_CANNY_PRUNING,
cvSize(30, 30));
```

```
        profileface_cascade.detectMultiScale(frame_gray, profileface, 1.1, 2, 0 |
CV_HAAR_DO_CANNY_PRUNING, cvSize(30, 30));


        Human Person = Human();
        for (int i = 0; i < faces.size(); i++) {
                Rect face_i = faces[i];
                Mat facegray = frame_gray(face_i);//Crop the face from the image.
                int prediction = model->predict(facegray);
                Person.setFace(prediction);
                rectangle(frame, face_i, CV_RGB(0, 255, 0), 1);
                string box_text = format("prediction = %d", prediction);
                //calculate position for text
                int pos_x = max(face_i.tl().x - 10, 0);
                int pos_y = max(face_i.tl().y - 10, 0);
                putText(frame, box_text, Point(pos_x, pos_y), FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255,
0), 2.0);
                Mat faceroi = frame(faces[i]);
                imwrite("face.jpeg", faceroi);
        }
        colour Hair;
        for (i = 0; i < eyes.size(); i++) {
                Rect eye_i = eyes[i];
                RectanglesForHair = DrawRectangle(eye_i, frame, 0);
                Hair = detectHair(RectanglesForHair, frame, eye_i);
                rectangle(frame, eye_i, CV_RGB(255, 0, 255), 1);
                Mat eyeroi = frame(eyes[i]);
                imwrite("eye.jpeg", eyeroi);
        }
        if (NonZeroColour(Hair)) {
                int colourHair = ClassifyBayesHair(Hair);
                Person.setHairColour(colourHair);
        }
        else {
                Person.setHairColour(0);
        }

        //TestHair.push_back(colourHair);
        /*detection of  iris and classification of colour*/
        vector <Vec3f> circles;
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        Mat src = imread("eye.jpeg", 1);
        Mat src_gray, dst, dstHSV;
        colour eyecolour;
        cvtColor(src, src_gray, CV_RGB2GRAY, 0); // Input has to be in gray scale.
        GaussianBlur(src_gray, src_gray, Size(1, 1), 0, 0, BORDER_DEFAULT);
        Canny(src_gray, src_gray, 0, 10); // canny edge detector.
        HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT, 1, 20, 50, 30, 0, 25);
        /*void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, dou-
ble param1=100, double param2=100, int minRadius=0, int maxRadius=0 )
                method – Detection method to use. Currently, the only implemented method is
CV_HOUGH_GRADIENT , which is basically 21HT , described in [Yuen90].
                dp – Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 ,
the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and
height.
                src_gray.rows / 8->minDist – Minimum distance between the centers of the detected circles. If
the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too
large, some circles may be missed.
```

```
                    param1 – First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher
threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
                    param2 – Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the
accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be
detected. Circles, corresponding to the larger accumulator values, will be returned first.
                    minRadius – Minimum circle radius.
                    maxRadius – Maximum circle radius.
                    */
          Mat output;
          Mat ImagesMatTest;
          if (circles.size() > 0) {
                    for (size_t j = 0; j < circles.size(); j++) {
                              Point center(cvRound(circles[j][0]), cvRound(circles[j][1]));
                              int radius = cvRound(circles[j][2]);
                              circle(src, center, 3, Scalar(0, 0, 0), -1, 8, 0);
                              circle(src, center, radius, Scalar(0, 0, 0), 1, 8, 0);
                              Mat mask = Mat::zeros(src.rows, src.cols, CV_8UC1);
                              circle(mask, center, radius, Scalar(255, 255, 255), -1, 8, 0);
                              src.copyTo(dst, mask);
                              Mat dsthsv;
                              cvtColor(dst, dsthsv, CV_RGB2HSV);
                              Mat ImgRE2;
                              Size size(30, 30);
                              resize(dsthsv, ImgRE2, size);
                              vector <Mat> HSV2;
                              split(ImgRE2, HSV2);
                              Mat reshaped2;
                              reshaped2 = HSV2[2].reshape(0, 1);
                              ImagesMatTest.push_back(reshaped2);
                              Mat Img322;
                              ImagesMatTest.convertTo(Img322, CV_32F);

                              nnetwork->predict(Img322, output);
                              //compare(results, LabelsMatTest, output, CMP_EQ);
                    }
          }
          float value = 0.0;
          int AreEqual = 0;
          for (int k = 0; k < output.rows; k++) {
                    vector<float> Row;
                    for (int l = 0; l < output.cols; l++) {
                              float valor = (output.at<float>(Point(l, k)));
                              Row.push_back(valor);
                              //cout<<"valor: "<<valor<<endl;
                    }
                    float MaxValue = Row[0];
                    int Class = 1;
                    for (int i = 0; i < Row.size(); i++) {
                              if (Row[i] > MaxValue) {
                                        MaxValue = Row[i];
                                        Class = i + 1;
                              }
                    }
                    cout << "Eye Colour:" << Class << endl;
                    Person.setEyeColour(Class);
          }

                    //Detect and display histogram of skin (using face)
                    colour faceColour;
```

```
				Mat forskin = imread("face.jpeg");
				Mat forskinHSV;
				cvtColor(forskin, forskinHSV, CV_BGR2HSV);
				Mat skinMat;
				skinMat = myskin.getDetection(forskin);
				Mat applyhist;
				forskinHSV.copyTo(applyhist, skinMat); // selecting only the pixels of the face that are skin
				faceColour = RGBHistogram(applyhist);
				if (NonZeroColour(faceColour)) {
					int skinColour = ClassifySkin(faceColour);
					Person.SetSkinColour(skinColour);

				}
				else {
					cout << "Invalid colour " << endl;
					Person.SetSkinColour(0);
				}

				TestGeneral.push_back(Person);
				namedWindow("Skin Image", WINDOW_AUTOSIZE);
				imshow("Skin Image", applyhist);

				namedWindow("IrisDetection", WINDOW_AUTOSIZE);
				imshow("IrisDetection", src);

				namedWindow("HumanDetection", WINDOW_AUTOSIZE);
				imshow("HumanDetection", frame);

				Classify(Person);
}

Boolean NonZeroColour(colour colour) {
		Boolean zero = false;
		if ((colour.getMeanH()> 1) || (colour.getSTDH() >1) || (colour.getMeanS() >1) || (colour.getSTDS() >1) ||
(colour.getMeanV() >1) || (colour.getSTDV() >1)) {
				zero = true;
		}
		return zero;
}
vector <Mat > DrawRectangle(Rect rect, Mat frame, int where) {

		vector <Mat> Rectangles; //vector is created with undefined size--> use pushback to include elements.
		Rect up = rect + Point(50, -40 - where);
		Mat upROI = frame(up);
		Rectangles.push_back(upROI);
		return Rectangles;
}

colour detectHair(vector <Mat> Rectangles, Mat frame,Rect input) {
		vector <Mat> SeedRectangles;
		vector <Mat> NewRectangles;
		Mat hsvup;
		Mat hsvright;
		Mat hsvleft;
		int channels[] = { 0,1 };
		// Quantisize saturation to 32 levels, hue to 30
		int hbins = 30; int sbins = 32;
		int histsize[] = { hbins, sbins };
		// hue varies from 0 to 179.
```

```
    float hranges[] = { 0,180 };
    //saturation varies from 0 (black-white) to 255(S=1) (pure color)
    float sranges[] = { 0,256 };
    const float* ranges[] = { hranges,sranges };
    // multi-dimensional dense multi-channel array for storing histogram.
    MatND histupseed;
    MatND histup;
    //Convert rectangles (regions) to HSV space & calculate histogram of each region = seed histogram.
    cvtColor(Rectangles [0], hsvup, CV_BGR2HSV);
    calcHist(&hsvup, 1, channels, Mat(), histupseed, 2, histsize, ranges, true, false);
    // Move rectangles and calculate new histogram.
    NewRectangles = DrawRectangle(input, frame, 40);
    //Recalculate Histogram moving rectangles
    cvtColor(NewRectangles[0], hsvup, CV_BGR2HSV);
    calcHist(&hsvup, 1, channels, Mat(), histup, 2, histsize, ranges, true, false);
    //Compare Histogram seed and new using chi-squared distance (statistical method -> it would not be per-
fect).
    double distUP=compareHist(histupseed, histup, CV_COMP_CHISQR);
    //Value of 0.1 taken from paper about histogram distance (Pele & Werman)
    if ((distUP > 0.1) ){//Añadir left y right si uso los tres rectángulos.
        NewRectangles = DrawRectangle(input, frame, 40);
        cvtColor(NewRectangles[0], hsvup, CV_BGR2HSV);
        calcHist(&hsvup, 1, channels, Mat(), histup, 2, histsize, ranges, true, false);
        double distUP = compareHist(histupseed, histup, CV_COMP_CHISQR);
    }
    //Find countours of hair in upper rectangle.
    colour hairColour;
    Mat forhair = hsvup;
    Mat hairMat,onlyhair;
    hairMat = myhair.getDetection(Rectangles[0]);
    Mat bgr;
    cvtColor(hairMat, bgr, CV_GRAY2BGR);
    Mat hsv (hairMat.size(),CV_8UC3);
    cvtColor(bgr, hsv, CV_BGR2HSV);
    Mat hairMask(hairMat.size(), CV_8UC3);
    //IMP: All the images (input and output) have to be of the same type.
    bitwise_xor(Rectangles[0], hsv, hairMask, Mat());// selecting only the pixels of the rectangle that are hair.
    hsvup.copyTo(onlyhair, hairMask);
    Mat RGB;
    cvtColor(onlyhair, RGB, CV_HSV2RGB);
    hairColour= RGBHistogram(onlyhair);
    namedWindow("Upper hair", WINDOW_AUTOSIZE);
    imshow("Upper hair", onlyhair);
    return hairColour;
}
colour RGBHistogram(Mat image) {
    Scalar mean, std;
    meanStdDev(image, mean, std, Mat());
    vector <Mat> HSV;
    split(image, HSV);
    double MaxValueH, MaxValueS, MaxValueV;
    minMaxLoc(HSV[0], 0, &MaxValueH, 0, 0);
    minMaxLoc(HSV[1], 0, &MaxValueS, 0, 0);
    minMaxLoc(HSV[2], 0, &MaxValueV, 0, 0);
    colour farbe(mean[0], std[0],  mean[1], std [1], mean[2], std[2]);
    return farbe;
}
void Write2CSV(vector <int> test) {
    ofstream myfile("EyeMatrix.csv");
```

```
		for (int i = 0; i < test.size(); i++) {
			myfile << test[i] << endl;
		}
		myfile.close();
}
void Write2CSV2(vector <Human> test) {
		ofstream myfile("PersonTest.csv");
		for (int i = 0; i < test.size(); i++) {
			myfile << test[i].getFace() << ';'<< test[i].getEyeColour() << ';' << test[i].getSkinColour() <<
';'<< test[i].getHairColour() << endl;
		}
		myfile.close();
}
void read_csv(string& filename, vector <Mat>& images, vector <int>& labels, char separator) {
		ifstream file(filename.c_str(), ifstream::in);
		if (!file) {
			string error_message = "No valid input file";
			CV_Error(CV_StsBadArg, error_message);
		}
		string line, path, classlabel;
		while (getline(file, line)) {
			stringstream liness(line);
			getline(liness, path, separator);
			getline(liness, classlabel);
			if (!path.empty() && !classlabel.empty()) {
				images.push_back(imread(path, 0));
				labels.push_back(atoi(classlabel.c_str()));
			}

		}
}
void read_csvTEST(string& filename, vector <string>& images, vector <int>& labels, char separator) {
		ifstream file(filename.c_str(), ifstream::in);
		if (!file) {
			string error_message = "No valid input file";
			CV_Error(CV_StsBadArg, error_message);
		}
		string line, path, classlabel;
		while (getline(file, line)) {
			stringstream liness(line);
			getline(liness, path, separator);
			getline(liness, classlabel);
			if (!path.empty() && !classlabel.empty()) {
				images.push_back(path);
				labels.push_back(atoi(classlabel.c_str()));
			}

		}
}
```

Annex B

In this file, called Detection.cpp, the code used for the detection of skin pixels can be found; it can be shown how the biggest contour is found in order to avoid false results.

```
#include "Detection.h"
#include "opencv2\opencv.hpp"
using namespace cv;
Detection::Detection(void) {
}
Detection:: ~Detection(void){}
Mat Detection::getDetection(Mat input) {
		blur(input ,input, Size(3, 3));//blur an image using normalized box filter
		Mat hsv;
		// Convert to HSV
		cvtColor(input, hsv, COLOR_BGR2HSV);

		Mat inter;
		Mat afterMask;
		inRange(hsv, Scalar(0, 10, 60), Scalar(20, 150, 255), inter);
		Mat canny_output;
		vector <vector<Point>> contours;
		vector <Vec4i> hierarchy;
		findContours(inter, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,
0));//Retrieves all the contours and reconstructs the full hierarchy of nested contours.
		int c = findBiggestContour(contours);
		Mat mask= Mat::zeros(input.size(), CV_8UC1);
		drawContours(mask, contours, c, Scalar(255), -1, 8, hierarchy, 0, Point());
		//bitwise_and(hsv, hsv, afterMask, inter);
		return mask;
}
int Detection::findBiggestContour(vector<vector<Point>> contours){
				int indexOfBiggestContour = -1;
				int sizeBiggestContour = 0;
				for (int i = 0; i < contours.size(); i++) {
						if (contours[i].size() > sizeBiggestContour) {
								sizeBiggestContour = contours[i].size();
								indexOfBiggestContour = i;
						}

				}
				return indexOfBiggestContour;
		}
```

Annex C

Bayesian Classifier for hair color (BayesHair.cpp)

```
#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include "colour.h"
#include "Detection.h"
#include "Human.h"
#include "Class.h"
#include "BayesCEye.h"
#include "BayesHair.h"
//Global variables
vector<vector<float>> features1H;
vector<vector<float>> features2H;
vector<vector<float>> features3H;
vector<vector<float>> features4H;
float piH = 3.14159265358979323846;
int ClassifyBayesHair(colour forClass) {
		vector <float> colour;
		colour.push_back(forClass.getMeanH());
		colour.push_back(forClass.getSTDH());
		colour.push_back(forClass.getMeanS());
		colour.push_back(forClass.getSTDS());
		colour.push_back(forClass.getMeanV());
		colour.push_back(forClass.getSTDV());
		vector <float> MeanClass1;
		vector <float> MeanClass2;
		vector <float> MeanClass3;
		vector <float> MeanClass4;
		vector <float> VarianceClass1;
		vector <float> VarianceClass2;
		vector <float> VarianceClass3;
		vector <float> VarianceClass4;
		MeanClass1 = ReadCSV("Mean1Hair.csv");
		MeanClass2 = ReadCSV("Mean2Hair.csv");
		MeanClass3 = ReadCSV("Mean3Hair.csv");
		MeanClass4 = ReadCSV("Mean4Hair.csv");
		VarianceClass1 = ReadCSV("Variance1Hair.csv");
		VarianceClass2 = ReadCSV("Variance2Hair.csv");
		VarianceClass3 = ReadCSV("Variance3Hair.csv");
		VarianceClass4 = ReadCSV("Variance4Hair.csv");

		vector <float> Prob(4);
		Prob[0] = 1;
		Prob[1] = 1;
		Prob[2] = 1;
		Prob[3] = 1;
		for (int i = 0; i <6; i++) { /*Using the Gaussian formula of the Bayes Rule*/
			Prob[0] = Prob[0] * (1 / sqrt(2 * piH*VarianceClass1[i])) * exp(-powf((colour[i] - Mean-
Class1[i]), 2) / (2 * VarianceClass1[i]));
			Prob[1] = Prob[1] * (1 / sqrt(2 * piH*VarianceClass2[i])) * exp(-powf((colour[i] - Mean-
Class2[i]), 2) / (2 * VarianceClass2[i]));
			Prob[2] = Prob[2] * (1 / sqrt(2 * piH*VarianceClass3[i])) * exp(-powf((colour[i] - Mean-
Class3[i]), 2) / (2 * VarianceClass3[i]));
			Prob[3] = Prob[3] * (1 / sqrt(2 * piH*VarianceClass4[i])) * exp(-powf((colour[i] - Mean-
Class4[i]), 2) / (2 * VarianceClass4[i]));
		}
		//Weighted probabilities
		vector <float> ProbW(4);
		for (int j = 0; j < Prob.size(); j++) {
			ProbW[j] = (Prob[j] / (Prob[0] + Prob[1] + Prob[2] + Prob[3])) * 100;
		}
		cout << "BROWN:" << ProbW[0] << "%" << endl;
		cout << "BLOND:" << ProbW[1] << "%" << endl;
		cout << "RED-HAIRED:" << ProbW[2] << "%" << endl;
		cout << "GRAY:" << ProbW[3] << "%" << endl;

		float maxProb = ProbW[0];
		int Colour = 1;
		for (int i = 0; i < ProbW.size(); i++) {
			if (ProbW[i] > maxProb) {
				maxProb = ProbW[i];
				Colour = i + 1;
			}
		}
		//cout << Colour << endl;
		return Colour;
}
vector<float> ReadCSV(string path) {
		ifstream file(path, ifstream::in);
		if (!file) {
			string error_message = "I can't find the data";
			CV_Error(CV_StsBadArg, error_message);
		}
		vector <float> data;
		char separator = ';';
		string line;
		while (getline(file, line)) {
			stringstream liness(line);
			float dato;
			string part;
			while (getline(liness, part, separator)) {
```

```
                    float part1 = atof(part.c_str());
                    dato = part1;
                }
                data.push_back(dato);
            }
        return data;
    }
```

Annex D

Bayes Classifier for skin color (BayesSkin.cpp)

```
#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include "colour.h"
#include "Detection.h"
#include "Human.h"
#include "Class.h"
#include "BayesHair.h"
#include <cmath>
vector <float> Mean1Skin(6);
vector <float> Mean2Skin(6);
vector <float> Variance1Skin(6);
vector <float> Variance2Skin(6);

float piS = 3.14159265358979323846;

int ClassifySkin(colour ForClass) {
		vector<float> unknown(6);
		unknown[0] = ForClass.getMeanH();
		unknown[1] = ForClass.getSTDH();
		unknown[2] = ForClass.getMeanS();
		unknown[3] = ForClass.getSTDS();
		unknown[4] = ForClass.getMeanV();
		unknown[5] = ForClass.getSTDV();
		/*for (int i = 0; i < unknown.size(); i++) {
				cout << unknown[i] << endl;
		}*/

		Mean1Skin = ReadCSV("Mean1Skin.csv");
		Mean2Skin = ReadCSV("Mean2Skin.csv");
		Variance1Skin = ReadCSV("Variance1Eye.csv");
		Variance2Skin = ReadCSV("Variance2Eye.csv");
```

```
	vector <float> Prob(2);
	Prob[0] = 1;
	Prob[1] = 1;

	for (int i = 0; i<6; i++) { /*Using the Gaussian formula of the Bayes Proability Theorem*/
		//Prob[0] = Prob[0] * (1 / sqrt(2 * piH*VarianceClass1[i])) * exp(-powf((colour[i] - Mean-
Class1[i]), 2) / (2 * VarianceClass1[i]));
		Prob[0] = Prob[0] * (1 / sqrt(2 * piS*Variance1Skin[i])) * exp(-powf((unknown[i] -
Mean1Skin[i]), 2) / (2 * Variance1Skin[i]));
		Prob[1] = Prob[1] * (1 / sqrt(2 * piS*Variance2Skin[i])) * exp(-powf((unknown[i] -
Mean2Skin[i]), 2) / (2 * Variance2Skin[i]));
		//cout << Prob[0] << Prob[1] << endl;
	}
	//Weighted probabilities
	vector <float> ProbW(2);
	ProbW[0] = 100;
	ProbW[1] = 100;
	for (int j = 0; j < Prob.size(); j++) {
		ProbW[j] = 100 * (Prob[j] / (Prob[0] + Prob[1]));
	}

	/*cout << "DARK:" << ProbW[0] << "%" << endl;
	cout << "HELL:" << ProbW[1] << "%" << endl;*/

	float maxProb = ProbW[0];
	int Colour = 1;
	for (int i = 0; i < ProbW.size(); i++) {
		if (ProbW[i] > maxProb) {
			maxProb = ProbW[i];
			Colour = i + 1;
		}
	}

	cout << "Skin: Class" << Colour << endl;
	return Colour;
}
```

Annex E

Template matching algorithm for overall classification (Classifier.cpp)

```
#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include "colour.h"
#include "Detection.h"
#include "Human.h"
#include "Class.h"
#include "Classifier.h"
vector <Human> ForCSV;
void WritetoCSV2(vector <Human> Write);
void read_csv2(string& filename, vector <string>& images2, vector <int>& labels2, char separator2);
vector <Human> ReadCSVfeatures(void);
void WritetoCSV(vector <Human> Write);
void includeToFaces(string Path, int Label);


using namespace std;
using namespace cv;

//void WritetoCSV2(vector <Human> Write);

void Classify(Human ForClass) {
		//cout << " I am classifying" << endl;
		vector <Human> Template = ReadCSVfeatures();
		//cout << "I have " << Template.size() << " persons to compare" << endl;
		ForCSV.push_back(ForClass);
		WritetoCSV2(ForCSV);
		Boolean HumanMatch = false;
		Boolean End = false;
		int Label;
		while (!End) {
```

```
//cout << "I have come" << endl;
for (int i = 0; i < Template.size(); i++) {
        if (!HumanMatch) {
                //cout << "I am here" << i << endl;
                Human Data = Template[i];
                Boolean face = (Template[i].getFace() == ForClass.getFace());
                Boolean Skin = (Template[i].getSkinColour() == ForClass.getSkinColour());
                Boolean Eye = (Template[i].getEyeColour() == ForClass.getEyeColour());
                Boolean Hair = (Template[i].getHairColour() == ForClass.getHairColour());
                cout << face << Eye << Skin << Hair << endl;
                if (face) {
                        if (Eye) {
                                HumanMatch = true;
                        }
                        else {
                                if (Skin) {
                                        if (Hair){
                                        HumanMatch = true;
                                        }
                                }
                        }
                }
                else {
                        if (Eye) {
                                if (Skin) {
                                        HumanMatch = true;
                                        //End = true;
                                }
                                else {
                                        if (Hair) {
                                                HumanMatch = true;
                                        }
                                }
                        }
                        else {
                                if (Hair) {
                                        if (Skin) {
                                                HumanMatch = true;
                                                //End = true;
                                        }
                                }

                        }
                }

                        if (HumanMatch) {
                                Label = Template[i].getFace();
                                cout << "I am seeing Human:" << Label << endl;
                                End = true;
                        }
                }

        }
        End = true;
}
if (!HumanMatch) {
        cout << "I don't know you!" << endl;
        Label = (Template[Template.size() - 1].getFace()) + 1;
        ForClass.setFace(Label);
```

```
				Template.push_back(ForClass);
				Mat face = imread("face.jpeg", 1);
				ostringstream forpath;
				forpath << "C:/Users/Usuario/Pictures/TUK/" << Label << ".jpeg";
				string path = forpath.str();
				imwrite(path, face);
				includeToFaces(path, Label);
				cout << "It is the first time I am with you! Nice to meet you!" << endl;
				cout << "From now on you are" << Label << endl;
			}
			WritetoCSV(Template);
		}


vector <Human> ReadCSVfeatures (void){
		ifstream file("Features.csv", ifstream::in);
		if (!file) {
			string error_message = "I can't find the data";
			CV_Error(CV_StsBadArg, error_message);
		}

		char separator = ';';
		string line;
		vector <Human> Existent;
		while (getline(file, line)) {
			stringstream liness(line);
			vector <float> data;
			string part;
			while (getline(liness, part, separator)){
				data.push_back(atoi(part.c_str())); // stod-> string to double / atof-> char to double.
				}


			Human Person(data[0], data[1], data [2], data[3]);
			Existent.push_back(Person);
		}
		return Existent;
}

void WritetoCSV(vector <Human> Write) {
		ofstream myfile("Features.csv");
		for (int i = 0; i < Write.size();i++) {
			int face = Write [i].getFace();
			int EyeColour = Write[i].getEyeColour();
			int SkinColour = Write[i].getSkinColour();
			int HairColour = Write[i].getHairColour();
			myfile << face << ';' << EyeColour << ';' << SkinColour<< ';' << HairColour <<  endl;
		}
		myfile.close();
}

void includeToFaces(string Path, int Label) {
		vector <string> faces;
		vector <int> names;
		string fileName = "FacesDataBase.csv";
		read_csv2(fileName, faces, names, ';');
		faces.push_back(Path);
		names.push_back(Label);
		ofstream write("FacesDataBase.csv");
		for (int i = 0; i < faces.size(); i++) {
				write << faces[i] << ';' << names[i] << endl;
		}
}
void read_csv2(string& filename, vector <string>& images2, vector <int>& labels2, char separator2) {
		ifstream file(filename.c_str(), ifstream::in);
		if (!file) {
				string error_message = "No valid input file";
				CV_Error(CV_StsBadArg, error_message);
		}
		string line, path, classlabel;
		while (getline(file, line)) {
				stringstream liness(line);
				getline(liness, path, separator2);
				getline(liness, classlabel);
				if (!path.empty() && !classlabel.empty()) {
						images2.push_back(path);
						labels2.push_back(atoi(classlabel.c_str()));
				}

		}
}
void WritetoCSV2(vector <Human> Write) {
		ofstream myfile("Features2.csv");
		for (int i = 0; i < Write.size(); i++) {
				int face = Write[i].getFace();
				int EyeColour = Write[i].getEyeColour();
				int SkinColour = Write[i].getSkinColour();
				int HairColour = Write[i].getHairColour();
				myfile << face << ';' << EyeColour << ';' << SkinColour << ';' << HairColour << ';' << endl;
		}
		myfile.close();
}
```

Annex F

Training Bayesian Classifier for Skin

```
#pragma comment(lib, "MSCOREE.lib")


#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include "colour.h"
#include "Human.h"
#include "Detection.h"




#using <mscorlib.dll>
#using <System.dll>


//used namspaces
using namespace std;
using namespace cv; //opencv
using namespace cv::face;//facerecognitionLHBP
using namespace System;




//Declarations
void detectandDisplay(int blabel,Mat frame);
```

```
colour RGBHistogram(Mat image);
void read_csv(string& filename, vector <string>& images, vector <int>& labels, char separator = ';');
Boolean NonZeroColour(colour colour);
void WritetoCSV(vector <Human> Write);
float CalculateMean(vector <float> feature);
vector<float> ClassMean(vector<vector<float>> features);
float CalculateMean(vector <float> feature);
vector <float> ClassVariance(vector<vector<float>> classfeatures, vector<float> ClassMean);
void WritetoCSV(vector <float> One, string path);
float CalculateVariance(vector< float> features, float mean);
void ReadCSVEye(void);


//Variables
string face_cascadeclas = "lbpcascade_frontalface.xml";
string fn_csv = "Skin.csv";
CascadeClassifier face_cascade;
vector<string> images;
vector<int> labels;
vector <Human> Training;
vector<vector<float>> features1;
vector<vector<float>> features2;
vector<vector<float>> features3;
vector<vector<float>> features4;
vector<float> ClassMean1;
vector<float> ClassMean2;
vector<float> ClassVariance1;
vector<float> ClassVariance2;
Detection myskin;

//main function
int main(int argc, const char ** argv) {
		try {
			read_csv(fn_csv, images, labels);
		}
		catch (cv::Exception& e) {
			cerr << e.msg << endl;
		}
		if (images.size() <= 1) {
			string error_message = "At least 2 images needed. Please add more images to your data base";
			CV_Error(CV_StsError, error_message);
		}
		// First step is loading cascades
		face_cascade.load(face_cascadeclas);
		if (!face_cascade.load(face_cascadeclas)) {
			cout << "Error loading face classifier" << endl;
			return -2;
		}
		//Second step is reading the video stream
		//0=default video camera.
		for (int i = 0; i < images.size(); i++) {
			try {
				char lab[30];
				Mat frame = imread(images[i]);
				_itoa(labels[i], lab, 10);
				imshow(lab, frame);
				int label = labels[i];
				if (frame.empty()) {
					cout << "No captured frame" << endl;
				}
				detectandDisplay(label, frame);
				waitKey(32);
				if (waitKey(30) == 27) {
					break;
				} //if ESC then stop debugging.
			}
			catch (cv::Exception& e) {
				cout << e.what() << endl;
				cout << Training.size() << endl;
			}
		}
		WritetoCSV(Training);
		cout << Training.size() << endl;
		ReadCSVEye();
		ClassMean1 = ClassMean(features1);
		ClassMean2 = ClassMean(features2);
		ClassVariance1 = ClassVariance(features1, ClassMean1);
		ClassVariance2 = ClassVariance(features2, ClassMean2);
		WritetoCSV(ClassMean1, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean1Skin.csv");
		WritetoCSV(ClassMean2, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean2Skin.csv");
		WritetoCSV(ClassVariance1, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance1Skin.csv");
		WritetoCSV(ClassVariance2, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance2Skin.csv");
}
 //function for detecting and displaying
void detectandDisplay(int cLabel,Mat frame){
		Mat frame_gray;
		vector <Rect> faces;
		cvtColor(frame, frame_gray, CV_BGR2GRAY, 0); // convert image to grayscale.
		equalizeHist(frame_gray, frame_gray); //equalize histogram of gray scale.
		//Applying object detection in frame.
		face_cascade.detectMultiScale(frame_gray, faces, 1.1, 3, 0 | CV_HAAR_DO_CANNY_PRUNING,
cvSize(30, 30));
		Human Person = Human();
		Person.setLabel(cLabel);
		//Face Detection
		for (int i = 0; i < faces.size(); i++) {
			Rect face_i = faces[i];
			Mat facegray=frame_gray(face_i);//Crop the face from the image.
			rectangle(frame, face_i, CV_RGB(0, 255, 0), 1);
			Mat faceroi = frame(faces[i]);
			imwrite("face.jpeg", faceroi);
			}
		//Detect and display histogram of skin(using face)
			colour faceColour;
			Mat forskin = imread("face.jpeg");
			Mat forskinHSV;
			cvtColor(forskin,forskinHSV, CV_BGR2HSV);
			Mat skinMat;
			skinMat=myskin.getDetection(forskin);
			Mat applyhist;
			forskinHSV.copyTo(applyhist, skinMat); // selecting only the pixels of the face that are skin
			faceColour = RGBHistogram(applyhist);
			if (NonZeroColour (faceColour)){
				Person.setEyeColour(faceColour);
				}
				else {
					cout << "Invalid colour label "<< cLabel << endl;
				}

			Training.push_back(Person);
			imshow("face", frame);
			imshow("Skin", applyhist);
}
void ReadCSVEye(void) {
			ifstream file("SkinDataBase.csv", ifstream::in);
			if (!file) {
				string error_message = "I can't find the data";
				CV_Error(CV_StsBadArg, error_message);
			}
			//Class 1
			vector <float> feature11;
			vector <float> feature12;
			vector <float> feature13;
			vector <float> feature14;
			vector <float> feature15;
			vector <float> feature16;
			//Class 2
			vector <float> feature21;
			vector <float> feature22;
			vector <float> feature23;
			vector <float> feature24;
			vector <float> feature25;
			vector <float> feature26;
			char separator = ';';
			string line;
			while (getline(file, line)) {// Mientras haya lineas.
				stringstream liness(line);
				vector <float> data;
				string part;
				while (getline(liness, part, separator)) { //Hasta que se acabe la linea.
					data.push_back(atof(part.c_str())); // stod-> string to double / atof-> char to double.
				}//cada vez que acabe una linea la meto dentro del vector de vectore float.
				if (data[0] == 1) {
					feature11.push_back(data[1]);
					feature12.push_back(data[2]);
					feature13.push_back(data[3]);
					feature14.push_back(data[4]);
					feature15.push_back(data[5]);
					feature16.push_back(data[6]);
				}
				if (data[0] == 2) {
					feature21.push_back(data[1]);
					feature22.push_back(data[2]);
					feature23.push_back(data[3]);
					feature24.push_back(data[4]);
					feature25.push_back(data[5]);
					feature26.push_back(data[6]);
				}

			}
			//Class1
			features1.push_back(feature11);
			features1.push_back(feature12);
```

```
		features1.push_back(feature13);
		features1.push_back(feature14);
		features1.push_back(feature15);
		features1.push_back(feature16);
		//cout << "Size of Class 1:" << features1.size() << endl;
		//Class2
		features2.push_back(feature21);
		features2.push_back(feature22);
		features2.push_back(feature23);
		features2.push_back(feature24);
		features2.push_back(feature25);
		features2.push_back(feature26);
}
void WritetoCSV(vector <Human> Write) {
		ofstream myfile("SkinDataBase.csv");
		for (int i = 0; i < Write.size(); i++) {
				colour EyeColour = Write[i].getEyeColour();
				myfile << Write[i].getLabel()<<';' << EyeColour.getMeanH() << ';' << EyeColour.getSTDH() <<
';' << EyeColour.getMeanS() << ';' << EyeColour.getSTDS() << ';' << EyeColour.getMeanV() << ';' << EyeCol-
our.getSTDV() << endl;
		}
		myfile.close();
}
Boolean NonZeroColour(colour colour) {
		Boolean zero = false;
		if ((colour.getMeanH()> 1) || (colour.getSTDH() >1) || (colour.getMeanS() >1) || (colour.getSTDS() >1) ||
(colour.getMeanV() >1) || (colour.getSTDV() >1)) {
				zero = true;
		}
		return zero;
}
void WritetoCSV(vector <float> One, string path) {

		ofstream myfile(path);
		for (int i = 0; i < (One.size()); i++) {
				myfile << One[i] << endl;
		}
		myfile.close();
}
colour RGBHistogram(Mat image) {
		Scalar mean, std;
		meanStdDev(image, mean, std, Mat());
		vector <Mat> RGB;
		split(image, RGB);
		double MaxValueH, MaxValueS, MaxValueV;
		minMaxLoc(RGB[0], 0, &MaxValueH, 0, 0);
		minMaxLoc(RGB[1], 0, &MaxValueS, 0, 0);
		minMaxLoc(RGB[2], 0, &MaxValueV, 0, 0);
		colour farbe(mean[0], std[0],  mean[1], std [1], mean[2], std[2]);
		return farbe;
}
void read_csv(string& filename, vector <string>& images, vector <int>& labels, char separator) {
		ifstream file(filename.c_str(), ifstream::in);
		if (!file) {
				string error_message = "No valid input file";
				CV_Error(CV_StsBadArg, error_message);
		}
		string line, path, classlabel;
		while (getline(file, line)) {
				stringstream liness(line);
				getline(liness, path, separator);
				getline(liness, classlabel);
				if (!path.empty() && !classlabel.empty()) {
					images.push_back(path);
					labels.push_back(atoi(classlabel.c_str()));
				}
		}
}
float CalculateMean(vector <float> feature) {
		float mean = 0;
		for (int i = 0; i < feature.size(); i++) {
				mean = mean + feature[i];
		}
		mean = mean / (feature.size());
		return mean;
}
vector<float> ClassMean(vector<vector<float>> features) {
		vector <float> ClassMean;
		for (int i = 0; i < features.size(); i++) {
				float mean = CalculateMean(features[i]);
				ClassMean.push_back(mean);
		}
		return ClassMean;
}
float CalculateVariance(vector< float> features, float mean) {
		float variance=0;
		float sumatorio=0;
		for (int i = 0; i < features.size(); i++) {
				sumatorio += (features[i] - mean) *(features[i] - mean);
		}
		int n = features.size();
		variance = sumatorio/n;
		return variance;
}
vector <float> ClassVariance(vector<vector<float>> classfeatures, vector<float> ClassMean) {
		vector<float> ClassVariance;
		for (int i = 0; i < classfeatures.size(); i++) {
				float variance = CalculateVariance(classfeatures[i], ClassMean[i]);
				ClassVariance.push_back(variance);
		}
		return ClassVariance;
}
```

Annex G

Training Bayesian Classifier for Hair

```
#pragma comment(lib, "MSCOREE.lib")
#include <opencv2/core/core.hpp>
#include <opencv2/face.hpp>
#include <opencv2/face/facerec.hpp>
#include <opencv2/face/predict_collector.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/videoio/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <fstream>
#include <sstream>
#include "colour.h"
#include "Human.h"
#include "Detection.h"
#using <mscorlib.dll>
#using <System.dll>
#using <System.Data.dll>
#using <System.Xml.dll>
#using <System.Xml.Linq.dll>
#using <System.Runtime.Serialization.dll>
//used namspaces
using namespace std;
using namespace cv; //opencv
using namespace cv::face;//facerecognitionLHBP
using namespace System;
using namespace System::Data;
using namespace System::Data::Sql;
using namespace System::Data::SqlTypes;
using namespace System::Data::Common;
using namespace System::Data::SqlClient; //Databases
using namespace System::Runtime::Serialization; //Object Serialization
using namespace System::IO; //StreamWriters
//Declarations
void detectandDisplay(int blabel,Mat frame);
colour RGBHistogram(Mat image);
void read_csv(string& filename, vector <string>& images, vector <int>& labels, char separator = ';');
Boolean NonZeroColour(colour colour);
void WritetoCSV(vector <Human> Write);
```

```
float CalculateMean(vector <float> feature);
vector<float> ClassMean(vector<vector<float>> features);
float CalculateMean(vector <float> feature);
vector <float> ClassVariance(vector<vector<float>> classfeatures, vector<float> ClassMean);
void WritetoCSV(vector <float> One, string path);
float CalculateVariance(vector< float> features, float mean);
void ReadCSVEye(void);
vector <Mat > DrawRectangle(Rect rect, Mat frame, int where);
colour detectHair(vector <Mat> Rectangles, Mat frame, Rect input);
//Variables
string eyes_cascadeclas = "haarcascade_eye_tree_eyeglasses.xml";
string fn_csv = "Hair.csv";
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
vector<string> images;
vector<int> labels;
vector <Human> Training;
vector<vector<float>> features1;
vector<vector<float>> features2;
vector<vector<float>> features3;
vector<vector<float>> features4;
vector<float> ClassMean1;
vector<float> ClassMean2;
vector<float> ClassMean3;
vector<float> ClassMean4;
vector<float> ClassVariance1;
vector<float> ClassVariance2;
vector<float> ClassVariance3;
vector<float> ClassVariance4;
vector <Mat>  RectanglesForHair;
Detection myhair;
//main function
int main(int argc, const char ** argv) {
		// Read in the data
		try {
			read_csv(fn_csv, images, labels);
			cout << images.size() << endl;
			cout << labels.size() << endl;
		}
		catch (cv::Exception& e) {
			cerr << e.msg << endl;
		}
		if (images.size() <= 1) {
			string error_message = "At least 2 images needed. Please add more images to your data base";
			CV_Error(CV_StsError, error_message);
		}
		// First step is loading cascades
		eyes_cascade.load(eyes_cascadeclas);
		if (!eyes_cascade.load(eyes_cascadeclas)) {
			cout << "Error loading eyes classifier" << endl;
			return -2;
		}
		for (int i = 0; i < images.size(); i++) {
			try {
				char lab[30];
				Mat frame = imread(images [i]);
				_itoa(labels[i], lab,10);
				imshow(lab, frame);
				int label = labels[i];
```

```
                                        if (frame.empty()) {
                                        cout << "No captured frame" << endl;
                                        }
                                detectandDisplay(label,frame);
                                waitKey(32);
                                if (waitKey(30) == 27) {
                                        break;
                                } //if ESC then stop debugging.
                        }
                        catch (cv::Exception& e) {
                                cout << e.what() << endl;
                                cout << Training.size() << endl;
                        }
                }
                WritetoCSV(Training);
                cout << Training.size() << endl;
                ReadCSVEye();
                ClassMean1 = ClassMean(features1);
                ClassMean2 = ClassMean(features2);
                ClassMean3 = ClassMean(features3);
                ClassMean4 = ClassMean(features4);
                ClassVariance1 = ClassVariance(features1, ClassMean1);
                ClassVariance2 = ClassVariance(features2, ClassMean2);
                ClassVariance3 = ClassVariance(features3, ClassMean3);
                ClassVariance4 = ClassVariance(features4, ClassMean4);
                WritetoCSV(ClassMean1, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean1Hair.csv");
                WritetoCSV(ClassMean2, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean2Hair.csv");
                WritetoCSV(ClassMean3, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean3Hair.csv");
                WritetoCSV(ClassMean4, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Mean4Hair.csv");
                WritetoCSV(ClassVariance1, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance1Hair.csv");
                WritetoCSV(ClassVariance2, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance2Hair.csv");
                WritetoCSV(ClassVariance3, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance3Hair.csv");
                WritetoCSV(ClassVariance4, "C:/Users/Usuario/Documents/Cuarto/Final Pro-
jekt/Project1/Project1/Variance4Hair.csv");
}
 //function for detecting and displaying
void detectandDisplay(int cLabel,Mat frame){
                vector <Rect> eyes;
                Mat frame_gray;
                cvtColor(frame, frame_gray, CV_BGR2GRAY, 0); // convert image to grayscale.
                equalizeHist(frame_gray, frame_gray); //equalize histogram of gray scale.
                //Applying object detection in frame.
                eyes_cascade.detectMultiScale(frame_gray, eyes, 1.1, 3, 0 | CV_HAAR_DO_CANNY_PRUNING,
cvSize(30, 30));
                Human Person = Human();
                Person.setLabel(cLabel);
                if (eyes.empty()) {
                        colour Zero(0, 0, 0, 0, 0, 0);
                        Person.setEyeColour(Zero);
                }
                for ( int i = 0; i < eyes.size(); i++) {
                        Rect eye_i = eyes[i];
```

```
			rectangle(frame, eye_i, CV_RGB(255, 0, 255), 1);
			Mat eyeroi = frame(eyes[i]);
			imwrite("eye.jpeg", eyeroi);
			RectanglesForHair=DrawRectangle(eye_i, frame,0);
			Mat framehsv;
			cvtColor(frame, framehsv, CV_BGR2HSV);
			try {
				colour Hair = detectHair(RectanglesForHair, frame, eye_i);
				if (NonZeroColour(Hair)){
				Person.setEyeColour(Hair);
				}
			}catch (cv::Exception &e){
				colour Zero(2, 2, 2, 2, 2, 2);
				Person.setEyeColour(Zero);
			}
			rectangle(frame, eye_i, CV_RGB(255, 0, 255), 1);
		}
		namedWindow("HumanDetection", WINDOW_AUTOSIZE);
		imshow("HumanDetection", frame);
}
void ReadCSVEye(void) {
		ifstream file("HairDataBase.csv", ifstream::in);
		if (!file) {
			string error_message = "I can't find the data";
			CV_Error(CV_StsBadArg, error_message);
		}
		//Class 1
		vector <float> feature11;
		vector <float> feature12;
		vector <float> feature13;
		vector <float> feature14;
		vector <float> feature15;
		vector <float> feature16;
		//Class 2
		vector <float> feature21;
		vector <float> feature22;
		vector <float> feature23;
		vector <float> feature24;
		vector <float> feature25;
		vector <float> feature26;
		//Class 3
		vector <float> feature31;
		vector <float> feature32;
		vector <float> feature33;
		vector <float> feature34;
		vector <float> feature35;
		vector <float> feature36;
		//Class 4
		vector <float> feature41;
		vector <float> feature42;
		vector <float> feature43;
		vector <float> feature44;
		vector <float> feature45;
		vector <float> feature46;
		char separator = ';';
		string line;
		while (getline(file, line)) {
			stringstream liness(line);
			vector <float> data;
		string part;
		while (getline(liness, part, separator)) {
			data.push_back(atof(part.c_str()));
		}
		if (data[0] == 1) {
			feature11.push_back(data[1]);
			feature12.push_back(data[2]);
			feature13.push_back(data[3]);
			feature14.push_back(data[4]);
			feature15.push_back(data[5]);
			feature16.push_back(data[6]);
		}
		if (data[0] == 2) {
			feature21.push_back(data[1]);
			feature22.push_back(data[2]);
			feature23.push_back(data[3]);
			feature24.push_back(data[4]);
			feature25.push_back(data[5]);
			feature26.push_back(data[6]);
		}
		if (data[0] == 3) {
			feature31.push_back(data[1]);
			feature32.push_back(data[2]);
			feature33.push_back(data[3]);
			feature34.push_back(data[4]);
			feature35.push_back(data[5]);
			feature36.push_back(data[6]);
		}
		if (data[0] == 4) {
			feature41.push_back(data[1]);
			feature42.push_back(data[2]);
			feature43.push_back(data[3]);
			feature44.push_back(data[4]);
			feature45.push_back(data[5]);
			feature46.push_back(data[6]);
		}
	}
	//Class1
	features1.push_back(feature11);
	features1.push_back(feature12);
	features1.push_back(feature13);
	features1.push_back(feature14);
	features1.push_back(feature15);
	features1.push_back(feature16);
	//Class2
	features2.push_back(feature21);
	features2.push_back(feature22);
	features2.push_back(feature23);
	features2.push_back(feature24);
	features2.push_back(feature25);
	features2.push_back(feature26);
	//Class3
	features3.push_back(feature31);
	features3.push_back(feature32);
	features3.push_back(feature33);
	features3.push_back(feature34);
	features3.push_back(feature35);
	features3.push_back(feature36);
	//Class3
```

```
		features4.push_back(feature41);
		features4.push_back(feature42);
		features4.push_back(feature43);
		features4.push_back(feature44);
		features4.push_back(feature45);
		features4.push_back(feature46);
}
void WritetoCSV(vector <Human> Write) {
		ofstream myfile("HairDataBase.csv");
		for (int i = 0; i < Write.size(); i++) {
				colour EyeColour = Write[i].getEyeColour();
				myfile << Write[i].getLabel()<<';' << EyeColour.getMeanH() << ';' << EyeColour.getSTDH() <<
';' << EyeColour.getMeanS() << ';' << EyeColour.getSTDS() << ';' << EyeColour.getMeanV() << ';' << EyeCol-
our.getSTDV() << endl;
		}
		myfile.close();
}
Boolean NonZeroColour(colour colour) {
		Boolean zero = false;
		if ((colour.getMeanH()> 1) || (colour.getSTDH() >1) || (colour.getMeanS() >1) || (colour.getSTDS() >1) ||
(colour.getMeanV() >1) || (colour.getSTDV() >1)) {
				zero = true;
		}
		return zero;
}
void WritetoCSV(vector <float> One, string path) {

		ofstream myfile(path);
		for (int i = 0; i < (One.size()); i++) {
				myfile << One[i] << endl;
		}
		myfile.close();
}
colour RGBHistogram(Mat image) {
		Scalar mean, std;
		meanStdDev(image, mean, std, Mat());
		vector <Mat> RGB;
		split(image, RGB);
		double MaxValueH, MaxValueS, MaxValueV;
		minMaxLoc(RGB[0], 0, &MaxValueH, 0, 0);
		minMaxLoc(RGB[1], 0, &MaxValueS, 0, 0);
		minMaxLoc(RGB[2], 0, &MaxValueV, 0, 0);
		colour farbe(mean[0], std[0],  mean[1], std [1], mean[2], std[2]);
		return farbe;
}
void read_csv(string& filename, vector <string>& images, vector <int>& labels, char separator) {
		ifstream file(filename.c_str(), ifstream::in);
		if (!file) {
				string error_message = "No valid input file";
				CV_Error(CV_StsBadArg, error_message);
		}
		string line, path, classlabel;
		while (getline(file, line)) {
				stringstream liness(line);
				getline(liness, path, separator);
				getline(liness, classlabel);
				if (!path.empty() && !classlabel.empty()) {
						images.push_back(path);
						labels.push_back(atoi(classlabel.c_str()));
```

```
                        }
                }
}
float CalculateMean(vector <float> feature) {
                float mean = 0;
                for (int i = 0; i < feature.size(); i++) {
                        mean = mean + feature[i];
                }
                mean = mean / (feature.size());
                return mean;
}
vector<float> ClassMean(vector<vector<float>> features) {
                vector <float> ClassMean;
                for (int i = 0; i < features.size(); i++) {
                        float mean = CalculateMean(features[i]);
                        ClassMean.push_back(mean);
                }
                return ClassMean;
}
float CalculateVariance(vector< float> features, float mean) {
                float variance=0;
                float sumatorio=0;
                for (int i = 0; i < features.size(); i++) {
                        sumatorio += (features[i] - mean) *(features[i] - mean);
                }
                int n = features.size();
                variance = sumatorio/n;
                return variance;
}
vector <float> ClassVariance(vector<vector<float>> classfeatures, vector<float> ClassMean) {
                vector<float> ClassVariance;
                for (int i = 0; i < classfeatures.size(); i++) {
                        float variance = CalculateVariance(classfeatures[i], ClassMean[i]);
                        ClassVariance.push_back(variance);
                }
                return ClassVariance;
}
vector <Mat> DrawRectangle(Rect rect, Mat frame, int where) {
                vector <Mat> Rectangles;
                Rect up;
                up.width = (rect.width);
                up.height = (rect.height);
                up=rect + Point(50, -40 - where);
                Mat upROI = frame(up);
                Rectangles.push_back(upROI);
                return Rectangles;
}
colour detectHair(vector <Mat> Rectangles, Mat frame, Rect input) {
                vector <Mat> SeedRectangles;
                vector <Mat> NewRectangles;
                Mat hsvup;
                Mat hsvright;
                Mat hsvleft;
                int channels[] = { 0,1 };
                // Quantisize saturation to 32 levels, hue to 30
                int hbins = 30; int sbins = 32;
                int histsize[] = { hbins, sbins };
                // hue varies from 0 to 179.
                float hranges[] = { 0,180 };
```

```
	//saturation varies from 0 (black-white) to 255(S=1) (pure color)
	float sranges[] = { 0,256 };
	const float* ranges[] = { hranges,sranges };
	// multi-dimensional dense multi-channel array for storing histogram.
	MatND histupseed;
	MatND histrightseed;
	MatND histleftseed;
	MatND histup;
	MatND histright;
	MatND histleft;
	//Convert rectangles (regions) to HSV space & calculate histogram of each region = seed histogram.
	cvtColor(Rectangles[0], hsvup, CV_BGR2HSV);
	calcHist(&hsvup, 1, channels, Mat(), histupseed, 2, histsize, ranges, true, false);
	// Move rectangles and calculate new histogram.
	NewRectangles = DrawRectangle(input, frame, 20);
	//Recalculate Histogram moving rectangles
	cvtColor(NewRectangles[0], hsvup, CV_BGR2HSV);
	calcHist(&hsvup, 1, channels, Mat(), histup, 2, histsize, ranges, true, false);
	//Compare Histogram seed and new using chi-squared distance
	double distUP = compareHist(histupseed, histup, CV_COMP_CHISQR);
	//Value of 0.1 taken from paper about histogram distance (Pele & Werman)
	if ((distUP > 0.1)) {
		NewRectangles = DrawRectangle(input, frame, 40);
		cvtColor(NewRectangles[0], hsvup, CV_BGR2HSV);
		double distUP = compareHist(histupseed, histup, CV_COMP_CHISQR);
	}
	//Find countours of hair in upper rectangle.
	colour hairColour;
	Mat forhair = hsvup;
	Mat hairMat, onlyhair;
	hairMat = myhair.getDetection(Rectangles[0]);
	Mat bgr;
	cvtColor(hairMat, bgr, CV_GRAY2BGR);
	Mat hsv(hairMat.size(), CV_8UC3);
	cvtColor(bgr, hsv, CV_BGR2HSV);
	Mat hairMask(hairMat.size(), CV_8UC3);
	//IMP: All the images (input and output) have to be of the same type.
	bitwise_xor(Rectangles[0], hsv, hairMask, Mat());// selecting only the pixels of the rectangle that are hair.
	hsvup.copyTo(onlyhair, hairMask);
	Mat RGB;
	cvtColor(onlyhair,RGB, CV_HSV2RGB);
	hairColour = RGBHistogram(onlyhair);
	namedWindow("Upper hair", WINDOW_AUTOSIZE);
	imshow("Upper hair", onlyhair);
	return hairColour;
}
```