KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Felipe Ballester Tomás

COMPUTER VISION AND FACE RECOGNITION

Bachelor’s Thesis 2010

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information Technology

BALLESTER TOMAS, FELIPE	Computer Vision and Face Recognition
Bachelor's Thesis	40 pages + 10 pages of appendices
Supervisor	Teemu Saarelainen, Senior Lecturer
Commissioned by	Kymenlaakso University of Applied Sciences
December 2010	
Keywords	computer vision, software development, face recognition, database, OpenCV

Computer vision is a rapidly growing field, partly because of the affordable hardware (cameras, processing power) and partly because vision algorithms are starting to mature. This field started with the motivation to study how computers process images and how to apply this knowledge to develop useful programs.

The purposes of this study were to give valuable knowledge for those who are interested in computer vision, and to implement a facial recognition application using the OpenCV library. Every effort has been made to define the basic structure of the application as clearly as possible in this study.

The application was designed to work on Windows Systems and it was developed by means of the C++ programming language because of its object-oriented structure and its high performance. The images and other functions were created using the open source OpenCV library. For the implementation of the facial recognition a database was also needed.

The resulting application was able to capture real time images and to realize continuous recognition of faces.

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka

BALLESTER TOMAS, FELIPE	Tietokonenäkö ja kasvojentunnistus
Opinnäytetyö	40 sivua + 10 liitesivua
Työn ohjaaja	Lehtori Teemu Saarelainen
Toimeksiantaja	Kymenlaakson ammattikorkeakoulu
Joulukuu 2010	
Avainsanat	tietokonenäkö, ohjelmointi, kasvojentunnistus, tietokanta, OpenCV

Tietokonenäkö on nopeasti kasvava tietotekniikan osa-alue, osittain koska laitteet ovat halpoja (kamerat, suoritinten teho) ja osittain siksi, että konenäköalgoritmit alkavat kypsyä. Tämä ala sai alkunsa halusta selvittää, miten tietokoneet käsittelevät kuvia ja kuinka tätä tietoa sovelletaan hyödyllisiä ohjelmia kehitettäessä.

Tämä tutkimus antaa arvokasta tietoa niille, jotka ovat kiinnostuneita konenäöstä ja siitä, miten kasvojentunnistus voidaan toteuttaa käyttämällä OpenCV-kirjastoa. Sovelluksen perusrakenne on määritelty mahdollisimman yksiselitteisesti tässä työssä.

Sovellus on suunniteltu toimimaan Windows-ympäristössä ja se on toteutettu käyttäen C++ ohjelmointikieltä sen tarjoaman oliopohjaisuuden ja suoritusnopeuden vuoksi. Kuvat ja muut toiminnot luodaan avoimen lähdekoodin OpenCV-kirjaston avulla. Kasvojentunnistusjärjestelmän toteuttamista varten oli tarpeen luoda myös tietokanta.

Tuloksena oli ohjelma, joka mahdollistaa reaaliaikaisten kuvien ottamisen ja toteuttaa jatkuvaa kasvojen tunnistamista.

RESUMEN

KYMENLAAKSON AMMATTIKORKEAKOULU
Universidad de Ciéncias Aplicadas de Kymenlaakso
Ingenieria Informática

BALLESTER TOMÁS, FELIPE	Visión Artificial y Reconocimiento facial
Tesis	40 páginas + 10 páginas de apéndices
Supervisor	Teemu Saarelainen, Profesor titular
Comisionado por	Universidad de Ciéncias Aplicadas de Kymenlaakso
Diciembre 2010	
Palabras clave	vision artificial, programación, reconocimiento facial, base de datos, OpenCV

La visión artificial es un campo de rápido crecimiento, en parte porque el hardware es asequible (cámaras, potencia de procesado) y en parte porque los algoritmos de visión artificial están empezando a madurar. Este campo se inició con la motivación de estudiar cómo los ordenadores procesan las imágenes y cómo aplicar este conocimiento para desarrollar programas útiles.

Los objetivos de este estudio eran dar un conocimiento valioso para aquellos que están interesados en la visión artificial e implementar una aplicación de reconocimiento facial utilizando las librerías de OpenCV. La estructura básica de la aplicación se ha definido tan claramente como ha sido posible en este documento.

La aplicación fue diseñada para funcionar en sistemas Windows y se desarrolló con el lenguaje de programación C++ debido a su estructura orientada a objetos y a su alto rendimiento. Las imágenes y otras funciones fueron creadas con las librerias de código abierto OpenCV. Para la implementacion de la aplicación de reconocimiento facial fue necesario crear una base de datos.

La aplicación resultante fue capaz de obtener imágenes a tiempo real y realizar reconocimiento continuo de caras.

ACKNOWLEDGEMENTS

First and foremost, I thank my family, especially my wife Leena, for the support they gave me during my studies. I also extend my thanks to all the people who believed in me: my brothers, colleagues and friends, for their constant encouragement.

I sincerely wish to express my appreciation to all the personnel of the Kymenlaakso University of Applied Sciences, especially to the lecturers Teemu Saarelainen, Paula Posio and Marko Oras, for their important roles in teaching and advising me during my studies. My warm thanks go to the Jinhua University professor Lu Yan Fei for the inspiring discussion we had.

I am also very grateful for the valuable help of all the test subjects who posed for my face database.

And finally, I warmly thank my parents for giving me the opportunity to see this world and for their continuous help.

Kotka, 1.12.2010

Felipe Ballester

TABLE OF CONTENTS

APPENDICES

TERMS AND ABBREVIATIONS

Biometric	Method for uniquely recognizing humans, based upon one or more intrinsic physical or behavioural traits
Bit	Binary digit, basic unit of information in computing
BSD license	Family of permissive free software licenses
C/C++	General purpose Programming language
CPU	Central Processing Unit
DirectX	Collection of application programming interfaces (APIs) for handling tasks related to multimedia
Eigenface	Set of vectors used in computer vision for human face recognition
FRVT	Face Recognition Vendor Tests, provide independent US government evaluations of commercially available and prototype face recognition technologies
GB	Gigabyte, unit of Information used
Haar classifier	Machine learning approach for visual object detection
HMM	Hidden Markov Model
LRPCA	Local Region Principal Component Analysis
MB	Megabyte, unit of Information used
OpenCV	Open Source Computer Vision Library
PCA	Principal Component Analysis
Pixel	A single point in a raster image
RGB	Red/Green/Blue colour model
SDK	Software Development Kit
.NET	Software Framework
2D / 3D	2 / 3 Dimensions

1 INTRODUCTION

The human ability to interact with other people is based on their ability of recognition. This innate ability to effortlessly identify and recognize objects, even if distorted or modified, has induced to research on how the human brain processes these images.

This skill is quite reliable, despite changes due to viewing conditions, emotional expressions, ageing, added artefacts, or even circumstances that permit seeing only a fraction of the face. Furthermore, humans are able to recognize thousands of individuals during their lifetime.

Understanding the human mechanism, in addition to cognitive aspects, would help to build a system for the automatic identification of faces by a machine. However, face recognition is still an area of active research since a completely successful approach or model has not yet been proposed to solve the face recognition problem.

Automated face recognition is a very popular field nowadays. Face recognition can be used in a multitude of commercial and law enforcement applications. For example, a security system could grab an image of a person and the identity of the individual by matching the image with the one stored on the system database.

1.1 Objective

The main purpose of this thesis was to design and to implement a software project in which the learned methods and programming languages would be used. From the beginning, only open source code was used.

The minimum requirements for the application were to be able to detect and to recognize different individuals by using their faces. The method used is the Principal Component Analysis, also known as Eigenfaces, which is considered to be the first successful example of facial recognition technology. This method is not 100% accurate, but is a good start for people who are interested in this field.

This thesis contains details for the research and implementation of frontal-view human face detection and recognition systems.

1.2 Biometric measures

Biometric systems are automated methods for identifying people through physiological or behavioural characteristics.

Face recognition technology is the least intrusive and fastest biometric technology. It works with the most obvious individual identifier: the human face.

A face recognition system would allow a person to be identified by walking in front of a camera instead of requiring them to place their hand on a reader or positioning their eyes in front of a scanner. There is no intrusion or delay, and in most cases the subjects are entirely unaware of the process.

1.3 Computer vision in general

Computer vision is the science and technology of machines that see, and seeing in this case means that the machine is able to extract from an image some information that is necessary for solving some task. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems.

Typical tasks of computer vision are:

- Recognition
- Motion analysis
- Scene reconstruction
- Image restoration

1.4 The difficulty with computer vision

At present, a computing machine is not able to actually understand what it sees. This level of comprehension is still a faraway goal for computers, as the ability to under-

stand an image is not just to collect some pixels. The capability to identify an object perfectly is truly incredible.

Computers only “see” just a grid of numbers from the camera or from a disk (Figure 1), and that is how far it can go. Those parameters have rather a large noise component, so the profitable information is quite small at the end.

Many computer vision problems are difficult to specify, especially because the information is lost in the transformation from the 3D world to a 2D image. Furthermore given a two-dimensional view of a 3D world, there is no unique solution to reconstruct the 3D image.

The noise in computer vision is typically dealt with the use of statistical methods. However, other techniques account for noise or distortions by building explicit models learned directly from the available data (1).

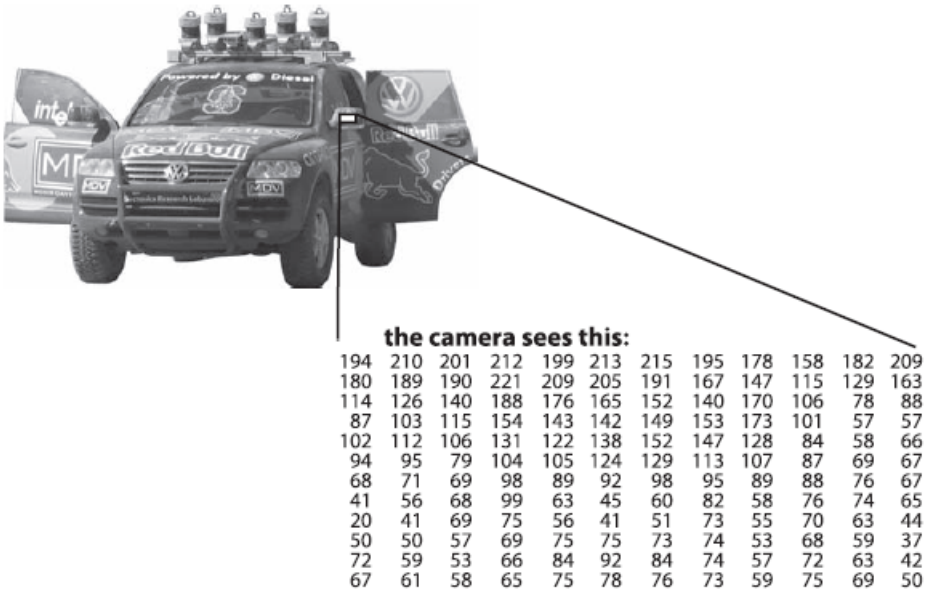

Figure 1. Representation of what a computer sees from a small portion of a picture (1)

The complexity of computer vision is aggravated by the fact we are dealing with a large amount of data. A typical greyscale image has 320x240 pixels, each with 8 bit intensity values, so the size of the whole image is 320x240x8 = 614400 bits (2).

1.5 Earlier studies on this topic

Research on automatic machine recognition of faces basically started in the 1970s. The interest in this kind of technology grew substantially after the increase of security related applications.

There are applications developed for many different platforms, ranging from computers to mobile phones.

Among the different biometric techniques, facial recognition may not be the most reliable and efficient, but recent improvements in the algorithms are currently being evaluated for better effectiveness. High resolution face images, 3D face scans and LRPCA Baseline are examples of new algorithms.

The Face Recognition Grand Challenge, sponsored by the U.S. Government and law enforcement agencies, was made to promote and advance the development of this technology (3).

The research directions (according to FRVT) for the FRGC 2002 were:

- Recognition from outdoor facial images
- Recognition from non-frontal facial images
- Understanding why males are easier to recognize than females
- Greater understanding of the effects of demographic factors on performance
- Development of better statistical methods for understanding performance
- Develop improved models for predicting mass identification performance
- Effect of algorithm and system training on covariate performance
- Integration of morphable models into face recognition performance

The research directions (according to FRVT) for the FRGC 2006 were:

- High resolution still imagery (5 to 6 mega-pixels)
- 3D facial scans
- Multi-sample still facial imagery
- Pre-processing algorithms that compensate for pose and illumination

The Computer Vision Papers Resource (4), collects all kinds of studies on computer vision. This organization offers open source algorithm implementations, datasets, conferences and forums to promote this technology.

2 BASIC CONCEPTS

The purpose of this thesis was to implement a sample application, which could perform simple facial detection and recognition using an affordable webcam.

The application was programmed using C++ language with the help of open source libraries. For the implementation and compilation, the Microsoft Visual Studio Developer Tools were used and the resulting application also used the .NET Framework.

The application was designed to work on Windows operating systems, and as prerequisite, they need to have the Microsoft Platform SDK (or DirectX SDK) installed.

2.1 Hardware

All the development process was carried out with a personal computer and a web camera. The components used were easy to find, but for the image processing it was necessary to have at least a required minimum capacity.

The technical specifications of the used hardware are:

Computer model: Acer TravelMate 5720
CPU: Intel Mobile Core 2 Duo T7300
RAM: 2 GB
Graphics: 256 MB - ATI Mobility Radeon X2500
Camera: Acer Crystal Eye webcam (Integrated).

2.2 OpenCV

The Open Source Computer Vision Library (OpenCV) developed by Intel, contains programming functions mainly aimed at real-time computer vision (5).

OpenCV has been released under a BSD license and it is free for both academic and commercial use. It is used worldwide and its application areas range from interactive art and games to advanced robotics.

The library is written in C/C++ and it has over 500 algorithms, including an object detector. Face detection is done by using a Haar-classifier.

2.3 Related work and previous research

The recognition process is not a simple task. Before comparing faces with the ones in the database, the algorithm needs to select an object inside an image to reduce the area of computation and then treat it for later manipulation.

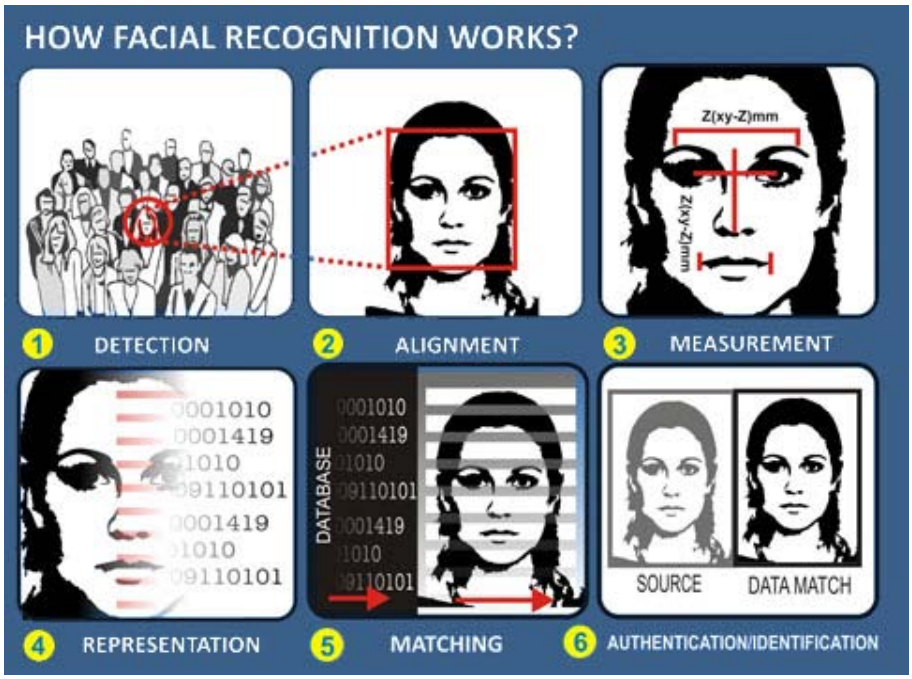

Figure 2. How facial recognition works (6)

This process is resumed in three steps: the first step is to get the input image, the second is to find a face on the image and the third is to let the algorithm to try to recognize who this person would be.

Figure 3. Basic steps of the Face recognition process

2.3.1 Real-time imaging and picture processing

The term real-time imaging refers to real-time image processing and image analysis. Simply put, the camera gets images which are digitized into computer graphics.

Picture processing is any form of signal processing for which the input is an image, such as a photograph or a video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it (7).

For this process the OpenCV library is used, which is able to query and manipulate images straight from a camera connected to the computer.

2.3.2 Face detection

Face detection is the process that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features and ignores anything else. This is essentially a segmentation problem and in practical systems, most of the effort goes into solving this task.

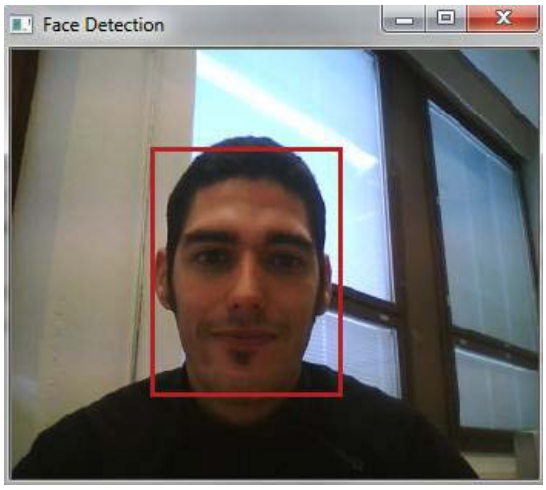

Figure 4. Real-time face detection

Real-time face detection involves detection of a face from a series of frames obtained from a video-capturing device. While the hardware requirements for such a system are far more stringent, from a computer vision stand point, real-time face detection is ac-

tually a far simpler process than detecting a face in a static image. This is because unlike most of our surrounding environment, people are continually moving.

2.3.3 Face recognition

Face recognition is the process to identify or verify people from a digital image or a video frame from a video source. One of the ways to do this is to compare facial features selected from the image and from a facial database.

According to the research by Brunelli and Poggio (8), all approaches to human face recognition can be divided into two strategies:

1. Geometrical features

   This technique involves computation of a set of geometrical features such as nose width and length, mouth position and chin shape, etc. from the picture of the face we want to recognize. This set of features is then matched with the features of known individuals. A suitable metric such as Euclidean distance (finding the closest vector) can be used to find the closest match. Most pioneering work in face recognition was done using geometrical features (9).

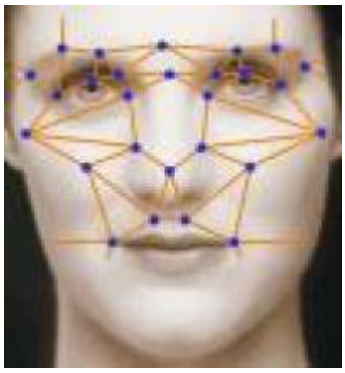

Figure 5. Geometrical features

2. Template matching

   The basis of the template matching strategy is to extract whole facial regions (matrix of pixels) and compare these with the stored images of known individuals. Once again Euclidean distance can be used to find the closest match. However, there are far more sophisticated methods of template matching for face recognition. These involve extensive pre-processing and transformation of the extracted

grey-level intensity values. For example, the Principal Component Analysis, sometimes known as the Eigenfaces approach, uses template matching to pre-process the gray-levels (9).

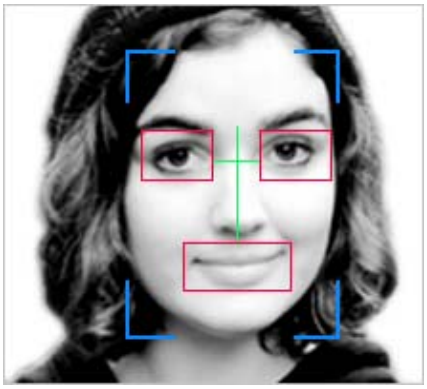

Figure 6. Template matching

2.4 Summary

To summarize this chapter, several things were needed for the sample application, which was implemented in this study, namely:

1. Image acquisition (via webcam)
2. Image pre-processing
3. Face detection
4. Face recognition (against a database of known faces)

The main steps for the implementation were provided by the OpenCV, while the face-database and other methods had to be performed for the final application. Also quite extensive programming effort was needed in the user-interface, data processing and integration of the OpenCV-library with the application.

The relevant information of an image was extracted and encoded as efficiently as possible. Then the similarities were compared with the face models of the database (2).

3 ALGORITHMS

For the implementation of the Facial Recognition, the Eigenface approach was proposed as a method. This approach, considered to be the first successful example in the field, is based on the PCA of the images. PCA involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components.

3.1 Principal Component Analysis applied to computer vision

Principal Component Analysis (also known as Karhunen-Loève transform) provides a suitable strategy for face recognition because it identifies variability between human faces, which may not be immediately obvious (10).

"Mathematically, PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA is theoretically the optimum transform for given data in least square terms.

For a data matrix, X^T, with zero empirical mean (the empirical mean of the distribution has been subtracted from the data set), where each row represents a different repetition of the experiment, and each column gives the results from a particular probe, the PCA transformation is given by:

$Y^T = X^T W = \Sigma^T V$

Where the matrix Σ is an m-by-n diagonal matrix with non-negative real numbers on the diagonal and $W \Sigma V^T$ is the singular value decomposition of X.

Given a set of points in Euclidean space, the first principal component (the eigenvector with the largest Eigenvalue) corresponds to a line that passes through the mean and minimizes the sum of squared errors with those points. The second principal component corresponds to the same concept after all correlation with the first principal component has been subtracted out from the points. Each Eigenvalue indicates the portion of the variance that is correlated with each eigenvector. Thus, the sum of all the Eigenvalues is equal to the sum of the squared distances of the points with their mean divided by the number of dimensions. PCA essentially rotates the set of points

around their mean in order to align with the first few principal components. This moves as much of the variance as possible (using a linear transformation) into the first few dimensions. The values in the remaining dimensions, therefore, tend to be highly correlated and may be dropped with minimal loss of information. PCA is often used in this manner for dimensionality reduction. PCA has the distinction of being the optimal linear transformation for keeping the subspace that has largest variance. This advantage, however, comes at the price of greater computational requirement if compared, for example, to the discrete cosine transform. Nonlinear dimensionality reduction techniques tend to be more computationally demanding than PCA" (11).

A set of human faces is analyzed using PCA to determine which factors account for the variance of faces. In face recognition, these variables are called Eigenfaces because when plotted, they resemble human faces.

Although PCA is used extensively in statistical analysis, the pattern recognition community started to use PCA for classification only relatively recently. PCA's greatest strengths are in its ability for data reduction and interpretation. For example a 100x100 pixel area containing a face can be very accurately represented by just 40 Eigenvalues, that means the image manipulation and recognition will be less demanding. Each Eigenvalue describes the magnitude of each Eigenface in each image.

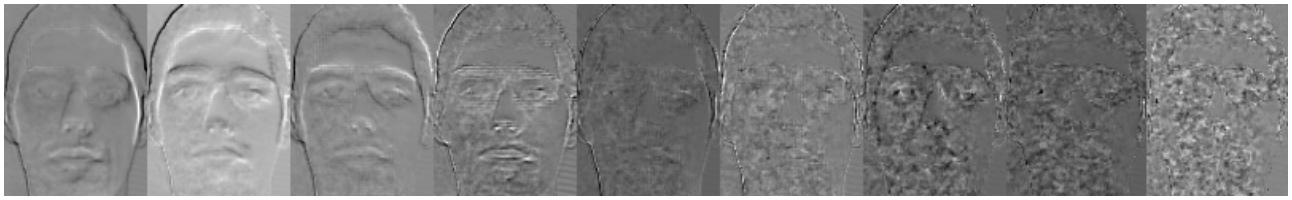

Figure 7. Sample of a simple Eigenface.

A test image is recognized by computing its distance (in Euclidean terms, for example) from the templates generated from the images in the training set and selecting the closest match. However, the array and the template are not the original face image but rather their projection onto an optimal coordinate system. These sets of vectors are the eigenvectors of the covariance matrix of the ensemble of training faces.

3.1.1 Calculating Eigenfaces

Eigenfaces are a set of eigenvectors used in the computer vision for face recognition. These eigenvectors are derived from the covariance matrix of the probability distribution of the high-dimensional vector space of possible faces of human beings (12).

To create a set of Eigenface, it is necessary to follow the next steps:

1. Prepare a training set of face images.

   The pictures constituting the training set should have been taken under the same lighting conditions, and must be normalized to have the eyes and mouths aligned across all images. They must also be all re-sampled to the same pixel resolution. Each image is treated as one vector, simply by concatenating the rows of pixels in the original image, resulting in a single row with r × c elements. For this implementation, it is assumed that all images of the training set are stored in a single matrix T, where each row of the matrix is an image.

2. Substrate the mean.

   The average image ‘a’ has to be calculated and then subtracted from each original image in T.

   

   Figure 8. Eigenfaces of the average image (the mean).

3. Calculate the eigenvectors and Eigenvalues from the covariance matrix S.

   Each eigenvector has the same dimensionality (number of components) as the original images, and thus can itself be seen as an image. The eigenvectors of this covariance matrix are therefore called Eigenfaces. They are the directions in which the images differ from the mean image. Usually this will be a computationally expensive step (if at all possible), but the practical applicability of Eigenfaces stems from the possibility to compute the eigenvectors of S efficiently, without ever computing S explicitly.

4. Choose the principal components and form a feature vector.

   The D x D covariance matrix will result in D eigenvectors, each representing a direction in the r × c-dimensional image space. The eigenvectors with largest associated Eigenvalue are kept.

These Eigenfaces can now be used to represent both existing and new faces: we can project a new (mean-subtracted) image on the Eigenfaces and thereby record how that new face differs from the mean face. The Eigenvalues associated with each Eigenface represent how much the images in the training set vary from the mean image in that direction. We lose information by projecting the image on a subset of the eigenvectors, but we minimize this loss by keeping those Eigenfaces with the largest Eigenvalues.

For instance, if we are working with a 90 x 120 image, then we will obtain 10,800 eigenvectors. In practical applications, most faces can typically be identified using a projection on between 100 and 150 Eigenfaces, so that most of the total of Eigenvectors can be discarded.

3.1.2 Using Eigenfaces

For recognition, face images are encoded by projecting them onto the axes spanning the feature space.

An unknown test image T is projected T', its image in the feature space, as follows:

$T' = U (T - m)$

Where U are the Eigenfaces used and m is the mean.

In order to determine which face best matches the test image, the Euclidean distance between each training image and the projected test image is used. Another option to get better recognition accuracy is the possibility to use the Mahalanobis distance.

3.2 Other algorithms

In order to gain an insight into how face recognition would be implemented, different algorithms were analyzed and studied.

3.2.1 Fisherfaces

Fisherfaces is a PCA method which is competing with the Eigenface technique. This method for facial recognition is less sensitive to variation in lighting and pose of the face than the method using Eigenfaces (13).

3.2.2 Hidden Markov Model

"Hidden Markov Models (HMM) are a set of statistical models used to characterize the statistical properties of a signal. HMM consists of two interrelated processes: an underlying, unobservable Markov chain with a finite number of states, a state transition probability matrix and an initial state probability distribution a set of probability density functions associated with each state" (14).

HMMs have been used mostly in speech recognition applications, but in the last decades, researchers have started to use it in computer vision and image recognition.

3.2.3 Other approaches

Newer techniques were not studied for this thesis, because of their difficult implementation and lack of documentation. However, they are listed here, with older algorithms (4) (14) (15):

- 3-D Face Recognition
- 3-D Morphable Model
- Skin Texture Analysis
- Dynamic Link Matching
- Linear Discriminate Analysis
- Independent Component Analysis
- Elastic Bunch Graph Matching
- Active Appearance Model
- Bayesian Framework
- Support Vector Machine

4 METHODS

The methods used to develop the program application, which made this research possible, are mainly found in the OpenCV libraries. These libraries can greatly simplify computer-vision programming.

4.1 Image capturing

Image capturing is the process of obtaining and treating images from a digital device or a file. OpenCV has a built-in property to get images straight from a camera device, a video file or a single picture. The images can be easily modified and escalated in order to obtain a steady processing.

4.2 Face detection

"OpenCV's face detector uses a method called the Viola-Jones method (16) published in 2001, also known as the Haar cascade classifier. This approach for detecting objects in images combines four key concepts:

- *Simple rectangular features, called Haar features*
- *An Integral Image for rapid feature detection*
- *The AdaBoost machine-learning method*
- *A cascaded classifier to combine many features efficiently*

The features that Viola and Jones used are based on Haar wavelets. These classifiers are single square waves (one high interval and one low interval). In two dimensions, a square wave is a pair of adjacent rectangles - one light and one dark.

Figure 9. Different Haar classifiers used in OpenCV (17)

The bottom figures represented in Figure 9 are not true Haar wavelets. These features are rectangle combinations used for visual object detection and they are better suited to visual recognition tasks.

Figure 10. Rectangular features over a face

The presence of a Haar feature is determined by subtracting the average dark-region pixel value from the average light-region pixel value. If the difference is above a threshold (set during learning), that feature is said to be present.

To determine the presence or absence of hundreds of Haar features at every image location and at several scales efficiently, Viola and Jones used a technique called an Integral Image. In this case, the integral value for each pixel is the sum of all the pixels above it and to its left. Starting at the top left and traversing to the right and down, the entire image can be integrated with a few integer operations per pixel.

To select the specific Haar features for use, and to set threshold levels, Viola and Jones used a machine-learning method called AdaBoost. AdaBoost combines many "weak" classifiers to create one "strong" classifier. This combined series of classifiers as a filter chain, shown in Figure 11, is especially efficient for classifying image regions. Each filter is a separate AdaBoost classifier with a fairly small number of weak classifiers.

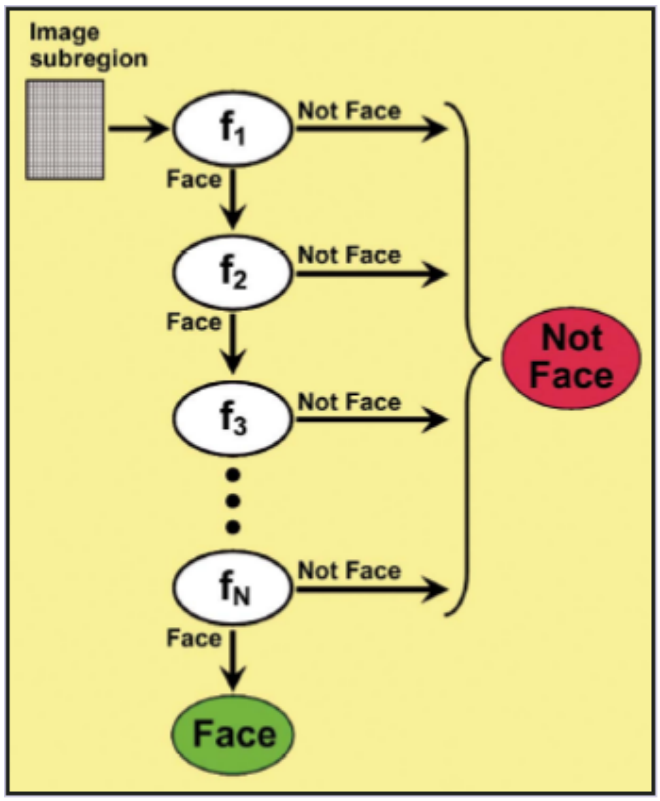

Figure 11. Classifier cascade as a chain of filters (17)

The filters at each level are trained to classify training images that have passed all previous stages. During use, if any one of these filters fails to pass an image region, that region is immediately classified as "Not Face". When a filter passes an image region, it goes to the next filter in the chain. Image regions that pass through all filters in the chain are classified as "Face".

The order of filters in the cascade is based on the importance weighting that AdaBoost assigns. The more heavily weighted filters come first, to eliminate non-face image regions as quickly as possible" (17).

The classifier uses data stored in an XML file to decide how to classify each image location. The classifier used is called haarcascade_frontalface_default.xml.

4.3 Face recognition

Before trying to recognize, the face image obtained has to be pre-processed. This is because the probability to get a correct result on a non-treated image is less than 10%.

It is very important to apply various techniques to standardize the images. Most face recognition algorithms are very sensible to lighting conditions. There are also other is-

sues such as hair style, makeup, rotation angle, size and emotion that can affect the recognizing process (18).

The method of recognition used in this study is the Eigenface, and this method works with greyscale images. However, the most important steps to take on pre-processing are:

- Face image scale, crop and resize. These are necessary for reducing the unnecessary pixels and for cutting the image onto a standard size. It is important to maintain the same aspect ratio.
- Image transformation to greyscale. This converts the image from RGB to greyscale.
- Histogram equalization. This method is used for automatically standardizing the brightness and contrast of the facial images.

Figure 12. Processing the face image

Processing the face image is the next step to take after the image has been treated. It performs the Eigenface on the image. OpenCV comes with a function which performs the PCA operation; however it needs a database (training set) of images for it to know how to recognize each of the people.

The PCA converts all the training images into a set of Eigenfaces that represent the differences between the training images and the average face image.

To explain Eigenfaces in simple terms, Eigenfaces figures out the main differences between all the training images, and then represents each training image using a combination of those differences.

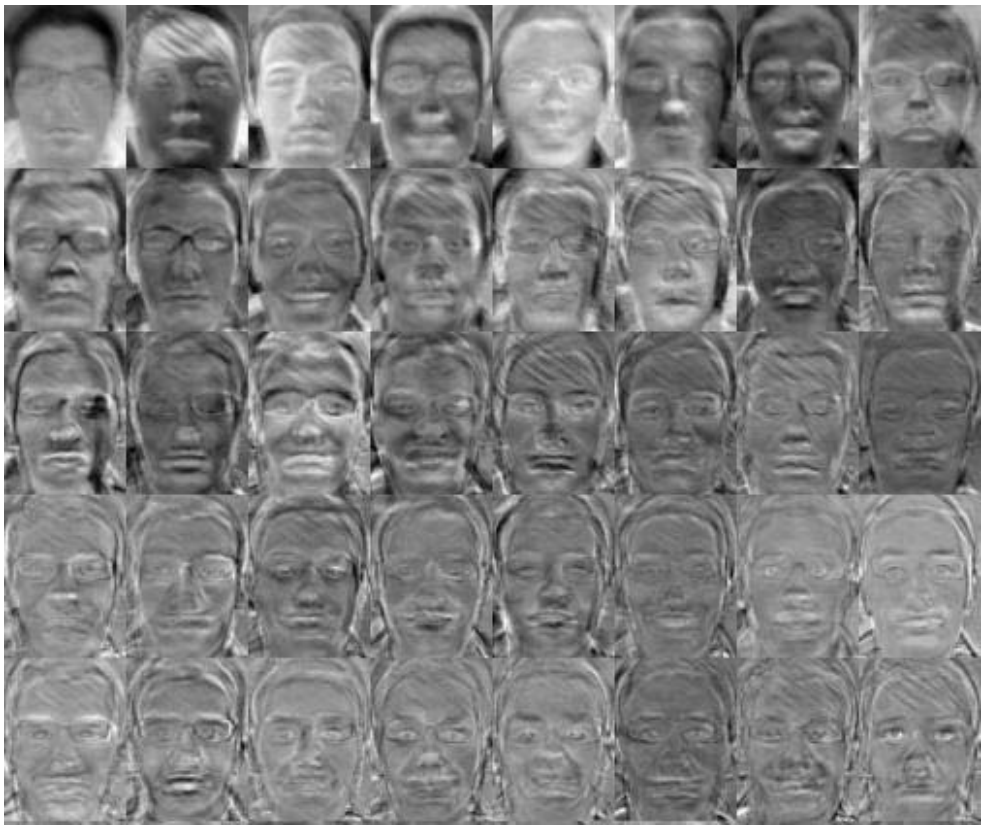

Figure 13. Different Eigenfaces from the database

So for example, one of the training images might be made up of:

(average Face) + (13.5% of eigenface0) - (34.3% of eigenface1) + (4.7% of eigenface2) + ... + (0.01% of eigenface299).

It is possible to regenerate the training image back from the Eigenfaces by multiplying the ratios with these images. This means that the images have been compressed into a smaller file for processing. The only issue in this process is the collection of noise in part of the images (19).

Once the program knows which training image is most similar to the input image, and assuming the confidence value is not too low (it should be at least 0.6 or higher), it has figured out who that person is, in other words, the person has been recognized.

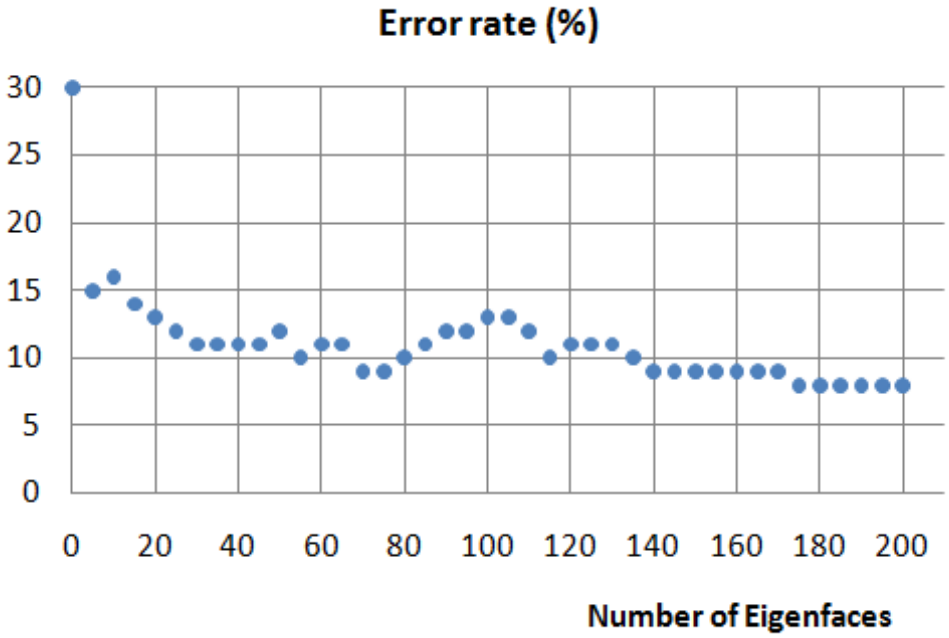

Figure 14. Eigenfaces error rate

4.4 Face Image database

A database was created with the images of the testers. The database consists of 300 images, 10 each of 30 different subjects. The subjects were students and faculty of Kymenlaakso Ammattikorkeakoulu University of Applied Sciences. The ages of the subjects ranged from 20 to 40, and they are all males. The subjects were asked to face the camera and no restrictions were imposed on the expression. Only limited side movement was tolerated. Some of the subjects were wearing glasses and others had a beard.

The images were automatically treated, escalated and cropped to the same resolution 90x120, 8 bit grey levels.

The images were classified in different folders, with the name of the subject. A database file was created to distinguish and organize the data of every image.

5 IMPLEMENTATION

The application was implemented on C++ language and .NET. The objective was to create an easy-to-control and user-friendly interface for our application. The application had different functions, including, naturally, abilities to read, write and play image or video files. However the most important functions for this project were the features that enabled the grabbing of images from a camera (See appendix 1), the detection of the faces and the actual recognition of a person from a face (See appendix 2).

5.1 Main Menu

The initial state of the program shows the main functions of the program.

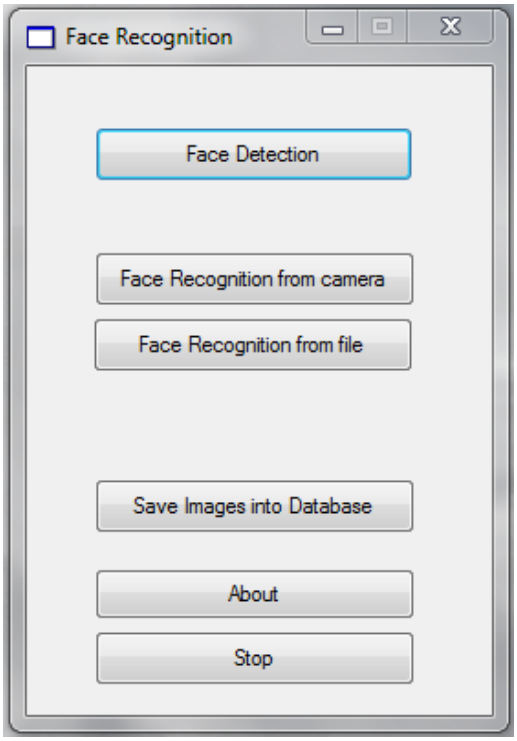

Figure 15. Main Menu

The menu contains buttons for the Face detection, the Face Recognition from the camera, the Face Recognition from the file, the Database builder, the About Information and the button for closing the application.

5.2 Face Detection

The first option on the main menu is the Face Detection. This part of the application is able to find faces in real-time images from a camera. The detection of faces is done automatically. When the detection occurred, a red rectangle appeared on the windows surrounding the face. At the same time a submenu was opened on the left side of the user interface. This submenu has different options to change the state of the video source (See Appendix 3).

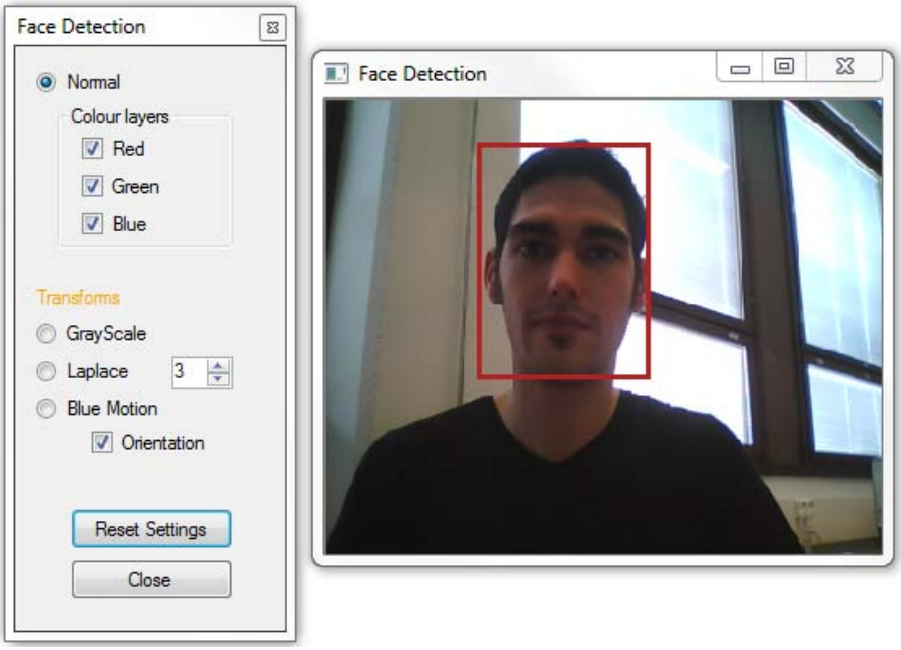

Figure 16. Face Detection

The detection can be made thanks to the Haar Cascade Classifier, which is not affected even if the image is simple-channelled or a combination of different colour channels.

This part of the application also offered the possibility to check other imaging tools, for example to apply the Laplace transform or to recognize movement and orientation on the images.

5.3 Face Recognition

The next two options of the main menu are dedicated to the Face Recognition. The options are to recognize from the camera or from a file. Both options worked under the same function, the only difference being the source of the images.

5.3.1 Recognition from camera

The Face Recognition from camera option was able to detect and identify faces, previously saved on the database, from a real time imaging source. The faces were identified and then the name of the person was visualized on the screen and on the Submenu. The Submenu also shows the image of the recognized person which is saved on the database.

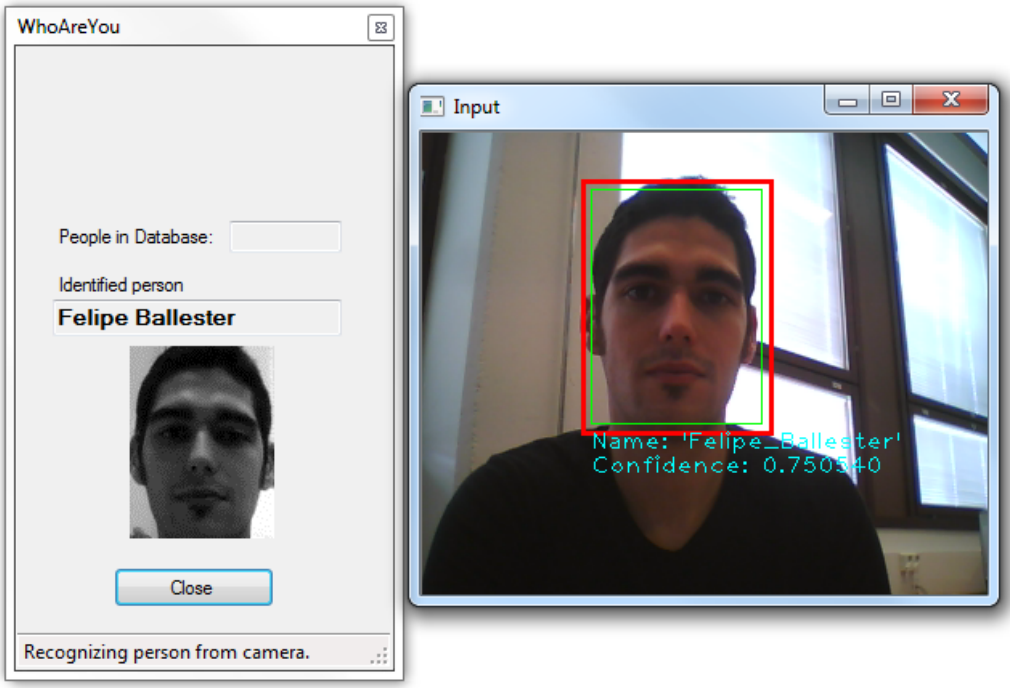

Figure 17. Face Recognition from camera

5.3.2 Recognition from file

The application had also an option to identify a subject from an image or a video file.

This part works exactly as the previously explained. The only difference was that the images are from a file.

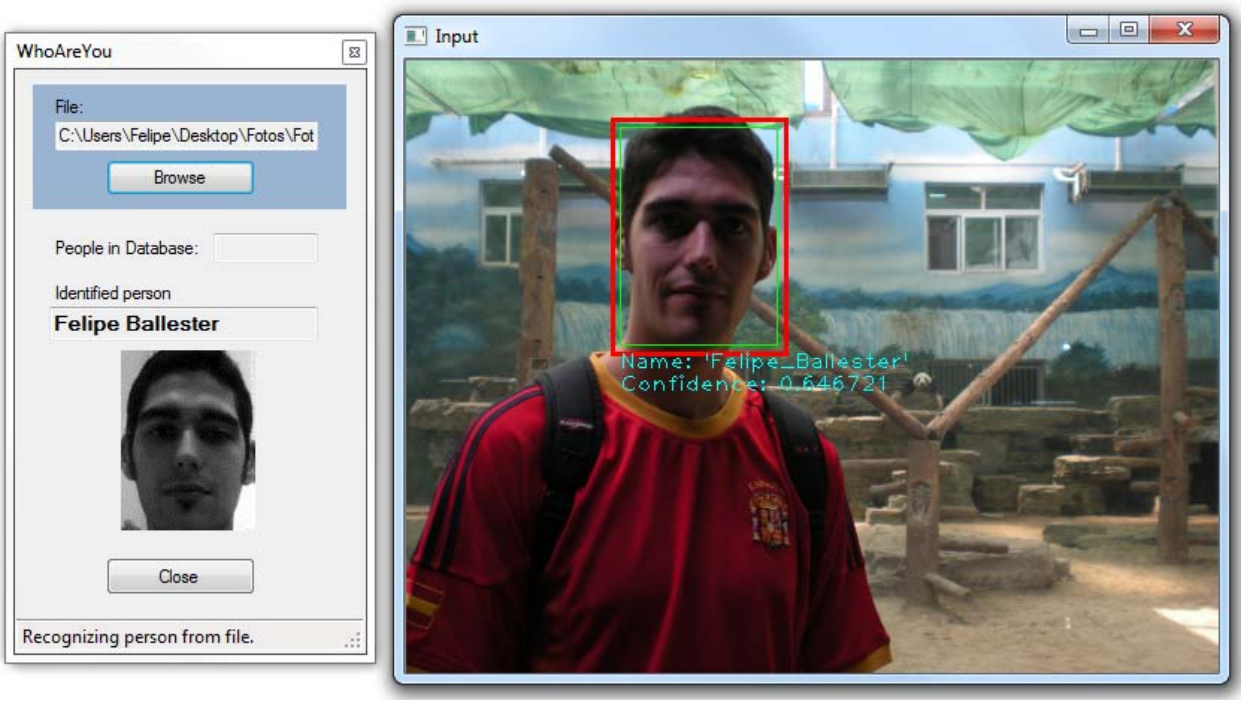

Figure 18. Face Recognition from file

5.4 Database builder

The Face Recognition works with a database of previously saved images. This part of the program was able to create a database, where the faces of different subjects were identified, in this case by giving the complete name.

The application took 10 images from the person, and then trained the Eigenfaces for later use in the recognition process.

The architecture of the database built was very simple; it was divided into three different columns: a correlative identity number, the name of the subject and the path of the ten files of each face.

This part also has a panel where it shows a list of all the people on the database.

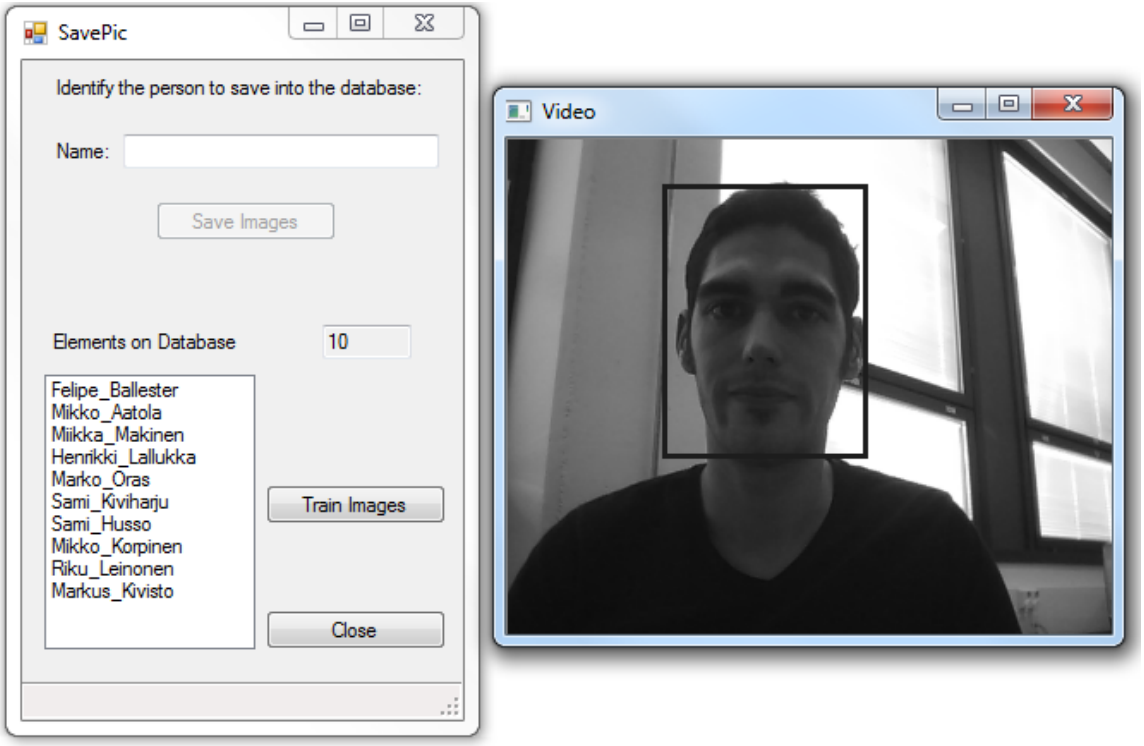

Figure 19. Database builder

5.5 About

The About screen displays information about the application and the author of the program. This window is only informative; the only useful code attached is in the email link.

Figure 20. About Screen

6 TESTING

The application was tested during the research. It was important to have the same conditions with the testers to prove how the program responded and to get conclusive results. Once their faces were added into the database the testers became database subjects.

6.1 Test situation

The test subjects were sitting on a chair at around one meter from the camera, which was always positioned at the same angle.

All the tests were done in a well illuminated room. The lighting was regular and white, mainly from the ceiling and from the front.

At the moment of saving the images into the database, it was important that the subjects stayed almost motionless, except for the head that had to be slightly turned to the side. The application detected the person and saved the images into the database.

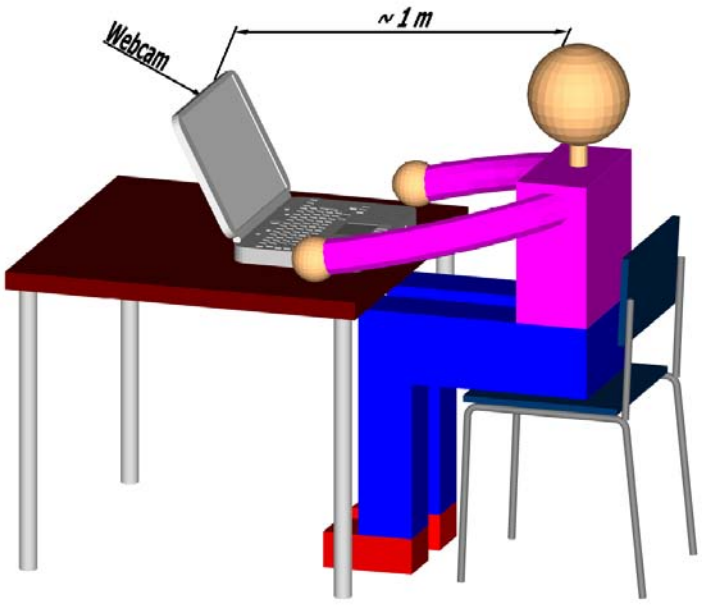

Figure 21. Representation of a test situation

6.2 Test Procedure

The testers were able to freely interact with the computer and inspect the different parts of the program. The tests were always conducted under the same conditions.

The name of the subject was asked to be written onto the appropriate name box, in order of adding the subject face into the database.

In case of problems with the interface, the test was repeated from the beginning.

6.3 Test Results

The results of the tests were favourable; around 85% of the tests were satisfactory. Some tests failed because of implementing problems. These tests were repeated, but the results of the investigation were not affected.

The rates represented in Figure 22 explain how, under the initial recognition, just after saving the face subject into the database, the success rate was around 90%. These rates dropped drastically by around 20% when trying to recognize the same subject again, after adding a new subject to the database.

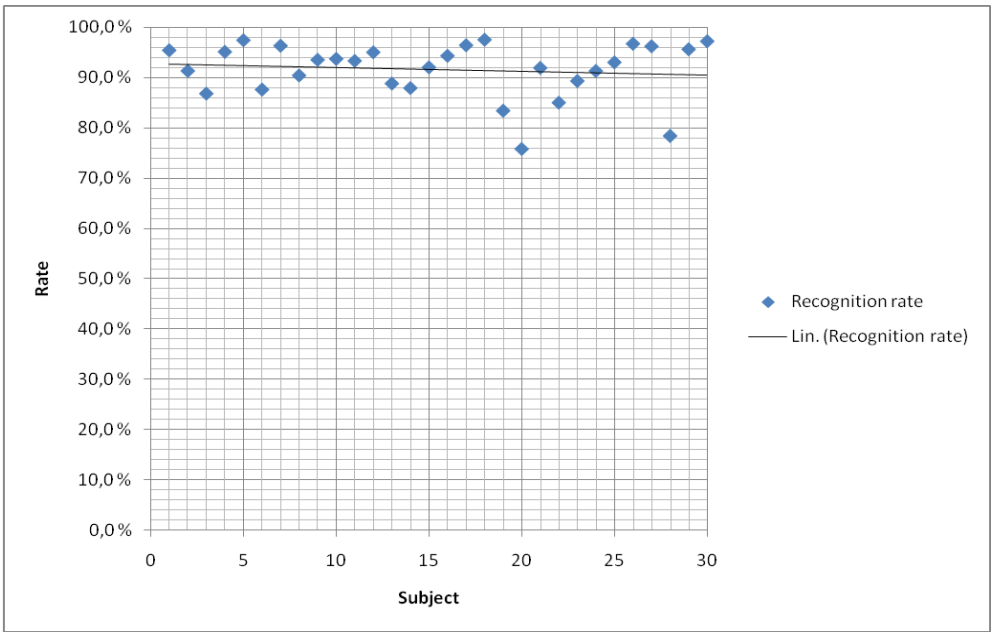

Figure 22. Recognition rate on initial recognition

7 CONCLUSIONS

The Face Detection System functioned with almost perfect accuracy (99 %), but many of the non-faces were erroneously “detected” as faces. This was acceptable because the false positive rate was only of 0.0001%. The application was able to detect, without problems, a number of faces at the same time from any source.

The Face Recognition System was not robust enough to achieve high recognition accuracy. The main reasons for this were the variance of noise in the images due to the illumination and the limited number of Eigenfaces that were used for the PCA transform. However, the results of the test were favourable, giving rates of over 90% in some cases. Thus, the Eigenface approach satisfied the requirements of this project with its speed, simplicity and learning capability for the resolution of this project.

The face database was created without problems, so there was not needed to be conducted further work in this area.

The main problems found in the implementation of the application were originated from memory leaks, caused by some images being incorrectly unloaded after use. Another problem was the noise on the images that worsened the accuracy of the recognition

There are many factors that can improve the performance of the program. Good preprocessing of the images is very important for getting adequate results. To improve the face recognition accuracy, it would be necessary to have more input images. This could be achieved by taking at least 50 photos of each person, particularly from different angles and in various lighting conditions, or by obtaining more training images, or by generating new images from the existing ones (mirroring, resizing, rotating or adding noise).

It is important to have a lot of variation of conditions for each person so that the classifier will be able to recognize the person in different lighting conditions and positions instead of looking for specific conditions. It is also important, however, to make sure that a set of images for a person is not too varied. This would make the classifier too generic and also yield very poor results.

The automated vision systems implemented in this thesis did not even approach the performance of a human’s innate face recognition system, nor were as robust as other

commercial applications. However, they gave an insight into what the future may hold for computer vision.

In conclusion, a robust face recognition system should be insensitive to:

• Changes in illumination

• Changes in head orientation and scale

• Presence of facial details such as glasses, beards or other artefacts

• Background

• Noise

However, the application created was quite effective to detect and recognize faces in controlled environments.

REFERENCES

1. Bradski, G. and Kaehler, A. 2008, Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol: O'Reilly.

2. Turk, M. and Pentland, A. 1991, Eigenfaces for recognition. Journal of Cognitive Neuroscience. PDF Available: www.face-rec.org/algorithms/PCA/jcn.pdf [cited 20.9.2010].

3. Face Recognition Vendor Tests. Available: http://www.frvt.org/ [cited 18.9.2010].

4. CVPapers - Computer Vision Resource. Available: http://www.cvpapers.com/ [cited 30.9.2010].

5. OpenCV Homepage. Available: http://opencv.willowgarage.com/wiki [cited 9.6.2010].

6. Inttelix Software. Available: http://www.inttelix.com/face-recognition-technology.php [cited 20.9.2010].

7. Image processing. Wikipedia article. Available: http://en.wikipedia.org/wiki/Image_processing [cited 21.9.2010].

8. Brunelli, R. and Poggio, T. 1993, Face Recognition: Features versus Templates. IEEE Transactions on Pattern Analysis and Machine Intelligence. Washington: IEEE Computer Society.

9. Sumitha, L. 2000. Frontal View Human Face Detection and Recognition. University of Colombo. Sri Lanka. Thesis.

10. Smith, L. 2002, A tutorial on Principal Components Analysis. PDF Available: http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf [cited 29.9.2010]

11. Principal Component Analysis. Wikipedia article. Available: http://en.wikipedia.org/wiki/Principal_component_analysis [cited 29.9.2010].

12. Eigenface. Wikipedia article. Available: http://en.wikipedia.org/wiki/Eigenface [cited 21.9.2010].

13. Belhumeur, P; Hespanha, J. and Kriegman, D. 1997, Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. IEEE Transactions on Pattern Analysis and Machine Intelligence. Washington: IEEE Computer Society.

14. Face recognition Homepage. Available: http://www.face-rec.org [cited 15.6.2010].

15. Delac, K.; Grgic, M. and Stewart, M. 2008, Recent Advances in Face Recognition. Croatia: In-Teh.

16. Viola, P. and Jones, M.J. 2004. Robust Real-time Face detection. International Journal of Computer Vision 57. PDF Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.4879&rep=rep1&type=pdf [cited 21.9.2010].

17. Cognotics - OpenCV Resources. Available: http://www.cognotics.com/opencv/ [cited 18.6.2010].

18. Face recognition project. Available: http://www-users.cs.york.ac.uk/~nep/research/3Dface/tomh/ [cited 10.8.2010].

19. Introduction to Face Detection and Face Recognition. Available: http://www.shervinemami.co.cc/faceRecognition.html [cited 17.6.2010].

```
IplImage*  FaceRecognized::getCameraFrame(void){

              // Grab the next camera frame. Waits until the next frame is ready, and
              // provides direct access to it, so do NOT modify or free the returned image!
              // Will automatically initialize the camera on the first frame.

              IplImage *frame;

              // If the camera hasn't been initialized, then open it.
              if (!capture && !cam) {

              capture = cvCaptureFromCAM( 0 );
              if (!capture) {
                System::Windows::Forms::MessageBox::Show("ERROR in getCameraFrame():
                Couldn't access the camera.", "Error Message");
                exit(1);
                           }
              // Try to set the camera resolution
              cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH, 480 );
              cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 360 );
              // Wait a little, so that the camera can auto-adjust itself
              Sleep(50);   // (in milliseconds)

              }

              frame = cvQueryFrame( capture );

              if (!frame) {
                 return NULL;
              }

              if(flip_1) cvFlip(frame,NULL,+1);   // Inverting mirror effect.

              return frame;


}
```

```
void FaceRecognized::recognizeFromCam(void)
{
		CvMat * trainPersonNumMat;  // the person numbers during training
		float * projectedTestFace;
		double timeFaceRecognizeStart;
		double tallyFaceRecognizeTime;
		CvHaarClassifierCascade* faceCascade;

		BOOL saveNextFaces = FALSE;

		int newPersonFaces;

		trainPersonNumMat = 0;  // the person numbers during training
		projectedTestFace = 0;
		saveNextFaces = FALSE;
		newPersonFaces = 0;


		FaceRecogn::WhoAreYou::TheInstance->set_Status("Recognizing person
		from camera.");

		// Load the previously saved training data
		if( loadTrainingData( &trainPersonNumMat ) ) {
				faceWidth = pAvgTrainImg->width;
				faceHeight = pAvgTrainImg->height;
		}

		// Project the test images onto the PCA subspace
		projectedTestFace = (float *)cvAlloc(  nEigens*sizeof(float) );

		// Create a GUI window for the user to see the camera image.
		cvNamedWindow("Input", CV_WINDOW_AUTOSIZE);

		// Load the HaarCascade classifier for face detection.
		faceCascade = (CvHaarClassifierCascade*)cvLoad(faceCascadeFilename, 0, 0, 0 );
		if( !faceCascade ) {
		  System::Windows::Forms::MessageBox::Show("ERROR in
		  recognizeFromCam(): Could not load Haar cascade Face detection classifier in
		  faceCascade", "Error Message");
		  exit(1);
		}

		timeFaceRecognizeStart = (double)cvGetTickCount(); // Record the timing.

		while (1)
		{
				int iNearest, nearest;//, truth;
				IplImage *camImg;
				IplImage *processedFaceImg;
				CvRect faceRect;
				IplImage *shownImg;
				float confidence;
				char path_name[256]="";

				// Get the camera frame
				camImg = getCameraFrame();
				if(!camImg ) break;


				// Images
				Image *greyImg=new Image(camImg);
				Image *faceImg =new Image(camImg);
				Image *sizedImg =new Image(camImg);
				Image *equalizedImg=new Image(camImg);
```

```
                // Perform cleaning
                FaceRecogn::WhoAreYou::TheInstance->set_Picture("");
                FaceRecogn::WhoAreYou::TheInstance->set_TextBox("Unknown person");

                // Perform face detection on the input image, using the given Haar
                cascade classifier.
                Rectang rec2 = greyImg->findFace(faceCascade);

                // Make sure the image is greyscale, since the Eigenfaces is only
                done on greyscale image.
                greyImg= greyImg->toGrayscale();

                // Make sure a valid face was detected.
                if(rec2.getWidth()>0  && rec2.top > 0 ){

                  faceRect.y= rec2.top+ 5;
                  faceRect.x= rec2.1eft+5;
                  faceRect.width= rec2.getWidth()-10;
                  faceRect.height= rec2.getHeight()-10;

                // Get the detected face image.
                  faceImg=faceImg->cropImage(greyImg, faceRect);

                // Make sure the image greyImg the same dimensions as the training
                images.
                  sizedImg=sizedImg->resizeImage(faceImg, faceWidth, faceHeight,
                  true);
                // Give the image a standard brightness and contrast, in case it
                was too dark or low contrast.
                  equalizedImg = sizedImg-> equalize();

                  processedFaceImg = equalizedImg->getImage();

                if (!processedFaceImg) {
                  FaceRecogn::WhoAreYou::TheInstance->set_Status("ERROR in
                            recognizeFromCam(): Don't have input image!");
                            continue;
                }

// If the face rec database has been loaded, then try to recognize the person
currently detected.
if ( nEigens > 0) {
                // project the test image onto the PCA subspace
                cvEigenDecomposite(processedFaceImg, nEigens, eigenVectArr, 0, 0,
                   pAvgTrainImg, projectedTestFace);

                // Check which person it is most likely to be.
                iNearest = findNearestNeighbor(projectedTestFace, &confidence);
                nearest  = trainPersonNumMat->data.i[iNearest];

                printf("Most likely person in camera: '%s' (confidence=%f.\n",
                personNames[nearest-1].c_str(), confidence);

FaceRecogn::WhoAreYou::TheInstance->set_TextBox(personNames[nearest-
1].c_str());


sprintf(path_name,"data/%s/%s.jpg",personNames[nearest-1].c_str(), person-
Names[nearest-1].c_str());

FaceRecogn::WhoAreYou::TheInstance->set_Picture(path_name);
                            }

//endif nEigens
```

```
            // Free the resources used for this frame.
            delete greyImg, faceImg, sizedImg, equalizedImg;
            // Copy  the image to more processing.
            shownImg = cvCloneImage(camImg);


            if (faceRect.width > 0) {            // Check if a face was detected.
                        // Show the detected face region.
            cvRectangle(shownImg, cvPoint(faceRect.x, faceRect.y), cvPoint(faceRect.x +
             faceRect.width-1, faceRect.y + faceRect.height-1), CV_RGB(0,255,0), 1, 8, 0);

            if ( nEigens > 0) {
            // Check if the face recognition database is loaded and a person was
            recognized.
            // Show the name of the recognized person, overlayed on the image below
            their face.

            CvFont font;
            cvInitFont(&font,CV_FONT_HERSHEY_PLAIN, 1.0, 1.0, 0,1,CV_AA);
            CvScalar textColor = CV_RGB(0,255,255);           // light blue text
            char text[256];
            sprintf_s(text, sizeof(text)-1, "Name: '%s'", personNames[nearest-1].c_str());
            cvPutText(shownImg, text, cvPoint(faceRect.x, faceRect.y + faceRect.height
            + 15), &font, textColor);

            sprintf_s(text, sizeof(text)-1, "Confidence: %f", confidence);

            cvPutText(shownImg, text, cvPoint(faceRect.x, faceRect.y + faceRect.height
             + 30), &font, textColor);
                                    }
                        }
                        // Display the image.
                        cvShowImage("Input", shownImg);

                        // Give some time for OpenCV to draw the GUI and check if the user
                        has pressed something in the GUI window.
                         cvWaitKey(10);
                         cvReleaseImage( &shownImg );

                        }
                        else{

                        cvShowImage("Input", camImg);
                        cvWaitKey(10);

                        }

            }

            tallyFaceRecognizeTime = (double)cvGetTickCount() - timeFaceRecognizeStart;

            // Free the camera and memory resources used.
            cvReleaseCapture( &capture );
            cvReleaseHaarClassifierCascade( &faceCascade );

}
```

```
#pragma once
#include "FaceRecogn.h"
#include "blueMotion.h"

namespace FaceRecogn {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Summary for Menu_1
	/// </summary>
	public ref class Menu_1 : public System::Windows::Forms::Form
	{
	public:
		Menu_1(void)
		{
			InitializeComponent();
			//
			//TODO: Add the constructor code here
			//
		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~Menu_1()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^  Reset_btn;
	protected:

	protected:
	private: System::Windows::Forms::Button^  button2;
	private: System::Windows::Forms::RadioButton^  radioButton1;
	private: System::Windows::Forms::RadioButton^  radioButton2;
	private: System::Windows::Forms::RadioButton^  radioButton3;
	private: System::Windows::Forms::CheckBox^  checkBox1;
	private: System::Windows::Forms::CheckBox^  checkBox2;
	private: System::Windows::Forms::CheckBox^  checkBox3;
	private: System::Windows::Forms::Label^  label1;
	private: System::Windows::Forms::NumericUpDown^  Sigma;
	private: System::Windows::Forms::GroupBox^  groupBox1;
	private: System::Windows::Forms::RadioButton^  radioButton4;
	private: System::Windows::Forms::CheckBox^  checkBox4;

	private:
		/// <summary>
		/// Required designer variable.
		/// </summary>
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
```

```
            /// the contents of this method with the code editor.
            /// </summary>
            void InitializeComponent(void)
            {
this->Reset_btn = (gcnew System::Windows::Forms::Button());
this->button2 = (gcnew System::Windows::Forms::Button());
this->radioButton1 = (gcnew System::Windows::Forms::RadioButton());
this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());
this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
this->checkBox2 = (gcnew System::Windows::Forms::CheckBox());
this->checkBox3 = (gcnew System::Windows::Forms::CheckBox());
this->label1 = (gcnew System::Windows::Forms::Label());
this->radioButton3 = (gcnew System::Windows::Forms::RadioButton());
this->Sigma = (gcnew System::Windows::Forms::NumericUpDown());
this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
this->radioButton4 = (gcnew System::Windows::Forms::RadioButton());
this->checkBox4 = (gcnew System::Windows::Forms::CheckBox());

(cli::safe_cast<System::ComponentModel::ISupportInitialize^ >(this->Sigma))
->BeginInit();

this->groupBox1->SuspendLayout();
this->SuspendLayout();
//
// Reset_btn
//
this->Reset_btn->Location = System::Drawing::Point(33, 282);
this->Reset_btn->Name = L"Reset_btn";
this->Reset_btn->Size = System::Drawing::Size(100, 25);
this->Reset_btn->TabIndex = 0;
this->Reset_btn->Text = L"Reset Settings";
this->Reset_btn->UseVisualStyleBackColor = true;
this->Reset_btn->Click += gcnew System::EventHandler(this,
&Menu_1::Reset_btn_Click);
//
// button2
//
this->button2->Location = System::Drawing::Point(33, 313);
this->button2->Name = L"button2";
this->button2->Size = System::Drawing::Size(100, 25);
this->button2->TabIndex = 1;
this->button2->Text = L"Close";
this->button2->UseVisualStyleBackColor = true;
this->button2->Click += gcnew System::EventHandler(this,
&Menu_1::button2_Click);
//
// radioButton1
//
this->radioButton1->AutoSize = true;
this->radioButton1->Checked = true;
this->radioButton1->Location = System::Drawing::Point(12, 12);
this->radioButton1->Name = L"radioButton1";
this->radioButton1->Size = System::Drawing::Size(58, 17);
this->radioButton1->TabIndex = 2;
this->radioButton1->TabStop = true;
this->radioButton1->Text = L"Normal";
this->radioButton1->UseVisualStyleBackColor = true;
this->radioButton1->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::radioButton1_CheckedChanged);
//
// radioButton2
//
this->radioButton2->AutoSize = true;
this->radioButton2->Location = System::Drawing::Point(12, 166);
this->radioButton2->Name = L"radioButton2";
```

```
this->radioButton2->Size = System::Drawing::Size(74, 17);
this->radioButton2->TabIndex = 3;
this->radioButton2->Text = L"GrayScale";
this->radioButton2->UseVisualStyleBackColor = true;
this->radioButton2->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::radioButton2_CheckedChanged);
//
// checkBox1
//
this->checkBox1->AutoSize = true;
this->checkBox1->Checked = true;
this->checkBox1->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkBox1->Location = System::Drawing::Point(15, 19);
this->checkBox1->Name = L"checkBox1";
this->checkBox1->Size = System::Drawing::Size(46, 17);
this->checkBox1->TabIndex = 4;
this->checkBox1->Text = L"Red";
this->checkBox1->UseVisualStyleBackColor = true;
this->checkBox1->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::checkBoxX_CheckedChanged);
//
// checkBox2
//
this->checkBox2->AutoSize = true;
this->checkBox2->Checked = true;
this->checkBox2->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkBox2->Location = System::Drawing::Point(15, 42);
this->checkBox2->Name = L"checkBox2";
this->checkBox2->Size = System::Drawing::Size(55, 17);
this->checkBox2->TabIndex = 5;
this->checkBox2->Text = L"Green";
this->checkBox2->UseVisualStyleBackColor = true;
this->checkBox2->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::checkBoxX_CheckedChanged);
//
// checkBox3
//
this->checkBox3->AutoSize = true;
this->checkBox3->Checked = true;
this->checkBox3->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkBox3->Location = System::Drawing::Point(15, 65);
this->checkBox3->Name = L"checkBox3";
this->checkBox3->Size = System::Drawing::Size(47, 17);
this->checkBox3->TabIndex = 6;
this->checkBox3->Text = L"Blue";
this->checkBox3->UseVisualStyleBackColor = true;
this->checkBox3->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::checkBoxX_CheckedChanged);
//
// label1
//
this->label1->AutoSize = true;
this->label1->ForeColor = System::Drawing::Color::Orange;
this->label1->Location = System::Drawing::Point(9, 146);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(59, 13);
this->label1->TabIndex = 7;
this->label1->Text = L"Transforms";
//
// radioButton3
//
this->radioButton3->AutoSize = true;
this->radioButton3->Location = System::Drawing::Point(12, 189);
this->radioButton3->Name = L"radioButton3";
this->radioButton3->Size = System::Drawing::Size(63, 17);
```

```
this->radioButton3->TabIndex = 9;
this->radioButton3->Text = L"Laplace";
this->radioButton3->UseVisualStyleBackColor = true;
this->radioButton3->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::radioButton3_CheckedChanged);
//
// Sigma
//
this->Sigma->Location = System::Drawing::Point(95, 189);
this->Sigma->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4)
{15, 0, 0, 0});
this->Sigma->Name = L"Sigma";
this->Sigma->Size = System::Drawing::Size(38, 20);
this->Sigma->TabIndex = 10;
this->Sigma->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) {3,
0, 0, 0});
this->Sigma->ValueChanged += gcnew System::EventHandler(this,
&Menu_1::Sigma_ValueChanged);
//
// groupBox1
//
this->groupBox1->Controls->Add(this->checkBox1);
this->groupBox1->Controls->Add(this->checkBox2);
this->groupBox1->Controls->Add(this->checkBox3);
this->groupBox1->Location = System::Drawing::Point(25, 35);
this->groupBox1->Name = L"groupBox1";
this->groupBox1->Size = System::Drawing::Size(108, 88);
this->groupBox1->TabIndex = 11;
this->groupBox1->TabStop = false;
this->groupBox1->Text = L"Colour layers";
//
// radioButton4
//
this->radioButton4->AutoSize = true;
this->radioButton4->Location = System::Drawing::Point(12, 212);
this->radioButton4->Name = L"radioButton4";
this->radioButton4->Size = System::Drawing::Size(81, 17);
this->radioButton4->TabIndex = 12;
this->radioButton4->Text = L"Blue Motion";
this->radioButton4->UseVisualStyleBackColor = true;
this->radioButton4->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::radioButton4_CheckedChanged);
//
// checkBox4
//
this->checkBox4->AutoSize = true;
this->checkBox4->Checked = true;
this->checkBox4->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkBox4->Location = System::Drawing::Point(46, 234);
this->checkBox4->Name = L"checkBox4";
this->checkBox4->Size = System::Drawing::Size(77, 17);
this->checkBox4->TabIndex = 13;
this->checkBox4->Text = L"Orientation";
this->checkBox4->UseVisualStyleBackColor = true;
this->checkBox4->CheckedChanged += gcnew System::EventHandler(this,
&Menu_1::checkBox4_CheckedChanged);
//
// Menu_1
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(164, 356);
this->Controls->Add(this->checkBox4);
this->Controls->Add(this->radioButton4);
this->Controls->Add(this->groupBox1);
```

```
            this->Controls->Add(this->Sigma);
            this->Controls->Add(this->radioButton3);
            this->Controls->Add(this->label1);
            this->Controls->Add(this->radioButton2);
            this->Controls->Add(this->radioButton1);
            this->Controls->Add(this->button2);
            this->Controls->Add(this->Reset_btn);
            this->FormBorderStyle =
            System::Windows::Forms::FormBorderStyle::FixedToolWindow;
            this->Name = L"Menu_1";
            this->Text = L"Face Detection";
            this->FormClosed += gcnew System::Windows::Forms::FormClosedEventHandler(this,
            &Menu_1::Menu_1_FormClosed);
            this->Load += gcnew System::EventHandler(this, &Menu_1::Menu_1_Load);
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this->Sigma))
            ->EndInit();
            this->groupBox1->ResumeLayout(false);
            this->groupBox1->PerformLayout();
            this->ResumeLayout(false);
            this->PerformLayout();

            }
#pragma endregion

private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
                        FaceRecognized::Stop_btn_OnClick();
                        this->Close();
                         }
private: System::Void Menu_1_FormClosed(System::Object^  sender,
             System::Windows::Forms::FormClosedEventArgs^  e) {
                        FaceRecognized::Stop_btn_OnClick();
                         }


private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
            System::EventArgs^  e) {
                         FaceRecognized::Change_Video(0);
                         }
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
            System::EventArgs^  e) {
                         FaceRecognized::Change_Video(1);
                         }
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
            System::EventArgs^  e) {
                        FaceRecognized::Change_Video(2);
                         }
private: System::Void radioButton4_CheckedChanged(System::Object^  sender,
            System::EventArgs^  e) {
                        FaceRecognized::Change_Video(3);
                         }
private: System::Void Sigma_ValueChanged(System::Object^  sender, System::EventArgs^  e) {
                        FaceRecognized::SetSigmaVal((int)this->Sigma->Value);
                         }
private: System::Void checkBoxX_CheckedChanged(System::Object^  sender,
            System::EventArgs^  e) {
                        bool R = this->checkBox1->Checked;
                        bool G = this->checkBox2->Checked;
                        bool B = this->checkBox3->Checked;
                        FaceRecognized::SetRGB(R,G,B);
                         }


private: System::Void Menu_1_Load(System::Object^  sender, System::EventArgs^  e) {
                        blueMotion::Set_Orientation(true);
                        FaceRecognized::SetRGB(true,true,true);
```

```
                        FaceRecognized::Change_Video(0);
                         }

private: System::Void Reset_btn_Click(System::Object^  sender, System::EventArgs^  e) {
                        this->radioButton1->Checked = true;
                        this->checkBox1->Checked = true;
                        this->checkBox2->Checked = true;
                        this->checkBox3->Checked = true;
                        this->checkBox4->Checked = true;
                        this->Sigma->Value = 3;
                         }

private: System::Void checkBox4_CheckedChanged(System::Object^  sender,
             System::EventArgs^  e) {
                        if ( checkBox4->Checked)
                          blueMotion::Set_Orientation(true);
                        else
                          blueMotion::Set_Orientation(false);
                         }
  };
}
```