Master Thesis

Model-Based Eye Detection and Animation

Sandra Trejo Guerrero

LiTH - ISY - EX - - 06 / 3909 - - SE

Model-Based Eye Detection and Animation

Department of Electrical Engineering - Image Coding Group, Linköpings Universitet

Sandra Trejo Guerrero

LiTH - ISY - EX - - 06 / 3909 - - SE

Examensarbete: **20 p**

Level: **D**

Supervisor: **Marco Fratarcangeli**,
Department of Electrical Engineering - Image Coding Group, Linköpings Universitet

Examiner: **Robert Forchheimer**,
Department of Electrical Engineering - Image Coding Group, Linköpings Universitet

Linköping: **June 2006**

LINKÖPINGS UNIVERSITET

Avdelning, Institution
Division, Department

Department of Electrical Engineering
581 83 LINKÖPING
SWEDEN

Datum
Date

June 2006

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Master Thesis
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN

ISRN
LiTH - ISY - EX - - 06 / 3909 - - SE

Serietitel och serienummer **ISSN**
Title of series, numbering 3909

URL för elektronisk version

http://www.ep.liu.se

Titel
Title

Model-Based Eye Detection and Animation

Författare
Author

Sandra Trejo Guerrero

Sammanfattning
Abstract

In this thesis we present a system to extract the eye motion from a video stream containing a human face and applying this eye motion into a virtual character. By the notation eye motion estimation, we mean the information which describes the location of the eyes in each frame of the video stream. Applying this eye motion estimation into a virtual character, we achieve that the virtual face moves the eyes in the same way than the human face, synthesizing eye motion into a virtual character. In this study, a system capable of face tracking, eye detection and extraction, and finally iris position extraction using video stream containing a human face has been developed. Once an image containing a human face is extracted from the current frame of the video stream, the detection and extraction of the eyes is applied. The detection and extraction of the eyes is based on edge detection. Then the iris center is determined applying different image preprocessing and region segmentation using edge features on the eye picture extracted.

Once, we have extracted the eye motion, using MPEG-4 Facial Animation, this motion is translated into the Facial Animation parameters (FAPs). Thus we can improve the quality and quantity of Facial Animation expressions that we can synthesize into a virtual character.

Nyckelord
Keyword

Eye motion estimation, MPEG-4 Facial Animation, Feature Points, Facial Animation Parameters, Feature extraction, face tracking, eye tracking.

Abstract

In this thesis we present a system to extract the eye motion from a video stream containing a human face and applying this eye motion into a virtual character. By the notation eye motion estimation, we mean the information which describes the location of the eyes in each frame of the video stream. Applying this eye motion estimation into a virtual character, we achieve that the virtual face moves the eyes in the same way than the human face, synthesizing eye motion into a virtual character. In this study, a system capable of face tracking, eye detection and extraction, and finally iris position extraction using video stream containing a human face has been developed. Once an image containing a human face is extracted from the current frame of the video stream, the detection and extraction of the eyes is applied. The detection and extraction of the eyes is based on edge detection. Then the iris center is determined applying different image preprocessing and region segmentation using edge features on the eye picture extracted.

Once, we have extracted the eye motion, using MPEG-4 Facial Animation, this motion is translated into the Facial Animation parameters (FAPs). Thus we can improve the quality and quantity of Facial Animation expressions that we can synthesize into a virtual character.

Keywords: Eye motion estimation, MPEG-4 Facial Animation, Feature Points, Facial Animation Parameters, Feature extraction, face tracking, eye tracking.

Acknowledgements

I would like to thank my supervisor, Prof. Dr. Forchheimer, for his support throughout the development of my Thesis.

I would like to thank my coordinator, Marco Fratarcangeli. This thesis really would not have been completed without his support, guidance and patience.

I also would like to thank Gabriele Fanelli and Mattias Hansson, for their permanent support in these months, and their friendship.

I also would like to thank all my Erasmus friends, specially to Sven Arndt for his constant support every day, and to Victoria Climent Fenollar, who has believed in my all these months.

I also would like to thank Tobias Ahlström, for his support throughout these last months, his company, his help, and his friendship.

Finally, I am grateful to my parents, family and Antonio Vega Palomas for their continuous encouragement.

My opponent Ravi Mandadi also deserves my thanks.

Nomenclature

Most of the reoccurring abbreviations and symbols are described here.

Abbreviations

FA	Facial Animation
MPEG-4 FA	Motion Picture Experts Group Facial Animation
FAP	Facial Animation Parameters
FP	Feature Points
fps	frames per second

Contents

Chapter 1

Introduction

1.1 Context

Nowadays real time communications, that enables the sender and the receiver to transmit video contents generated by for example a web camera or a video mobile camera are a very relevant field. This is due to an increasing need for communication over large distance.

We can transmit a video conference by using a radio link recorded by a video mobile camera, or via Internet, using a web camera. In both cases, the goal of the communication is transmitting the information in real time and with the lowest bandwidth, which implies a low bit-rate. In order to achieve these purposes, we need to compress the information. The compression of the information can be done according to different concepts: Model-Based Coding and a standard to compress the information, how is MPEG-4. Related with the first concept, many facial animation models have been developed based on Model-Based Coding, a concept which will be explained later. Regarding MPEG-4, a part of the standard has been developed in order to deal with the animation of virtual faces: MPEG-4 Facial Animation. MPEG-4 FA is the standard used to translate, code and transmit the motion efficiently, using a low bandwidth. On the receiver side the motion is synthesized.

The Facial Animation models have experienced great evolution in the last years. Animation of human faces has been an active topic of research in computer graphics since the 1970's. The most important characteristics of these models are that they must be efficient, accurate, and use few parameters to allow low bit-rate(low bandwidth).

According to these models, we only transmit some parameters, which will be interpreted at the receiver side of the communication system. This in order to reduce the quantity of information sent over the channel, the parameters should be coded to admit efficient transmission in terms of bandwidth and reconstructability at the receiving side of the system.

One of this models is Candide. Candide is a parameterized face mask specif-

ically developed for model-based coding of human faces. It uses a low number of polygons (approximately 100) representing the shape of the face. This allows fast reconstruction with moderate computing power.

During the 90's, several advanced face models have been created by different research groups and companies, sometimes using thousands of polygons to describe a face, and sometimes with complex underlying physical and anatomical models. Despite this, the Candide model is still widely used, since its simplicity makes it a good tool for image analysis tasks and low complexity animation.

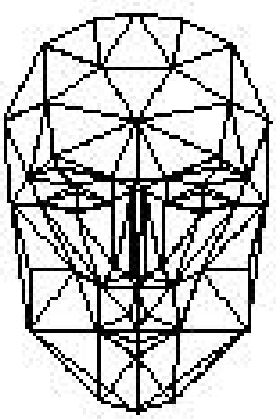


Figure 1.1: Candide-1, figure extracted from [26]

1.2 Problem: Eye Motion Estimation and Animation

The problem dealt with in this thesis is estimating the eye motion from a video stream containing a human face and applying this motion estimation to a virtual face. By eye motion we understand the information which tells us the location of the eyes in each frame of the video. Applying this eye motion to a virtual face, we achieve that the virtual face moves the eyes in the same way as the human face, synthesizing eye motion into a virtual character.

After detecting the eye motion, we translate this motion into Facial Animation parameters (FAP). Through the FAPs, MPEG-4 enables the animation of a face model. FAPs are based on the study of the facial movements and are closely related with the muscle actions.

Since, MPEG-4 facial animation is a standard used to describe the motion. Regarding the eye motion, it is important to point out that with the same FAPs, we can synthesize the same eye motion using different virtual characters. Thus, we can improve the quality and quantity of facial expressions shown by a virtual character.

Figure 1.2: Eye motion estimation

Figure 1.3: Eye motion synthesis

1.3 Solution

In this thesis we present a non-intrusive eye tracker which can detect and track a user's eyes. We receive an image provided for instance by a web camera or a video mobile camera. Features such as color, edges and shape are incorporated in different stages of our method.

First, we extract the face image from the input image, employing the face detector already available in OpenCV. OpenCV is an open-source image processing library [1]. Second, we process the image working with the edges of the image. Detecting the vertical edges in the face image, we know the position of the eyes and we can extract them from the image region which contains only the face information. Therefore, we reduce the number of pixels to process and we reduce the risk of mistaking the eyes with another part of the image. After that we have extracted the two eye regions, we can process them in order to detect the iris. Finally, detecting the iris in the eye images, we can obtain the coordinates where the iris is located.

1.4 Requirements

The system for tracking the eyes should be robust, non intrusive and inexpensive and this is quite a challenge in the computer vision field. Nowadays eye tracking receives a great deal of attention for applications such as Facial Expression analysis and driver awareness systems.

Today we can find very accurate eye tracker using external devices. Most modern eye-trackers use contrast to locate the centre of the pupil and use infrared cameras to create a corneal reflection, and the triangulation of both to determine the fixation point. However, eye tracking setups vary greatly; some are head-mounted, some require the head to be stable (for example, with a chin rest), and some automatically track the head as well.

The eye-tracker described in this thesis is characterized for being a non-invasive eye-tracker, we do not need any external devices for tracking the eyes, besides the web camera which records the video stream.

Moreover, the efficiency of the eye-tracker is very important when working with real time communications. Real time communications are those video streams with frame rate higher than 15 frames per second (fps) and with a delay of only few frames. If we achieve this, we could apply our eye-tracker to this kind of communications, for instance mobile communications or IP communications.

As it is mentioned in a previous section, the purpose of this eye-tracker is to detect the eye motion from a video stream containing a human face, and apply this motion to a virtual character, in order that the virtual character synthesizes the same eye motion as the human face. Translating the eye motion into the FAPs using MPEG-4 FA, transmitting them in each frame, and finally, synthesizing this information on the receiving side, we avoid to transmit a big quantity of information, we save bandwidth and we achieve a very low bit rate. In this way, we work with low resolution and our application could be suitable for video-conferences using a web camera or a mobile camera phone.

1.5 Previous work

Eye tracking is a technique used in cognitive science, psychology, human-computer interaction, advertising, medical research, and other many areas. The most widely used current designs are video based eye trackers. In the recent years, various methods of eye detection, iris detection and eye tracking have been proposed. Here, we present some of these approaches.

Haro et al. [2] use the physical properties of pupils along with their dynamics and appearance to extract regions with eyes. Infrared lighting is used to capture the physiological properties of the eyes, Kalman trackers are used to model eye/head dynamic and a probabilistic based appearance model is used to represent eye appearance. The camera has two concentric rings of IR LEDs,

one a long the camera's axis, and off-axis, to exploit the red eye effect. They generate two interlaced images for a single frame. The image where the inner ring is on, the bright image, has white pupils. The image where the outer ring is on, the dark image, has dark pupils. The difference image that results by subtracting both images contains valuable information about the iris location. Since this approach uses infrared cameras in order to track the eyes and we are interested in a non-invasive application, we will not consider this approach.

Singh et al. [3] describe a non-intrusive approach for real-time detection of driver fatigue. It monitors the driver's eyes, estimating the position where they are located in order to detect the micro-sleeps (short periods of sleep). The system deals with skin-color information in order to search for the face in the current frame. Then, they reduce the search space by analyzing the horizontal gradient map of the face, taking into account the knowledge that the eyes regions have a great change in the horizontal intensity gradient. In order to find the exact position of the pupil, they use a gray scale model matching. Gray scale correlation is a pattern matching technique that allows to search for a pattern in the image. If the match score of a pixel is about the acceptance level the application considers that the iris of the eye is center at that pixel. As we stated earlier, this system detects micro-sleep symptoms in order to diagnose driver fatigue. As the driver fatigue increases, the blinks of the driver tend to last longer. They can determine the blink rate by counting the number of consecutive frames in which the eye remain closed. Since this approach has been develop in order to detect the driver fatigue it is not really accurate in order to locate the position of the pupil working with different gaze orientation due the method works with a template containing an open eye with the iris located in the center of the eyes.

Kashima et al. [4] present an iris detection method that is adaptable to various face and eye movements from a face image. First, the method determines the standard skin color from the previous frame. The face region is divided by the color difference from the standard skin color. Secondly, the eye and the mouth regions are extracted from the face region by hybrid template matching, which consist of the four directional features and color distance features. After, the iris regions are detected by using saturation and brightness from the extracted eye regions in order to separate the eye from the white part of the eye, segmenting the image. Then the edge of the iris is detected from the gray-scale image in iris region by applying the Prewitt's operator. Finally, the iris positions are determined by the Hough Transform. Thus, this method can detect the iris no matter which gaze direction.

Weiner et al. [5] present a system for gaze redirection in color faces images via eye synthesis and replacement. First, the face is detected via skin detection. Skin is detected using the C_r (chromatic red) and C_b (chromatic blue) component of the YC_bC_r color space. Following the face detection and extraction, the eye detection is performed working with the red channel of the color image. Edge detection is applied and after the position of the iris is determined by Hough Transform. Then, fine determination of the iris center position and the iris radius are performed by applying another version of the Hough Transform in a rectangular bounding box containing the eye. Finally the image is segmented

using the fact that the iris is darker than its surroundings. The intensity image of the eye image is thresholded using an adaptative threshold designed to be bigger than the intensity of the iris pixels but lower than the intensity of the eye white or skin. This system provides good iris segmentation. However it is affected by the lighting conditions.

Chapter 2

Background

Almost all the solutions to computer vision problems involve manipulation and image processing. So in this chapter I include the basic concepts needed in order to understand this paper.

2.1 A visual communication system

A general communication system can be described as follows; the transmitter sends a message to the receiver via a channel. The channel can be a telephone line, a radio link, a satellite link or a storage device. If the input and output of the communication system are images, this system is called a visual communication system.

Usually, the capacity of the channel is limited and we need to compress the information in order to achieve a low bit rate, low bandwidth and therefore fast communications. As we commented before, the focus of Model-Based coding is to allow the transmission of the motion extracted from facial images, for instance image video streams containing human faces.

So in order to compress the information, a facial animation model is constructed and adapted to the current image sequence. Parameters describing how to transform the wire-frame of the facial model in order to animate a face are extracted, coded and transmitted through the channel. At the receiver side, in which is set up the facial animation model, as well, the wire-frame of the model is adapted according to the parameters received, resulting in an image very close to the original image.

2.2 Facial Animation Theory

2.2.1 Model Based Coding

According to Jörgen Ahlberg [6], *the main idea of Model-Based Coding of video sequences is as follows: At the encoding side of a visual communication system, the frame extracted from the video stream is analysed using different models and*

techniques. These models describe the shape and the texture of the objects.

Instead of transmitting the full image pixel-by-pixel, the image is handled as a 2-D projection of 3-D objects. In order to reduce the quantity of information and therefore working with low bandwidth, parameters describing the objects are extracted, coded and transmitted. Typical parameters are size, position and shape.

At the receiver side of the system, the parameters are decoded and the decoders model is modified according to this parameters. The receiver and the transmitter have the same model, in this way we reduce the quantity of information to transmit and we only transmit the parameters with which the receiver will know how to adapt the model to the current frame of the video sequence. The receiver collects the information and adapts the model of the transmitter with this parameters. Therefore, at the decoder, the video sequence is synthesized by rendering the models at the estimated positions.

These parameters usually describe how to adapt the model in the receiver side, for instance how to rotate and translate the model in order to obtain as an output with the maximum similarity respect the input frame.

2.2.2 Terminology for Face Image Processing

In Model-Based Coding different kinds of image processing, analysis and synthesis are performed. In order to process the face image, we have to process different stages. The five stages are: Face Detection, Facial Feature Extraction, Face/Facial Feature Tracking, Face Parameter Coding and Face Synthesis. Usually the application of the previous processing benefits the next one, decreasing the quantity of information to transmit. For instance in this thesis, we extract the face from the video sequence in order to reduce the quantity of information and to prepare the image to the next processing steps.

Each stage of the face image processing is described below:

1. Face Detection: Given an image, face detection algorithm tries to find the face area in the input image. If there is any face in the input image, the face detector returns the pixels containing the face. Obviously, the performance of the Face Detector depends on the head rotation. In our thesis we have used a face detector which work only with frontal faces without dealing with head rotation.

2. Facial Feature Extraction: Given an image and the location of the face, facial feature extraction is related to the process of extracting the coordinates which define the features of the face in the current image. From the extracted facial area, we can calculated the facial model parameters.

3. Face Tracking or Facial Feature Tracking: Tracking of a face or facial features is the process of following a detected face or extracted facial features

through an image sequences. In this paper we deal with how to track the movement of the eyes in a video stream.

4. Face Model Parameter Coding: with face model parameter coding we introduce the task to represent a set of face model parameters in an efficient way for transmission over a channel with low capacity. We code the parameters to reduce the quantity of information to transmit. The parameters should be decoded and synthesized in the receiver side of the system with reasonable image quality.

5. Face Synthesis: This task is related to the rendering of the information from the transmitter from a set of parameters in the model so as to reproduce with the minimum error the input image on the receiver side of the system.

2.2.3 Face Model Parameters

Three kinds of parameters need to be transmitted in a model-based coding scheme: shape, texture, and animation parameters.

1. Shape parameters: They content information about the shape and contours of the face. Shape parameters need to be transmitted in the first frame only, to tell the 3-D shape of the head. Specially, these parameters can be 3-D coordinates of facial feature points (FDP points), vertices of a face model, or a complete 3-D mesh.

2. The texture is an image to be mapped onto the face model. In its uncompressed form, the texture will by far constitute the largest part of the transmitted data, and compression is certainly needed. The most common solution is to use a waveform coder for the first frame and then only send the differential or no images at all for the following frames. However, we can use other methods in order to compress the information.

3. The animation parameters consist of global and local motion parameters. With global motion, the rotation and translation of the face are modelled. The amount of data to be transmitted is small, but it needs to be transmitted every frame. Whereas, by local motion parameters, we can represent mimics and facial expressions. Usually, the local motion parameters transmit a huge quantity of information, therefore these parameters will often need compression.

2.2.4 Candide Model

According to Jörgen Ahlberg in [7], Candide is a parameterized face mask specifically developed for model-based of human faces. Its low number of polygons, approximately 100, allows fast reconstruction with small computing power.

Candide is controlled by global and local Action Units (AUs). The global ones correspond to rotations around three axes. The local Action Units control

the mimics of the face so that different expressions can be obtained.

The original Candide model described by Rydfalk, contained 75 vertexes and 100 triangles. This version is rarely used. The most widespread version, the de facto standard Candide model, is a slightly modified model with 79 vertices, 108 surfaces and 11 Action Units. This model was created by Mårten Strömberg while implementing the first Candide software package and is refereed to as Candide-1. Candide-1 is shown in figure 1.1.

Later, Bill Welsh at British Telecom created another version with 160 vertices and 238 triangles covering the entire frontal head (including hair and teeth) and the shoulders. This version, known as Candide-2 also included in the Candide software package, is delivered with only six Actions Units. Candide-2 is shown in figure 2.1.

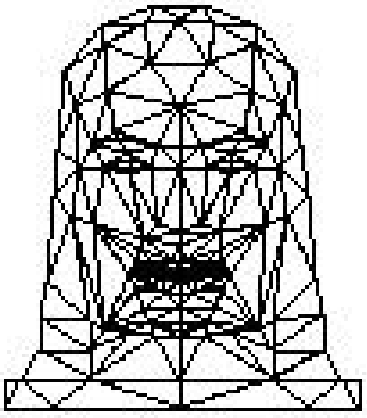


Figure 2.1: Candide-2, figure extracted from [26]

Candide model is still widely used, since its simplicity makes it a good tool for image analysis tasks and low complexity animation. However, there was a need for an updated model, due to the following reasons:

1. The very crude mouth and eyes make the Candide model very unrealistic. Adding only a few vertices improve the quality significantly.
2. A standard for which facial feature points should be used in face animation has been set in MPEG-4. Several of those facial feature points (FFPs) are missing among the vertices in the Candide model.

In order to deal with these issues several vertices have been added on the mouth, cheek, nose and eyes, improving the correspondence between Candide Model and the FFPs of MPEG-FA. The Action Units defined in Candide-1 have been extended to include these now vertexes. Thus, animation can be performed by MPEG-4 FAPs as well as Action Units. This version is known as Candide-3. Candide-3 is shown in figure 2.2.

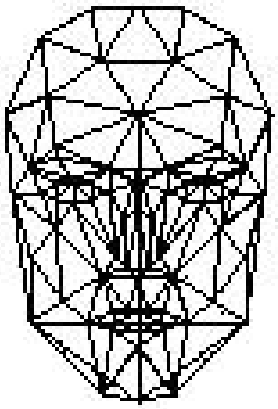


Figure 2.2: Candide-3, figure extracted from [26]

Additionally, the functionality has been extended as well, by allowing Shape Units. A Shape Unit defines a deformation of a standard face toward a specific face. The Shape Parameters thus describe the static shape of a face, while the Animation Parameters describe the dynamic shape. Shape Parameters are invariant over time, but specific to each individual. Animation Parameters naturally vary over time, but can be used for animating different faces.

Description of the Candide Model

The model is a wire-frame model where a texture can be mapped onto its surfaces. The vertex coordinates are stored in a wfm-file, and can be seen as a 3N-dimensional vector $\bar{g}$ contains the (x,y,z) coordinates of the vertexes.

The model is then reshaped according to

$$g(\sigma, \alpha) = \bar{g} + \mathbf{S}\sigma + \mathbf{A}\alpha \tag{2.1}$$

where the resulting vector g contains the new (x,y,z) vertex coordinates. The columns of $\mathbf{S}$ and $\mathbf{A}$ are the Shape and Animation Units respectively, and thus the vectors σ and α contain the shape and animation parameters.

Since we also want to perform global motion, we need a few more parameters for rotation, scaling and translation. Thus, we replace (2.1) with

$$t\mathbf{g} = \mathbf{R}s(\bar{g} + \mathbf{S}\sigma + \mathbf{A}\alpha) + t \tag{2.2}$$

where $\mathbf{R} = R(r_x, r_y, r_z)$ is a rotation matrix, s is the scale, and $\mathbf{t} = t(t_x, t_y, t_z)$ the translation vector.

The geometry of our model is thus parameterized by the parameter vector

$$\mathbf{p} = [\nu, \sigma, \alpha] = [r_x, r_y, r_z, s, t_x, t_y, t_z, \sigma, \alpha] \tag{2.3}$$

where ν is the vector of global motion parameters. The file format supports having different scalings in the three dimensions, i.e,

$$\mathbf{g} = \mathbf{R}\mathbf{S_3}(\bar{g} + \mathbf{S}\sigma + \mathbf{A}\alpha) + t \tag{2.4}$$

where $\mathbf{S_3} = S_3(s_x, s_y, s_z)$.

2.3 MPEG-4 Facial Animation

As is defined in [8], *MPEG-4 is the first audiovisual representation standard modeling an audiovisual scene as a composition of audiovisual objects with specific characteristics and behavior, notably in space and time. The object composition approach allows MPEG-4 to support for new functionalities, such as content-based interaction and manipulation, as well as improvements to already available functionalities, such as coding efficiency, by just using for each type of data the most adequate coding technology.*

The standardization of the parts of this technology, essential to guarantee interoperability, may significantly accelerate the deployment of applications using synthetic human heads, which represent real or fictitious humans. The animation of 3D facial models requires animation data, which may be synthetically generated or extracted by analysis from real faces, depending on the application.

Taking into account the relevance of the applications and the maturity of facial animation technology, MPEG-4 decided to standardize the necessary tools to allow the provision of a new range of applications relying on standardized facial animation technology. The face object specified by MPEG-4 is a representation of the human face structured in a way that the visual manifestations of speech are intelligible, the facial expressions allow the recognition of the speakers mood, and also the reproduction as faithfully as possible of a real speaker is supported.

As we commented before, MPEG-4 FA is used in order to translate in a efficient way the eye motion estimated by the eye tracker application into the Facial Animation Parameters (FAPs). Therefore we achieve a compressed and efficient transmission of the information. At the receiver side, parameters are decoded and synthesized.

2.4 Structure of the human eye

The eye is one of the most complex organ in the human body. Briefly, the human eye consists of several basic parts. Among them there are: the retina, the lens, the vitreous body, the cornea and a number of protecting structures.

Here the different parts of the eye are described. The information given by them is processed in order to implement the algorithm is extracted and processed. These parts are represented in the figure 2.3.

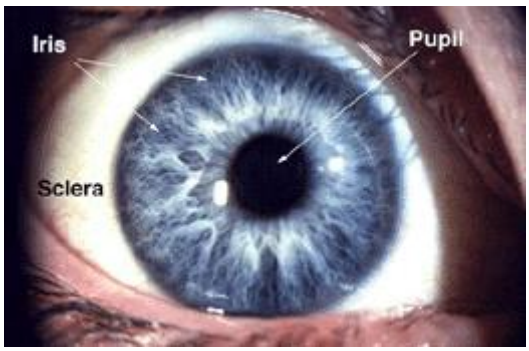


Figure 2.3: Frontal view of the human eye, figure extracted from [27]

In order to detect the position of the eyes, we process the information given by the iris, the pupil, and the sclera (white part of the eye). According to [9], these parts of the eyes are described as follows

1. *The Iris is the coloured part of the human eye. A simple description of the iris is that it is a coloured diaphragm of variable size whose function is to adjust the size of the pupil to regulate the amount of light admitted into the eye. It does this via the pupillary reflex (which is also known as the "light reflex"). That is, when bright light reaches the retina, nerves of the parasympathetic nervous system are stimulated, a ring of muscle around the margin of the iris contracts, the size of the pupil is reduced, hence less light is able to enter the eye. Conversely, in dim lighting conditions the pupil opens due to stimulation of the sympathetic nervous system that contracts of radiating muscles, hence increases the size of the pupil.*

2. *The Pupil is located in the centre of each eye in the human body. It generally appears to be the dark "centre" of the eye, but can be more accurately described as the circular aperture in the centre of the iris through which light passes into the eye. The size of the pupil (and therefore the amount of light that is admitted into the eye) is regulated by the pupillary reflex (also known as the "light reflex"). That is, when bright light reaches the retina, nerves of the parasympathetic nervous system are stimulated, a ring of muscle around the margin of the iris contracts, the size of the pupil is reduced, hence less light is able to enter the eye. Conversely, in dim lighting conditions the pupil opens due to stimulation of the sympathetic nervous system that contracts of radiating muscles, hence increases the size of the pupil.*

3. *The sclera is the tough white sheath that forms the outer-layer of the ball. It is also referred to by other terms, including the sclerotic and the sclerotic coat (both having exactly the same meaning as the sclera). In all cases these names are due to the the extreme density and hardness of the sclera (sclerotic layer). It is a firm fibrous membrane that maintains the shape of the eye as an approximately globe shape. It is much thicker toward the back/posterior aspect of the eye than toward the front/anterior*

of the eye.

The white sclera continues around the eye; most of which is not visible while the eyeball is located in its socket within the face/skull. The main area of the eye that is not covered by the area the front part of the eye that is protected by the transparent cornea instead.

The most important property of our eye movement system is that it can move the eye from one point of the gaze to a new location very quickly. These movements are called *saccadic* eye movements, and they are the fastest movements that the human body can perform. During the day the eyes can rotate over 500 deg/sec, and one hundred thousand of these *saccades* can be performed. A set of six muscles attached to each eyeball accomplish these fast movements. They are arranged in three pairs of agonist-antagonist pairs; one of them rotates the eye horizontally (left-right), the second rotates the eye vertically (up-down), and the third allows rotation over the line of the sight.

The eye tracking application developed in this thesis provides the position of the iris of the eyes during

1. saccadic movements while face is stationary
2. eyes are stationary but the face is moving (without rotation)
3. face is moving (without rotation) and the eyes are making saccadic movements

in video frames which do not only contain a human face.

2.5 Tools and environments used

In order to develop the application the code implementing the system has been firstly developed using Matlab language programming. Afterwards, and with the purpose of improving and optimizing the code, the application has been developed using C++ language programming. In the following section, we describe, briefly, the main libraries which has been used in the application. Finally, we have designed a Graphical Interface using FLTK. Moreover, in order to check the application we have worked with different Facial Database, which are referenced in [24, 25].

2.5.1 Matlab: Image Toolbox Processing

According to [10], *the Image Processing Toolbox provides a comprehensive set of reference-standard algorithms and graphical tools for image processing, analysis, visualization, and algorithm development. You can restore noisy or degraded images, enhance images for improved intelligibility, extract features, analyze shapes and textures, and register two images. Most toolbox functions are written in the open MATLAB language. This means that we can inspect the algorithms, modify the source code, and create your own custom functions.*

The Image Processing Toolbox supports engineers and scientists in areas such as biometrics, remote sensing, surveillance, gene expression, microscopy, semiconductor testing, image sensor design, color science, and materials science. It also facilitates the learning and teaching of image processing techniques.

Key features:

1. *Image enhancement, including linear and nonlinear filtering, filter design, deblurring, and automatic contrast enhancement.*
2. *Image analysis, including texture analysis, line detection, morphology, edge detection, segmentation, region-of-interest (ROI) processing, and feature measurement.*
3. *Color image processing, including color space conversions and device-independent ICC profile import and export.*
4. *Spatial transformations and image registration, including a graphical tool for control-point selection.*
5. *Image transforms, including FFT, DCT, Radon, and fan-beam projection.*
6. *DICOM import and export.*
7. *Interactive image display and modular tools for building image GUIs.*
8. *Support for multidimensional image processing.*

2.5.2 OpenCV: Open Source Computer Vision Library

As is defined in [1], *OpenCV implements a variety of tools for image interpretation. It is compatible with Intel Image Processing Library (IPL), that implements low level operations on digital images. In spite of primitives such as binarization, filtering, image statistics, pyramids, OpenCV is mostly a high-level library implementing algorithms for calibration techniques (Camera calibration), feature detection (Feature) and tracking (Optical Flow), shape analysis (Geometry, Contour Processing), motion analysis (Motion Templates, Estimators), 3D reconstruction (View Morphing), object segmentation and recognition (Histogram, Embedded Hidden Markov Models, Eigen Objects). The essential feature of this library along with functionality and quality is performance. The algorithms are based on highly flexible data structures (Dynamic Data Structures).*

The software provides a set of image processing functions, as well as image and pattern analysis functions. The functions are optimized for Intel architecture processors.

2.5.3 FLTK: Fast Light Toolkit

According to [11], *FLTK is a cross-platform C++ GUI toolkit for UNIX/Linux (X11), Microsoft Windows, and MacOS X. FLTK provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL and its built-in GLUT emulation.*

Using FLTK we have designed the graphical interface of the eye tracker application in order to offer to the user an easy way to execute the application. Information related to this interface is included in the appendix B.

2.6 Color Space

The color magnitude is composed by two components: *Chrominance* and *Luminance*. The Chrominance identifies the coloring property, whereas the luminance is a photometric measure of the density of luminous intensity in a given direction.

A color model is an abstract mathematical model describing the way colors can be represented as numbers, typically as three components (e.g. RGB color model). In the algorithm developed in order to track the eyes, information about the color of the objects is processed. In this section, the different color spaces processed are described.

2.6.1 RGB

Color images are usually represented in RGB. These are the red, green and blue components residing in three separate intensity matrices, each one with the same dimensions as the RGB image. Combining the intensity of corresponding pixels of each matrix, the actual pixel color at the given location is created. The RGB color space is an additive model in which red, green and blue are combined in various ways to reproduce other colors. Usually these components are stored as an 8-bit integer each one. Thus we can generate 2^{24} different colors.

2.6.2 Gray color Space

A gray scale image is composed for one matrix in which the colors are shades of gray. When we convert from color images to gray-scale images less information needs to be provided for each pixel. A gray scale image is an image where the red, green and blue components have equal intensity in the RGB space, so it is only necessary to specify a single value per pixel, instead of three in full color images. The conversion formula from RGB to gray color images is:

$$Gray = 0.212R + 0.715G + 0.072B, \tag{2.5}$$

where R, G and B are the matrices containing the red, green and blue component of the full color image.

Usually, gray scale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. Gray scale images are very common in technologies and image processing algorithms; they are enough for many tasks and therefore, it is not needed to used full color images. For instance, in our algorithm, eye detection and extraction is performed using gray scale images.

2.7 Thresholding

In many computer vision applications, regions of the image, containing objects in which we are interested, need to be extracted from the whole image. Thresholding provides an easy way to perform the segmentation of the image, dealing with the different intensities of the image. Thresholding can be divided into two categories: Global Thresholding and Adaptative Thresholding. Global Thresholding is applied in this application.

Using Global thresholding we can convert a gray scale image into a binary image, which is a special type of gray scale image, containing only two different values: black and white. Single or multiple threshold levels can be applied to the image to be segmented. On the one hand, for single level thresholds, each pixel in the image is compared with this threshold, the pixels is set to white (or black) if the value of the pixels is bigger than the threshold, otherwise, the pixel is set to black (or white). On the other hand, for multiple thresholding values, instead of each pixel being compared with a single value, each pixel is compared with a band of intensities. If the pixel is included in this band, the pixel is set to white (or black), and the regions out of this band are set to black (or white).

Chapter 3

Implementation

There are three main stages in order to detect the position of the iris in a picture or in a frame extracted from a video stream. The first step is to extract the face from the whole image. As we commented before, we can get the image from a web camera or from a video mobile camera. In this system we get the video sequence containing a human face using a web camera. Usually, this picture contains the head facing the camera, and a background. To eliminate the effects of the non-face regions, for each frame from the video sequence, we use a face detector to locate the face (bound by a rectangle) in the frame. Once we have an image containing the human face, the result from the second stage are two images containing the region of the eyes. Those pictures are processed in the third stage to locate the position of the iris.

3.1 Face detection

Face detectors have broad practical applications for instance, user interface, image database, teleconferencing. In our case, the face detector is used in order to extract a region of the image containing the face in the video image.

First, to eliminate the effects of the non-face regions on the recognition performance, we need to crop the face area from the whole video frame, eliminating the background and perform the recognition on the cropped face area.

In this step we use the function *cvHaarDetectObjects* already designed in OpenCV. OpenCV is an open source library written in C++. As is defined in [12] *This function finds rectangular regions in the given image that are likely to contain objects the cascade has been trained for and returns those regions as a sequence of rectangles. The function scans the image several times at different scales. This function may apply some heuristics to reduce number of analyzed regions, such as Canny pruning. After it has proceeded and collected the candidate rectangles, it groups them and returns a sequence of average rectangles for each large enough group.* More information about this function is included in the appendix A.1.

Using cvHaarDetectObjects, we obtain the coordinates where the face image

is located. Afterwards, we extract the face region from the video image. The performance of this process is shown in images 3.1 and 3.2.

Figure 3.1: Frame image extracted from the web camera

3.2 Eye detection and extraction

After the face region is extracted from the image, the location of the eyes have to be determined. It may be intuitive to threshold the gray-level images to segment the eyes. However, the eyebrows, part of the hairs, the nostrils also display low intensity. Actually, the eyes exhibit strong horizontal transitions (vertical edges) due to the existence of the iris and eye white. The projection of the horizontal transitions is obtained applying the Sobel operator. Looking for the maximum of this projection we can extract the position of the eyes in the image containing the face. Methods for eye detection are suggested in Ref. [13].

3.2.1 Edge detection: Sobel operator

In order to detect the vertical edges of the image we apply the Sobel operator into a gray-level color image containing the face, which has been obtained in the previous step.

In a nutshell, the Sobel operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light

Figure 3.2: Face image extracted from the frame image

to dark and the rate of change in that direction. The result after applying the Sobel operator is an image in which the region of constant image intensity are set to zero, and the edges are pointed out by transitions between darker and brighter intensity values.

Sobel operator consists of a pair of 3×3 convolution kernels as shown in the figure 3.3. One kernel is simply the other rotated by 90°.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 3.3: Sobel Masks, extracted from [28]

The result after filtering the image using the Sobel mask in order to detect the vertical edges (G_x) is shown in figure 3.4.

Figure 3.4: Result after applying the Sobel Operator

3.2.2 Projection functions for eye detection

Once we have applied the Sobel mask in the gray-level image containing the face, we obtain a gray-level image in which the vertical edges are detected. We have to process this image in order to detect the position where the eyes are located. As we considered before, eyes present strong horizontal transitions

(vertical edges). Therefore around the eye area, we should obtain a high level, close to 255. In order to detect these transitions, we apply image projection functions (PF). According to [14] the concept of projection functions is defined as follows.

Suppose I(x,y) is the intensity of a pixel with location (x,y), the vertical integral projection function $IPF_v(x)$ and horizontal integral projection function $IPF_h(y)$ of I(x,y) in intervals $[y_1, y_2]$ and $[x_1, x_2]$ can be defined respectively as:

$$IPF_v(x) = \int_{y_1}^{y_2} I(x,y)dy \tag{3.1}$$

$$IPF_h(y) = \int_{x_1}^{x_2} I(x,y)dx \tag{3.2}$$

In general, all image projection function can be used to detect the boundary of different image regions. After applying the projection function we must detect a quick transition in the results. Considering PF is a Projection Function and ϵ a small constant, if the value of PF rapidly changes from z_0 to $(z_0 + \epsilon)$, then z_0 may lie at the boundary between two homogeneous regions. Given a threshold T, the vertical boundaries in the image can be identified according to:

$$\theta_\nu = max\left\{\left\|\frac{dPF_\nu}{dx}\right\| > T\right\} \tag{3.3}$$

where θ_ν is the set of vertical critical points, such as $\{(x_1, PF_\nu(x_1)),$ $(x_2, PF_\nu(x_2)), \cdots, (x_k, PF_\nu(x_k))\}$ which vertically divides the image into different regions. This property of Projection Functions is exploited in our algorithm for eye detection.

Since eyes present a strong vertical edges, after filtering the image using the Sobel mask the vertical edges are detected. Then we apply the vertical integral projection function $IPF_v(x)$ in order to detect the vertical location of the eyes in the image, where y_1 and y_2 are the y-coordinates of the top and the middle of the image, respectively, considering that the eyes are located in the upper half part of the image. These values are estimated heuristically. Looking for the maximum of the IPF, we can extract the y coordinates where the eyes are located. We extract the x coordinates where the eyes are located applying to the following definitions:

$$cte = scale\_factor * width\_image; \tag{3.4}$$

$$x\_left_1 = cte; \tag{3.5}$$

$$x\_left2 = \frac{width\_image}{2} - cte; \tag{3.6}$$

$$x\_right_1 = \frac{width\_image}{2} + cte; \tag{3.7}$$

$$x\_right2 = width\_image - cte; \tag{3.8}$$

The scale-factor is different depending if there is head rotation or not. This is issue will be consider in the next section in which eye detection dealing with head rotation is explained.

Once we get the x and y coordinates where the eyes are located, we can design the windows in order to enclose the Region of Interest, the eyes. Afterwards, we can process both of them to locate the position of the iris in each one.

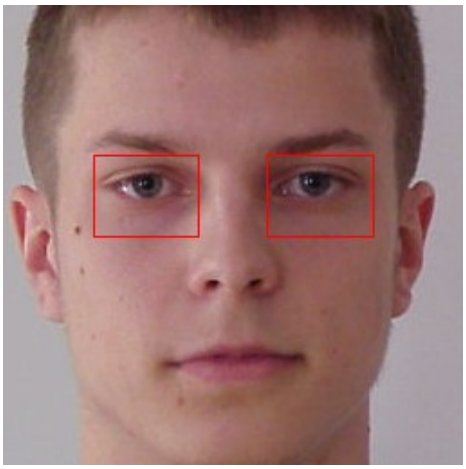


Figure 3.5: Face image with eye detected areas

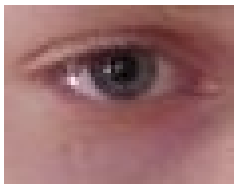


Figure 3.6: Left eye

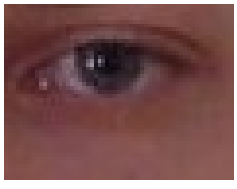


Figure 3.7: Right eye

3.2.3 Eye detection dealing with head rotation

In this section we describe a procedure in order to process and detect the eyes working with single images dealing with different angles of head rotation. In the previous section, we detected the position of the eyes when the the frame video contains a frontal face, without dealing with head rotation. The location of the vertical coordinates where the eyes are located is based on the same principle that for frontal faces. We continue assuming the eyes present a strong vertical edges. Applying the Sobel Operator and processing the information of the filtered image using a Projection Function, we obtain the vertical location of the eyes. However, we introduce a small modification in order to locate the horizontal position of the eyes, since that the position of both eyes respective to the middle of the image is not the same. Methods for detecting the eyes dealing with head rotation are suggested in Ref. [15].

First, as we commented before three parameters need to be transmitted in a model-based coding scheme: shape, texture and animation parameters. Regarding animation parameters, they are composed by global and local motion parameters. The global parameters deal with the location, rotation and translation of the face. Since from these parameters we get the information about the angle of head rotation, we calculate the distance d_0 between the projection of the mid-point of two eyes and the center of the face as follows:

$$d_0 = f * sin(\theta), \tag{3.9}$$

where f is a constant that we define heuristically, and is related to the radius of the head, supposing that the head is shaped as a circle.

Once, we have obtained the projection of the mid-point between the two eyes and the center of the face, we can locate the horizontal position of the eyes. According to the last equation, the result of this projection will be different depending on the sign of the head rotation. We define the middle position of the eyes in the video frame as follows:

$$d = \frac{image\_width}{2} + d_0, \tag{3.10}$$

After we know this information, we locate and extract the eyes applying the following definitions:

$$z_0 = 0.2(\frac{image\_width}{2} - d_0), \tag{3.11}$$

$$z_1 = \frac{image\_width}{2} 0.75, \tag{3.12}$$

$$x\_left_1 = \begin{cases} z_0 & \text{if } \theta < 0 \\ z_1 + d_0 & \text{if } \theta > 0 \end{cases} \tag{3.13}$$

$$x\_left_2 = d - z_1 \tag{3.14}$$

$$x\_right_1 = d + z_1 \tag{3.15}$$

$$x\_right_2 = \begin{cases} image\_width - z_1 - d_0 & \text{if } \theta < 0 \\ image\_width - z_0 & \text{if } \theta > 0 \end{cases} \tag{3.16}$$

Some results regarding eye detection in single images dealing with head rotation are shown in figures 3.8, 3.9, 3.10, 3.11, 3.12 and 3.13.

Figure 3.8: Face image with head rotation

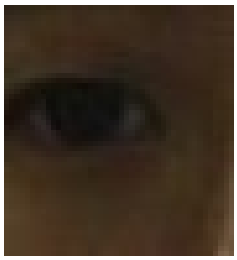


Figure 3.9: Left eye location and extraction with $\theta = -24.5$

3.3 Iris detection

As we commented before, the color images are composed from three different channels or matrices: red, green and blue. We take the assumption that the iris

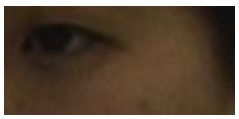


Figure 3.10: Right eye location and extraction with $\theta = -24.5$

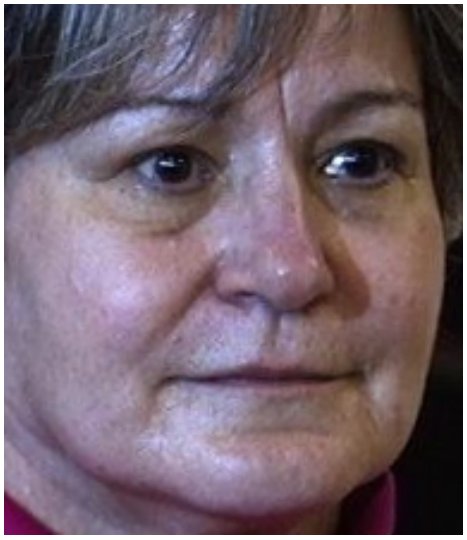


Figure 3.11: Face image with head rotation

usually exhibits low red level (both for dark and light eyes), while the surrounding pixels (sclera and skin) have significantly higher red values. Therefore, the iris center and radius detection are performed working with the red channel of the image, which emphasizes the iris border.

In this step, the eye regions are assumed to be known. The iris detection algorithm works in several steps. First iris center and radius are estimated, after the approximate iris center is known it is refined together with iris radius detection. Methods for iris detection are suggested in Ref. [16].

3.3.1 Eye image pre-processing

The goal of this section is to describe all the processing applied to the eye image before searching for the position of the iris in the image.

As we commented before, we select the red channel of the image. After that we apply a Smooth filter. These kind of filters are applied in order to reduce noise and to prepare images for further processing such as segmentation. In particular, we use a Gaussian filter. This processing allows to eliminate a great deal of noise and naturally-occurring jitter, that is present around the user in the frame due to the lighting conditions and the camera resolution, as well as the possibility of background movement.

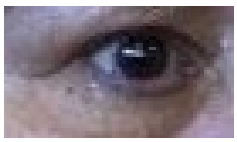


Figure 3.12: Left eye location and extraction with $\theta = 24.5$

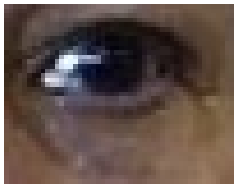


Figure 3.13: Right eye location and extraction with $\theta = 24.5$

The use of the Gaussian kernel for smoothing has become extremely popular. This has to do with certain properties of the Gaussian (e.g. the central limit theorem, minimum space-bandwidth product) as well as several application areas such as edge finding and scale space analysis.

$$g_{1D}[n] = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} e^{-(\frac{\pi^2}{2\pi^2})} & \text{if } |n| \leq N_0 \\ 0 & \text{if } |n| < N_0 \end{cases} \tag{3.17}$$

where it is common to choose N_0 between 3σ or 5σ.

Then, on the resulting image we apply a *minimum filter*. The minimum filter has the effect of spreading an image, making the lighter pixels smaller and the darker ones larger, increasing its area. Therefore, we achieve to suppress the light areas in the image. The minimum filter replaces a value with the minimum value in the neighborhood. Bellow we include the mask of the minimum filter applied.

$$\mathbf{h} = \begin{pmatrix} 0 & 2 & 1 & 1 & 0 \\ 1 & 11 & 0 & 1 & 3 \\ 1 & 2 & 2 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Finally, the eye image is convolved with a function, emphasizing the circular regions of dark pixels:

$$W(x, y, c) = \frac{\frac{sin(x^2+y^2)}{c}}{\frac{(x^2+y^2)}{c}} \tag{3.18}$$

This function emphasizes the circular areas at the center of the convolution. The parameter c controls the radius of region of maximum weight and it is chosen according to the expected iris size.

3.3.2 Approximate iris detection

Only three parameters are needed to describe and locate an iris. These are the x and y coordinates of the iris, and the iris radius.

In order to obtain the last three parameters, the central part of the eye image is checked for strong highlight presence. Remember we work with the red channel. If strong highlight is found, the central area of the eye bounding box is scanned with a circular search window with radius close to the expected iris, checking for several conditions:

1. The local minimum inside the search window should not be differ more than a given threshold from the global minimum inside the eye boundary box. It is known that the iris presents a low red level. Considering this feature, we check the intensity of each pixel in order to define if this pixel is included inside the iris region or not.

2. The circular window must be contained in the eye image. According to this, we search the circular window removing the borders of the image, considering that the iris is in a defined area in the image.

Considering the last conditions, we approximate the iris region with the circular window which has the least mean value of intensity. The iris center is found by searching for a circle which lies on the border between dark pixels of iris (low red level) and bright pixels of the eye white.

According to this, we obtain an approximation of the iris center and moreover an approximation of the radius of the iris.

3.3.3 Accurate iris detection

Once, we know an approximation of the iris center (x_c, y_c) and an approximation of the iris radius, we can refine together with the iris radius detection. We achieve this goal, applying a modified version of an algorithm developed by Jörgen Ahlberg [1999] [6].

Jörgen makes two assumptions: the iris is approximately circular, and an iris is dark against a bright background (the white part of the eye). Jörgen defined the following function:

$$f_{\Theta}(x_0, y_0, r) = \int_{\theta \in \Theta} I(x_0 + rcos\theta, y_0 + rsin\theta) d\theta, \tag{3.19}$$

where I(x,y) is a gray level image and, for example $\Theta =]0, 2\pi]$. For a given (x_0, y_0), the most likely radius of an iris center around (x_0, y_0), will then be the value for which

$$\frac{d}{dr} f(x_0, y_0, r) \tag{3.20}$$

is larger.

Consequently, we apply this method searching through a neighborhood of the previously estimated iris center (x_c, y_c) and a reasonable range of iris radius, looking for the highest values of $\frac{d}{dr} f_\Theta$.

Moreover, we introduce a modification in the last algorithm. Instead of searching for a complete circle, we limit the range of Θ between $[-\frac{\pi}{4}, \frac{\pi}{4}] \cup [\frac{3\pi}{4}, \frac{5\pi}{4}]$. In this way, we avoid the eyelid area, because the upper and lower iris parts are often covered with eyelids, and therefore not visible. Furthermore, we reduce the amount of pixels to examine and the algorithm will be faster.

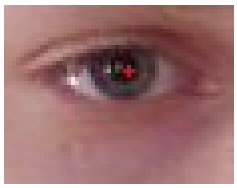


Figure 3.14: Left eye

Although, this algorithm provides very accurate results, it is very time consuming. Since, we are interested in working in real time and with the last step of the algorithm - approximation of the iris center -, we obtain a well enough approximation of the iris center, we decided to remove the refinement step from the whole application, reducing the computing time.

3.4 Blink detection

In order to improve the animation of virtual character, which we will explain in a later chapter, we include in the whole process the blinking detection. In this way, synthesizing the eye motion estimation and the blinking into the virtual character, we will achieve a more natural animation.

In the last stages we have determined the eye motion from the different video images. Subsequently, in order to detect the blinking the eyes are tracked by correlation across the time and flow movement, and appearance changes are automatically analyzed in order to classify the eye as either open or closed at each frame of the video sequence. Methods for detecting the blinking in video sequences are suggested in Refs. [17, 18].

The detection of the blinking is mainly based on observation of the correlation coefficient generated by the eye image of the current frame, and the eye image used as an open eye template. The correlation coefficient is defined as follows:

$$\mathbf{R} = \frac{\sum_{x,y}[f(x,y) - \overline{f}][t(x,y) - \overline{t}]}{\sqrt{\sum_{x,y}[f(x,y) - \overline{f}]^2 \sum_{x,y}[t(x,y) - \overline{t}]^2}}, \tag{3.21}$$

where $f(x, y)$ is the intensity of the video frame at the point (x, y), $\overline{f}$ is the average value of the video frame in the current search region, $t(x, y)$ is the intensity of the template, and $\overline{t}$ is the average value of the template image.

When the user is closing the eyes, the value of the correlation decreases, which correspond with a decrease of the similarity of the images. As a template of the open eye, we use the eye image extracted from the first frame of the video stream. We assume that in the first frame, the eyes are open and with sight fixed into the camera.

The coefficient **R** is a measure of match between the open eye template and the eye image containing the current position of the eyes. The result of this computation is a correlation score between -1 and 1 that indicates the similarity between the open eye template and the eye image extracted from the current frame, while scores closer to 0 indicate a low level of similarity. We must consider that the eye template and the current eye image may not have the same size, in this case we resize the template, in order that both image present the same size and we can apply the definition of the correlation coefficient. Once we have applied the correlation, if the correlation coefficient is lower than a given threshold we will assume this condition as true. Furthermore, we consider the flow movement. By flow movement we mean the displacement up-down of the located iris. If this displacement is bigger than a given threshold, we will consider this second condition to be true.

According to the implementation of the blinking detection, we define two thresholds: one of them controlling the correlation coefficient and the other controlling the flow movement. When both conditions are fulfilled, the application will detect the blinking. Those thresholds are set by the user through three sliders included in the graphical interface.

Chapter 4

Applications: Eye Motion Synthesis

4.1 Definition of MPEG-4 Facial Animation

According to [19]*the ISO/IEC JTC1/SC29/WG11 (Moving Pictures Expert Group - MPEG) has formulated the MPEG-4 standard. SNHC (Synthetic Natural Hybrid Coding), a subgroup of MPEG-4, has devised an efficient coding method for graphic models and the compressed transmission of their animation parameters specific to the model type.*

MPEG-4 is an object-based multimedia compression standard that allows for encoding of different audiovisual objects (AVO) in the scene independently. The MPEG-4 standard allows using a 3-D face model set up into the transmitter and receiver side, the transmission of the facial parameters in such efficient, compressed and properly way, that the encoder can predict the quality of the presentation of the information at the decoder.

According to [20] *MPEG-4 specifies a face model in its neutral state, a number of Feature Points (FPs) on this neutral face as reference points and a set of Facial Animation Parameters (FAPs), each corresponding to a particular facial action deforming a face model in its neutral state. Deforming a neutral face model according to some specific FAP values at each time instant generates a facial animation sequence. The FAP value for a particular FAP indicates the magnitude of the corresponding action, for example, a deformation of a mouth corner. For an MPEG-4 terminal to interpret the FAP values using its face model, it has to have predefined model specific animation rules to produce the facial action corresponding to each FAP. The FAP values are defined in face animation parameter units (FAPU). The FAPUs are computed from spatial distances between major facial features on the model in its neutral state.*

4.2 MPEG-4 Face Model in Neutral State

The Neutral Face is used as a reference for the interpretation of the FAP values.

It is defined as follows:

1. The coordinate system is right handed.
2. Gaze is in direction of the z-axis.
3. Eyelids are tangent to the iris.
4. The pupil diameter is one third of the IRISD0
5. Lips are in contact; the line of the inner lips is horizontal and at the same height of lip corners.
6. The tongue is flat, horizontal, with the tip of the tongue touching the boundary between upper and lower teeth.

4.2.1 The Facial Animation Parameters Units

The amount of the displacement described by a FAP is expressed in specific measurement units, called Facial Animation Parameter Units (FAPU), which represent fractions of key facial distance. Rotations are instead described as fractions of a radian.

All the FAPUs are listed in the following table extracted from [20].

Description	FAPU value
IRIS Diameter (by definition it is equal to the distance between upper and lower eyelid) in neutral face	IRISD = IRISD0/1024
Eye Separation	ES = ES0/1024
Eye-Nose Separation	ENS = ENS0/1024
Mouth-Nose Separation	MNS = MNS0/1024
Mouth-Width Separation	MW = MW0/1024
Angular Unit	AU = 10^{-5} rad

4.2.2 Feature Points

MPEG-4 specifies 84 FPs on the neutral face (figure 4.1). The main purpose of these FPs is to provide spatial references for defining FAPs. The FAPs are defined by motion of some of these FPs. Some FPs, such as the ones along the hairline are not affected by FAPs. However, they are required for defining the shape of a proprietary face model using FPs. FPs are arranged in groups such as cheeks, eyes and mouth. The location of these FPs has to be known for any MPEG-4 compliant face model.

4.3 MPEG-4 Facial Animation Parameters

For synthetic faces, the Facial Animation Parameters (FAP) are designed to encode animation of faces reproducing expressions, emotions and speech pronunciation. The 68 parameters are categorized into 10 different groups related to parts of the face. FAPs represent a complete set of basic facial actions, and thus allows the representation of most natural facial expressions. FAPs represent a complete set of basic facial actions including head motion, tongue, eye

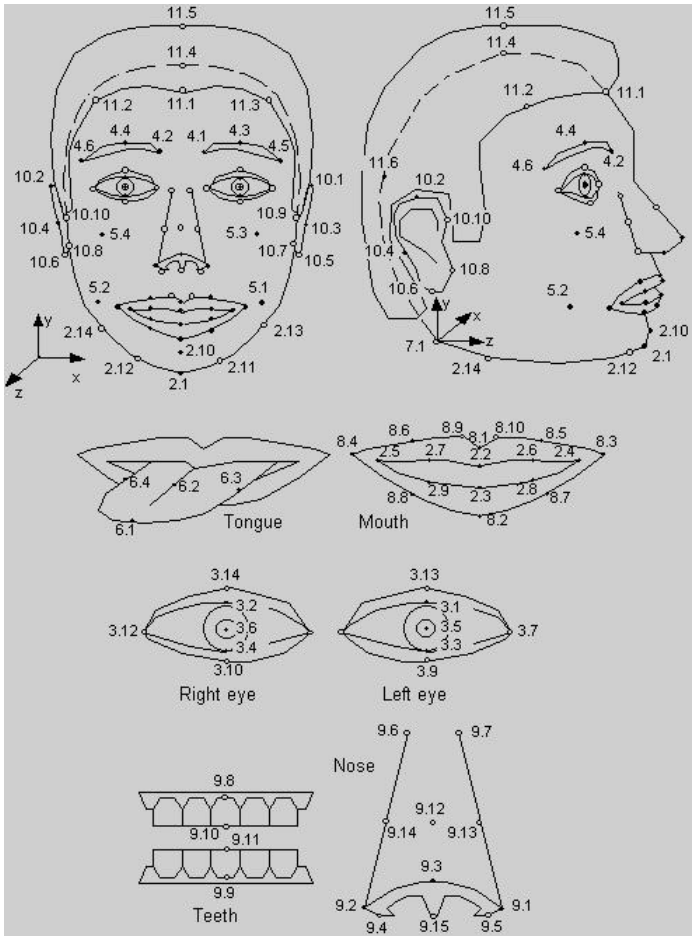


Figure 4.1: Feature Points, figure extracted from [20]

and mouth control.

The parameter set contains two high level parameters, the first group: visemes and expressions. On the one hand, The viseme parameter allows the rendering of visemes on the face without the need to express them in terms of other parameters or to enhance the result of other parameters, insuring the correct rendering of visemes. Only 14 static visemes, the most distinguished are included in the standard. In order to allow for co-articulation of speech and mouth movement, transitions from one viseme to another are defined, modelling both visemes by a weighting factor. On the other hand, expressions allow parameter allows definition of high-level facial expressions. The 6 high level expressions are joy, sadness, anger, fear, disgust and surprise. The expressions are defined with a value defining the excitation of the expression.

FAPs in groups 2 to 10 are considered low-level parameters. They specify precisely how much a FP of a face has to be moved for a given amplitude. These FAPs affect to the same FPs as group 1.

Group	Number of FAPs
1. Visemes and expressions	2
2. Jaw, chin, inner lowerlip, cornerlips, midlip	16
3. Eyeballs, pupils, eyelids	12
4. Eyebrow	8
5. Cheeks	4
6. Tongue	5
7. Head rotation	3
8. Outer-lip positions	10
9. Nose	4
10. Ears	4

4.4 Facial Animation Tables

As we explained in a previous chapter, there are different face models. In order to follow the MPEG-4 FA standard, we need to specify how each FAP affects a facial animation model. This is the purpose of the Facial Animation Tables (FATs). Using the FATs it is determined which vertices are affected by a particular FAP and how to obtain a exact reproduction of the animation.

As is described in [19], the FATs define how a model spatially deforms as a function of the amplitude of the FAPs. Each facial animation model is represented by a wire-frame with a set of vertices distributed around the face area and information about the joining triangles associated to them. This information is called "IndexedFaceSet" in a geometric representation. For each IndexedFaceSet and for each FAP, the FAT defines which vertices are affected and how they are deformed or displaced. The FAP can perform two different kinds of deformation: rotation, scale and translation.

4.5 Animation of Virtual faces

In this section we deal with the animation of virtual faces as one of the applications of eye tracking system. Animation of virtual faces is a relevant field of Image Coding in which a lot of developments have been done. The animation of virtual faces has a lot of applications in the real life, and specially in Facial and Human Animation. A lot of research is being done in this field, and consequently, a lot of challenges have been achieved.

According to MPEG-4 standard specifications, a human head is a synthetic visual object whose representation is based on VRML standard. Each decoder has its own face model. In our case, we are using the library supplied by Visage Technologies developed under MPEG-4.

As we described in the introduction chapter, our main task is to extract the eye motion from a video stream containing a human face and applying this motion to a virtual character, in order that the virtual character moves the eyes in the same way as the human face. Since, the eyes and the information related with where they are looking give us a lot of information, we can improve the

quantity and quality of the expressions that can be synthesized using a virtual character.

Once, we extract from the video stream the information related with the eye motion, we have to process the data in order to obtain the value of the MPEG-4 FA parameters (FAPs) to control the animation of the virtual character. In order to synthesize the eye motion in the virtual character, we will work with a limited number of FAPs. Thus, we will consider only the FAPs defined in the standard MPEG-4 FA related with the movements of the eyes. Afterwards, we calculate and set the proper value of each FAP and apply them to a virtual character which will reproduce the eye motion extracted from the video stream containing the human face.

4.5.1 Detection and tracking

In our application, we define two possibles stages: detection and tracking of the eyes. Once we have detect the position of the eyes in the image, we process this part of the image in order to track the eyes. The system looks in all the frames contained in the video stream to locate the eyes and the iris position. Since from the eye tracker application we obtain the position of both eyes in all the frames, we follow the movement of the eyes in the whole video stream, and therefore extract the eye motion. Processing this information, we obtain the movement of the eyes in two consecutive frames or we can obtain the absolute movement with respect to a defined or neutral position.

Once we have extracted the eye motion information, we define the position of the eyes in the first frame as the neutral position. In this frame, we make two assumptions: the eyes are looking straight to the camera and the eyes are open. The second assumption is related with the blinking detection. We use the first image of the eyes as a template in order to find the correlation between the eye image of current frame and the template containing an open eye. If the correlation coefficient is lower than a given threshold, this condition will be considered true. Then, we define the movements of the eyes with respect to the neutral position. Following the changes in the position of the eyes, we can obtain the appropriate FAPs which describe the different vertical and horizontal angle of rotation of the eyeballs with respect to the neutral state defined in the standard MPEG-4 FA.

As we described in the implementation chapter, we detect the blinking, as well. As is referenced in the correspondent appendix, the user has to introduce two threshold in order that the application decides that the eyes are closed: the first one is the transition value of the correlation coefficient with which we can decide if there is possible blinking or not. If the correlation coefficient is lower than the given threshold, this condition will assume true. The second one is related with the flow movement. By flow movement, we understand the up-down displacement of the y-coordinate of both iris centers. If this displacement is bigger that a given threshold, this condition will be true. If both conditions are fulfilled, the application will detect the blinking in the current video frame. Besides obtaining the position of the eyes in each frame by using the eye tracker application, we get a parameter pointing out the blinking. This parameter will

be set to 1 if the application detects blinking, otherwise this parameter will be set to 0. Thus using this value we can set to the correct value of the FAP related with the blinking.

4.5.2 FAP Estimation

After obtaining the eye motion estimation from the eye tracker application, we process the data in order to obtain the proper value of the Facial Animation Parameters which will describe the animation of the eyes in a virtual character. First, we define as a neutral position of the eyes the location of them in the first frame. Consequently all the location of the eyes in the following frames will be refered to the position of the eyes in the neutral position. Obtaining this vector movement, we can translate it into MPEG-4 FAPs, just working with the appropriate parameters, which will consolidate the eye motion into the virtual character.

The FAPs which allow to animate the eyes of a virtual character are the following:

FAP name	FAP description	Units	Group
close_t_l_eyelid	Vertical displacement of top left eyelid	AU	3
close_t_r_eyelid	Vertical displacement of top right eyelid	AU	3
yaw_l_eyeball	Horizontal orientation of left eyeball	AU	3
yaw_r_eyeball	Horizontal orientation of right eyeball	AU	3
pitch_l_eyeball	Vertical orientation of left eyeball	AU	3
pitch_l_eyeball	Vertical orientation of right eyeball	AU	3

As we commented before, as a result from the eye tracker application, we obtain the eye motion described for the human face contained in the video stream, which provides us with the position of the eyes in each frame of the video sequence. We are working in a discrete domain. The position of the eyes is expressed in pixels, as a consequence the difference between the position of the eyes in the neutral position and in the current frame will be expressed in pixels, as well. The FAPs concerned with the eyes are expressed in angular units. Therefore, we have to find a way to translate the result from the eye tracker application in pixels into the FAPs, measured in angular units.

In order to solve this, we study the maximum vertical and horizontal angle that the eyes can go through in a virtual character. We modify the last FAPs concerned with the animation of the eyes in order to get the maximum value of these angles. Since it is difficult to define the maximum angle in a virtual character, we guess the maximum angle with which the expression of the virtual facial is natural and it can be considered as a human expression. According to this, by evaluating the movement of the eyes with different angles, horizontal and vertical, we obtained 60 degrees for the horizontal angle and 30 degrees for the vertical angle. Moreover, we extract heuristically the maximum displacement that the eyes can experience. Thus, we can define for both kind of movements (vertical and horizontal), according to the number of pixels that the movement takes, the angle that the eyes should be rotated. Therefore, we obtain per each pixel of movement of the eyes in the human face, how many de-

grees we have to rotate the eyeball into the virtual character in order to describe the same eye motion extracted from the video sequence containing a human face.

$$\frac{horizontal\_degrees}{pixel} = 4 \tag{4.1}$$

$$\frac{vertical\_degrees}{pixel} = 2.5 \tag{4.2}$$

According to this value, we can define the horizontal and vertical angle of rotations as follows:

$$\alpha_h = (x_0 - x_i) * \frac{horizontal\_degrees}{pixel} * \frac{\pi}{10^{-5} * 180}[AU] \tag{4.3}$$

$$\alpha_v = (y_0 - y_i) * \frac{vertical\_degrees}{pixel} * \frac{\pi}{10^{-5} * 180}[AU] \tag{4.4}$$

$$AU = 10^{-5} rad \tag{4.5}$$

where (x_0,y_0) is the position of the eyes in the first frame of the video stream, and (x_i,y_i) is the position of the eye in the current frame. These equations are applied for both eyes, defining the neutral position of the eyes (x_0,y_0) for each eye, and also the current position of the eyes in the current frame (x_i ,y_i), obtaining α_{h_left}, α_{h_right}, α_{v_left} and α_{v_right}.

After that, we assign each of the last values to the appropriate FAPs, which are *yaw_l_eyeball*, *yaw_r_eyeball*, *pitch_l_eyeball* and *pitch_r_eyeball*, respectively.

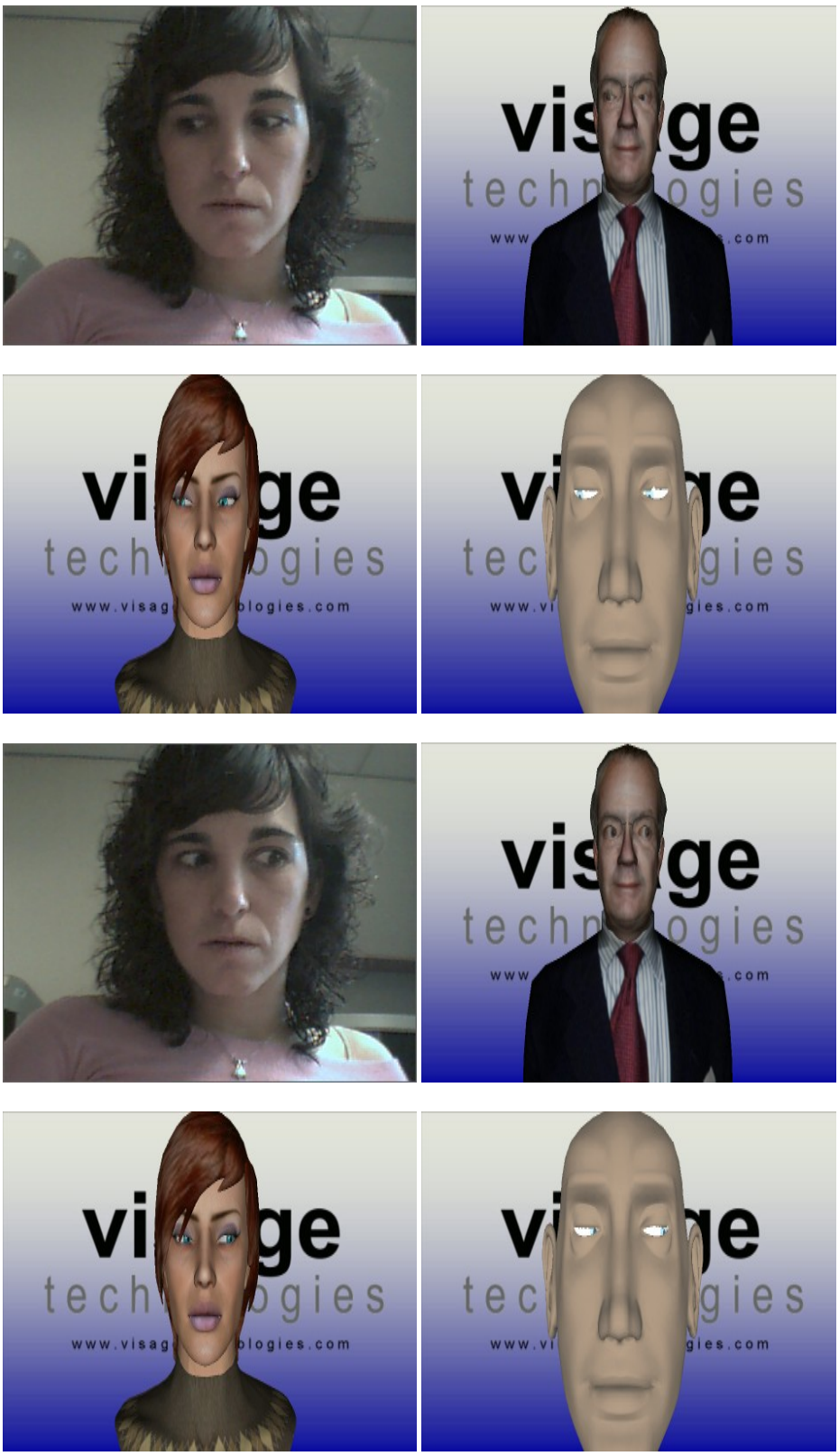
Considering the blinking, we work with the two first parameters of the table: *close_t_l_eyelid* and *close_t_r_eyelid*. From the result received from the eye tracker application we decide the value of these parameters. If the eye tracker application detects blinking, we set these parameters to the maximum value (1860 IRISD), blinking the eyes of the virtual character. Otherwise, we do not modify these parameters and we guess that eyes are open. Furthermore, we control the down-vertical displacement of the eyes. As we described in the last chapter, in the section related with blinking detection, if this movement is bigger than a given threshold decided heuristically, we will assume that the eyes are closing or eyelids are in half way. In the last case, we modify these parameters in order to place the eyelids half way, as well.

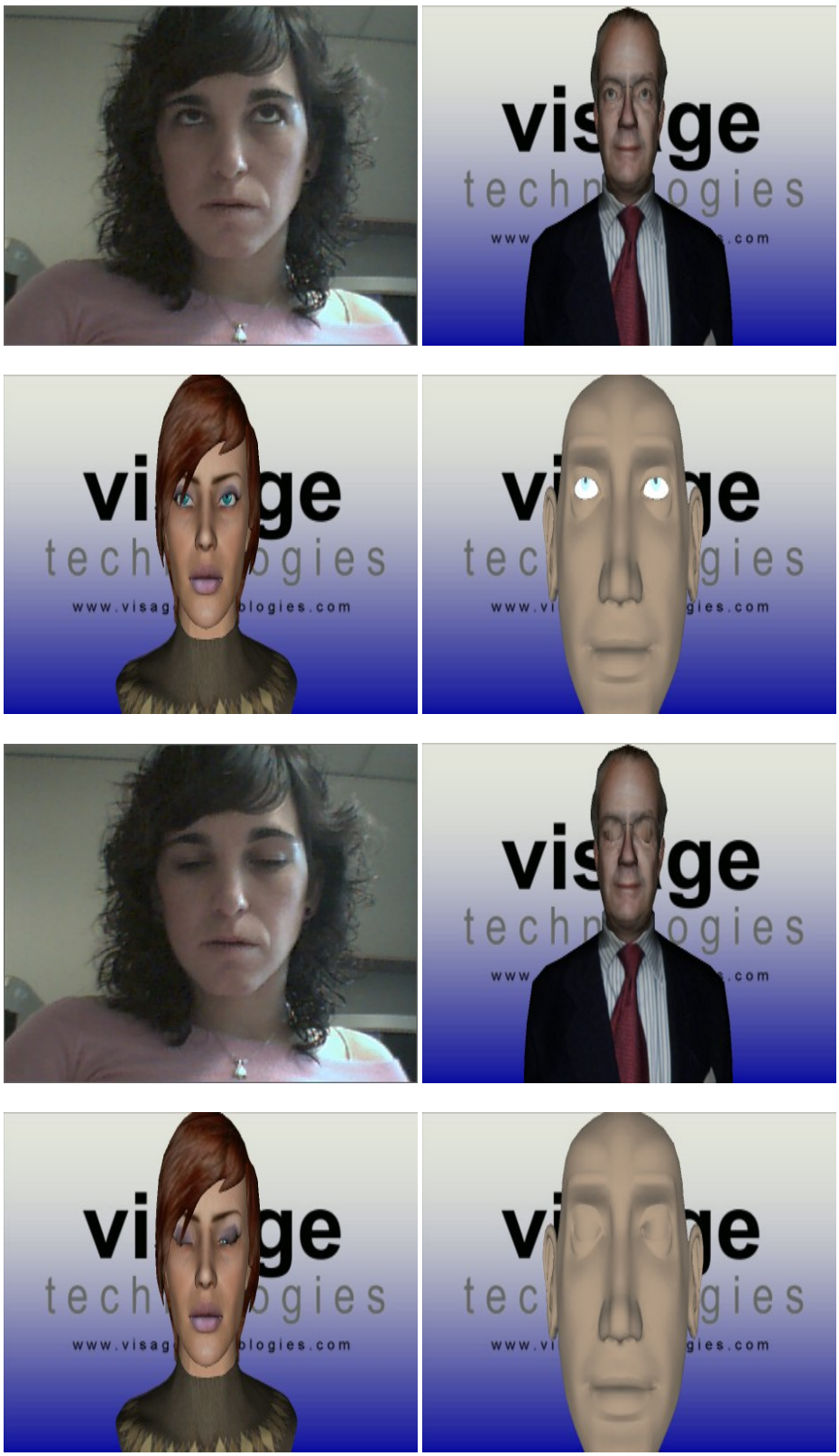
Figure 4.2: Animation of virtual character 1

Figure 4.3: Animation of virtual character 2: blinking

Figure 4.4: Animation of virtual character 3

Now, we include some frames extracted from videos containing a human face and frames extracted from videos including a different virtual characters, in which the virtual characters describe the eye motion extracted from the first ones. Since we are using the standard MPEG-4 FA we can conclude that we can use the same FAPs in order to animate different virtual faces.


visage
technologies
www.visage.com



4.5.3 Coding, Decoding and Visualization

In this section we explain the coding, decoding and visulization process used in order to animate virtual characters using the *Visage Technologies* library. This library provides a developed set of tools, to animate different virtual characters, coding and encoding the parameters which describe the animation in a very efficient way.

As we commented before, the animation of the virtual faces is one of the applications of eye tracking. Extracting the facial animation parameters, coding and recording them in a .fba file, we can show in a very simple way, the usefulness of our application in this field. In this way we can combine the eye animation track with other different animations tracks, for instance, an animation moving the mouth and the head, or a track animation of a person talking. Thus, we can realise the quantity of information that the movements of the eyes can provide.

Coding process

From [21] we cite: *The library visagefbaencoder implements the FBA Encoder functions: encoding Face and Body Animation Parameters (FBAPs) into an FBA file either frame-by-frame, or from an ASCII FAP and/or BAP files. If encoding from an ASCII FAP and/or BAP files, it requires an appropriate EPF (Encoder Parameter File, .epf) file. Sample ASCII FAP and BAP files with corresponding EPF files are provided too. When encoding frame-by-frame, FBAPs structure is sent to the encoder for each frame.*

Decoding process

The library visagefbadecoder provides the low-level, frame-by-frame decoding of an MPEG-4 FBA file. The Face and Body Animation Parameters (FBAPs), CodingParameters and the time stamp can be obtained for each frame after it is decoded.

Play process

The library visageFAPlayer and its FAPlayer and FbaAction classes, are central to visage-SDK. They allow to load one or more virtual characters and simultaneously play multiple animations on each character, animations coming from various sources (files, lip sync, speech synthesis, procedural animations, custom sources...). The animation sources are often provided by other libraries (such as VisageLipSync, Visagesapi5tts). The visageFAPlayer library implements a high-level MPEG-4 face and body animation player functionality. It reads a character model from a file and plays animations on this model. It can read one or more animation tracks from MPEG-4 FBA bitstreams, and accept other animation sources through the FbaAction abstract class interface. All animation tracks are played simultaneously. Multiple FAPlayers can be used simultaneously to animate multiple virtual characters.

Chapter 5

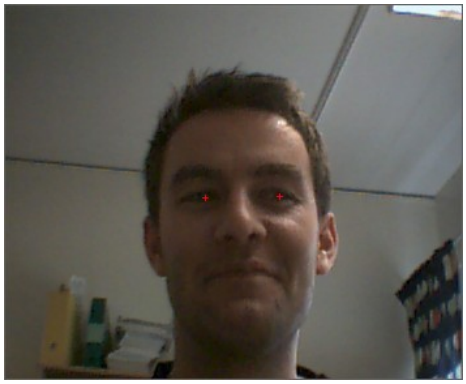
Results

5.1 Time Performance Analysis

The purpose of this section is to show some results related with the time performance analysis and the accuracy after applying our application to extract the eye motion from a video stream containing a human face. First, we process the input video, in order to extract the current frame. Once, we have an image containing the current frame, we apply the eye tracker application, where, initially we extract the location of the face, obtaining an image in which is stored only the information related with this feature. In this way, we avoid to work with the background of the video, reducing the quantity of information that we need to process. The next steps are detecting and extracting the eyes from the last image, and finally, processing these images in order to extract the position of the iris. The position of the iris is pointed out by placing crosses in the eye centers. Moreover, after processing the current frame and extracting the eye motion, we record this information in a new video stream, which is adjusted for the different frames of the input video stream, in which we have pointed out the position of the eyes. In this way, we can easily follow the movements of the eyes.

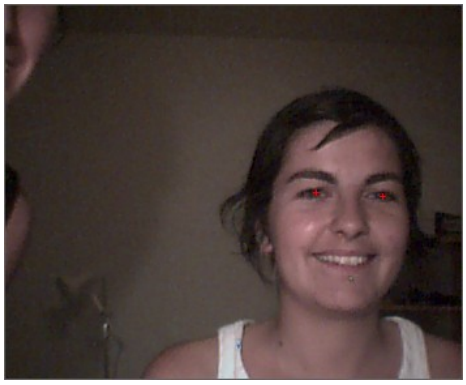
In this section, we include the results from the time performance analysis from each video. Moreover, we include one of the frames of each video, which the application records in order to show the results once we have extracted the eye motion.

5.1.1 Resolution 352x288



time (ms)	
Face detector	28
Eye tracker	33
Total	61

Frames per second (fps)	
Eye tracker	30.3
Whole application	16.39



time (ms)	
Face detector	25
Eye tracker	35
Total	60

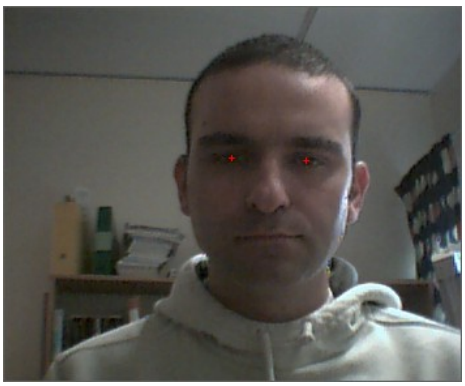
Frames per second (fps)	
Eye tracker	28.57
Whole application	16.67

time (ms)	
Face detector	27
Eye tracker	18
Total	45

Frames per second (fps)	
Eye tracker	55.59
Whole application	22.22

time (ms)	
Face detector	26
Eye tracker	24
Total	50

Frames per second (fps)	
Eye tracker	41.47
Whole application	20

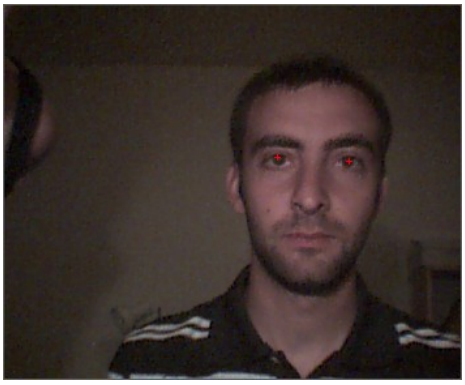


time (ms)	
Face detector	23
Eye tracker	42
Total	66

Frames per second (fps)	
Eye tracker	23.18
Whole application	15.15

time (ms)	
Face detector	24
Eye tracker	30
Total	54

Frames per second (fps)	
Eye tracker	33.33
Whole application	18.52

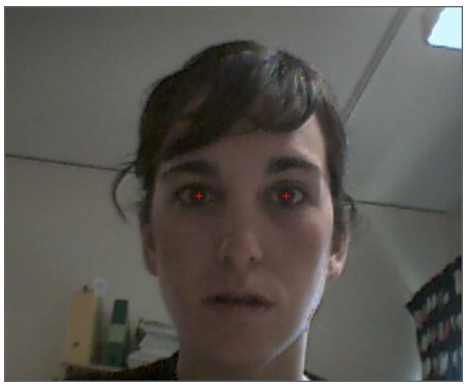


time (ms)	
Face detector	25
Eye tracker	30
Total	55

Frames per second (fps)	
Eye tracker	33.33
Whole application	18.18

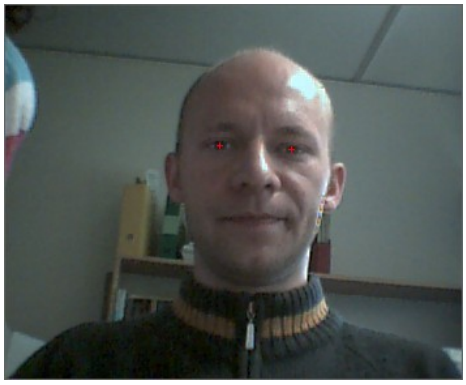
time (ms)	
Face detector	27
Eye tracker	27
Total	54

Frames per second (fps)	
Eye tracker	37.03
Whole application	18.51



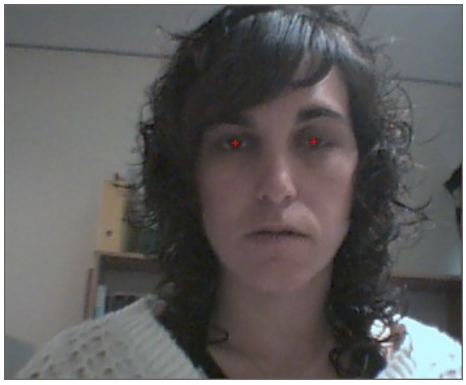
time (ms)	
Face detector	25
Eye tracker	65
Total	91

Frames per second (fps)	
Eye tracker	15.38
Whole application	10.99



time (ms)	
Face detector	23
Eye tracker	22
Total	45

Frames per second (fps)	
Eye tracker	45.45
Whole application	22.22



time (ms)	
Face detector	24
Eye tracker	38
Total	62

Frames per second (fps)	
Eye tracker	26.32
Whole application	16.13

time (ms)	
Face detector	21
Eye tracker	28
Total	50

Frames per second (fps)	
Eye tracker	35.71
Whole application	20

5.1.2 Resolution 177x144

time (ms)	
Face detector	6
Eye tracker	15
Total	22

Frames per second (fps)	
Eye tracker	66.67
Whole application	45.45

time (ms)	
Face detector	5
Eye tracker	15
Total	20

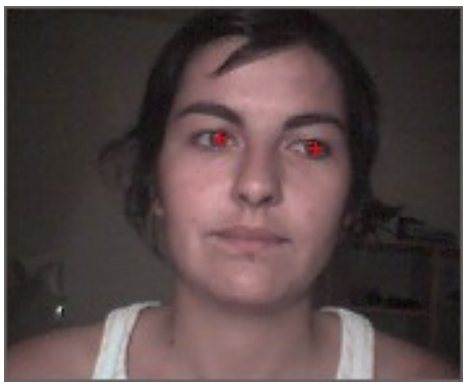
Frames per second (fps)	
Eye tracker	66.67
Whole application	50

time (ms)	
Face detector	7
Eye tracker	17
Total	24

Frames per second (fps)	
Eye tracker	58.82
Whole application	41.67

time (ms)	
Face detector	18
Eye tracker	30
Total	48

Frames per second (fps)	
Eye tracker	33.33
Whole application	20.83



time (ms)	
Face detector	20
Eye tracker	26
Total	47

Frames per second (fps)	
Eye tracker	38.46
Whole application	21.27

5.1.3 Resolution 640x480

time (ms)	
Face detector	82
Eye tracker	499
Total	582

Frames per second (fps)	
Eye tracker	2.01
Whole application	1.72

5.2 Comments to the results

As can be seen in the previous section, we have included some results from the application, where we show frames extracted from different video streams and also different resolution. In this section, we analyze these results in order to make some conclusions related to the time performance analysis of the method and also some conclusions related to the accuracy of the application. The results from the time performance analysis have been obtained using a Pentium(R) M, processor 2 GHz and 1GByte of RAM.

According to the values obtained for high resolution, $640x480$ [pixels] it is seen that the application is not suitable to work in real time. However, we can use the application in order to detect the eye motion offline. Anyway, the results we obtain working with high resolution are not very accurate. Considering the results obtained from the time performance analysis, and the conclusions related with accuracy, we can summarize that our application is not suitable to work with high resolution.

Concerning the results obtained working in lower resolutions, they are more accurate and the time consumption is reasonable. First, working with $352x258$ [pixels] the results are accurate enough, and the time analysis performance of the application, allows us working in real-time. As we commented before, we consider transmission in real time, when the number of frames per seconds (fps)

is higher than 15 fps.

Second, working with $176x144$ [pixels], the application is runner even faster. However, the extraction of the eye motion depends directly on the distance between the user and the camera and the size of the eyes. There are some cases in which the system can detect the eyes. However, it can not extract the information of the iris, since the eye images are pretty small, and we can not process them in order to extract the eye motion. In order to solve this problem, we can interpolate the image, obtaining an image with larger dimensions. In order to figure out this problem, we interpolate current image frame extracted from the camera, in order to get an image of $352x258$ [pixels]. Processing this image, we can extract the eye motion in the larger image. Afterwards, we can translate the results using the image of $352x258$ [pixels] to get the results of the eye motion of the image of $176x144$ [pixels]. Therefore, since we are working with a larger image to detect the position of the eyes the time consumption of the application increases, but still it is appropriate to work in real time.

Chapter 6

Conclusions and Improvements

In this study, a system capable of face tracking, eye detection and extraction, and finally iris position extraction using video streams containing a human face has been developed. The face tracking algorithm is based on the function *cvHaardetectObject*, which is a function already available in OpenCV, Open Source library written in C++. Therefore, from the frame extracted from the video stream, we extract an image only containing the human face. The detection and extraction of the eyes is based on edge detection. After, the iris center is determined applying different image preprocessing and region segmentation using edge features. The boundary of the iris is found by a circle detection algorithm. After we have located the iris, we can obtain the iris center and thus, the position of the iris in the current frame. Finally, the blinking is detected applying two different thresholds. In order to detect the blinking in a video sequence, two conditions must be fulfilled. The first one is related with the correlation coefficient. We make the correlation between a template containing an open eye and the current eye image extracted from the current frame. If the correlation coefficient is lower than a given threshold, this condition will be true. The second condition is related with the flow movement. By flow movement we understand the displacement up-down of the y-coordinate of the iris center. If this displacement is bigger than a given threshold, also this condition will be assumed true. Both thresholds must be specified by the user through the Graphical Interface.

The system works well using an inexpensive web camera. It performs in real-time when we work with low resolution ($352x288$ and $176x144$). Furthermore, the designed system is non-invasive. By non-invasive we mean that we do not need any external devices to run the application.

Once, we have extracted the eye motion from the video stream containing a human face, we apply this eye motion to a virtual face. The virtual face will move the eyes in the same way as the human face. To implement this application, we use MPEG4-FA to translate the eye motion extracted from the analysis of the video stream into the parameters needed to animate the faces.

These parameters are called Facial Animation Parameters. Since, MPEG4-FA is a standard, we can use the same FAPs in order to animate different virtual faces.

As is well known, the eyes give us a lot of information. An expression can be absolutely different depending on the information given by the eyes. Thus, tracking the eye motion, and applying this motion to a virtual face, we can improve the quality and quantity of the facial expressions that we can generate. Most of the models do not include tracking points in the eyes. Therefore, tracking the eye, we can improve the information given by a Facial Animation model. Using the software of *Visage Technologies*, we can use different animation in order to animate different parts of the face. For instance, we can use an animation to animate the mouth and the different visemes, a different animation to animate the movements of the eyes and another different animation to animate the movements of the head. Combining these animations we only need to take care that the same part of the face is not animated by different sources. Otherwise, we will not achieve the result which we expect due to the collision of two different animations controlling the same part of the face.

6.1 Limitations and Future Work

The system we have developed is not fully automatic; there are some adjustments that the user has to perform.

1. Blinking detection: the application will detect the blinking if two conditions are fulfilled. The first conditions is related with the correlation coefficient. If this correlation coefficient is lower than a given threshold, the first condition will be assumed true. The second condition is related with the flow movement. As we commented before, by flow movement we understand the displacement up-down of the y-coordinate of the iris center. If this displacement is lower than a given threshold, the second condition will be assumed true. Both threshold are set up by the user. The user has to decide the value of these thresholds in order to detect the eyes in the video sequence, by using two slides available into the graphical interface.

   To achieve an automatic blinking detection these threshold should be decided by the application. Since these thresholds are depending on a lot of factors (size of the video, lighting conditions, displacement of the iris, etc.), we point out this topic as a possible improvement in future work.

2. Temporal low pass filter: in order to process the eye motion extracted after applying the system, a temporal low pass filter can be applied. The user has to decide the size of the filter by using a list-choice available in the graphical interface.

Furthermore, there are other improvements that we would like to point out.

First, the model is not suitable for users wearing glasses. The glasses display strong edges and they can cause a strong reflection of the light. The result is that the eye estimation tracking process is not very accurate. We show some

results in the next pictures. In the first one, the reflection of the light causes non-correct tracking of the eyes.

Figure 6.1: User wearing glasses

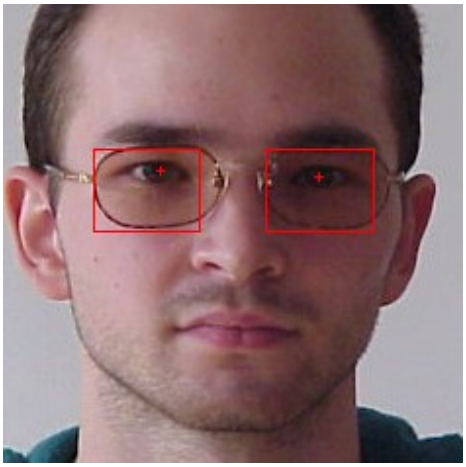


Figure 6.2: User wearing glasses

Second, our model is a 2D model, we can not deal with head rotation. Thus, we can only work with frontal faces. Fortunately, this problem can be solved, with the application of this 2D model into a Facial Animation 3D model. A 3D model can track the face dealing with different head rotations. After the face has been extracted from the video stream, we can apply the eye tracking model. An example of these models is Candide 3, which was described in a previous chapter.

Besides the extension from 2D model to 3D model, we can improve the time performance analysis, if we only apply the eye tracking application to the triangles included in the eye area. As is seen in the figure 6.3, Candide-3 is a

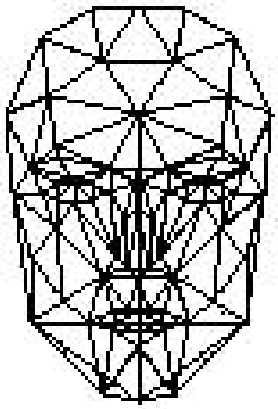


Figure 6.3: Facial Animation Model: Candide 3. Figure extracted from [**?**]

wire-frame composed by different triangles. If we only consider the information related with the triangles in the eye area, we can speed up the algorithm and improve the accuracy of the results. In this way, we avoid to extract the face from the current frame and the detection and extraction of the eyes.

6.2 Eye Tracking Applications

In this section we include some specific applications of the eye tracker systems defined in [22]:

1. Driver monitoring systems:
   Lack of visual attention plays and important role in road traffic accidents. So monitoring and controlling the eye states (open and close), and the gaze direction can be used as an accident countermeasure system. This kind of systems can also work, monitoring the blinking and the frequency and duration of the blinking in order to avoid the user to fall asleep.

2. Marketing and advertising:
   Eye tracking can get insight into how much visual attention the consumer pays in different forms of advertising in printed medias or visual systems.

3. Interactive information system:
   Eye Movement based informational display system, in which users use their eyes as a pointing device, such as a mouse pointer to navigate throughout the screen and select the different items.

4. Research related applications
   It can be used as a tool during by researches to find the relation between the eye movements and cognitive tasks. Eye movements can provide information related with a person's inner strategies for decision making, learning, motivation and memory. According to the results of this research it can be used to determinate the mental stage and psychology of the person.

5. Eye tracking can be used to help disabled people, who can move their eyes more efficiently than they can operate any other device.

6. Computer games:
   Eye motion can be used as an additional input in computer games.

Bibliography

[1] OpenCV. *Open Computer Vision Library Manual*. Open Source Technology Group(OSTG), Intel Coorporation. Available on http://sourceforge.net/projects/opencvlibrary.

[2] Harot, Flickner and Essa. *Detecting and tracking eyes using their physiological properties, dynamics and appearance*. GVU Center/College of Computing Georgia Institute of Technology, Atlanta, GA 30332-0280. Computer Vision Enhanced User Interfaces, IBM Almaden Research Center, San Jose, CA 95120.

[3] Singh and Papanikolopoulos. *Vision-Based Detection of Driver Fatigue*. Artificial Intelligence, Robotics and Vision Laboratory. Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

[4] Kashima, Hongo, Kato and Yamamoto. *A Robust Iris Detection Method of Facial and Eye Movement*. Gigu University, Department of Information Science 1-1, Yanagido, gifu Japan 501-1193. HOIP, Softopia & JST 4-1-7 Kagano, Ogaki, Gigu Japan 503-8569.

[5] Winer and Kiryati. *Virtual Gaze Redirection in Face Images*. Department of Electrical Engineering-Systems. Tel Aviv University, Tel Aviv 69978, Israel.

[6] J. Ahlberg. *Extraction and Coding Face Model Parameters*. Department of Electrical Engineering, Linköping University, Linköping, Sweden (1999).

[7] J. Ahlberg. *An updated parameterised face*. Department of Electrical Engineering, Linköping University, Linköping, Sweden (January 2001)

[8] G. Antunes Abrantes and F. Pereira. *The IST MPEG-4 Facial Animation System* Instituto Superior Técnico. Instituto de Telecomunicaciones, Lisboa, Portugal.

[9] Ivy Rose.*The anatomy of the Human Eye*IvyRose Ltd., Hampden House, Monument Business Park, Warpsgrove Lane, Chalgrove, Oxford (February 2006). Available on http://www.ivyrose.co.uk.

[10] MathWorks. *Image Processing Toolbox Manual*. Available on http://www.mathworks.com

[11] FLTK. *Fast Light Toolkit Manual*. Available on http://www.fltk.org

[12] OpenCV. *Intel Open Source Computer Vision Library Manual*. Available on http://sourceforge.net/projects/opencvlibrary.

[13] X.Deng, C.Chang and E.Brandle. *A new method for eye extraction from Facial Image*. Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore. University of Applied Sciences, Rpperwill, Switzerland.

[14] Z.Zhou and X.Geng. *Projection Functions for Eye Detection*. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China.

[15] F.Jie, Z.Zhou, H.Zhang and T.Chen. *Pose Invariant Face Recognition*. IEEE international Conference on Automatic Face and Gesture Recognition, Grenoble, France, 2000, pp.245-250.

[16] V.Vezhnevets and A.Degtiareva. *Robust and Accurate Eye Contour Extraction*. Graphics and Media Laboratory. Faculty of Computational Mathematics and Cybernetics, Moscow State University, Moscow, Russia.

[17] K.Grauman, M.Betke, J.Gips and G.Bradsky. *Communication via Eye Blinks - Detection and Duration Analysis in Real Time*. IEEE Conference on Computer Vision and Patter Recognition (December 2001).

[18] M.Chau and M.Betke. *Real Time Eye Tracking and Blink Detection with USB Cameras*. Boston University Computer Science Technical Report. Number 2005-12.

[19] S.Gachery and N. Magnenat. *Designing MPEG-4 Facial Animation Tables for Web Applications*. MIRALab, University of Geneva.

[20] I. Pandzic and R. Forchheimer. *MPEG-4 Facial Animation: The standard, Implementation and Applications*.

[21] Visage Technologies Library . *Visage Technologies AB*. Tröskaregatan 90. Linköping. Sweden.

[22] R.J.K. Jacob. *The use of eye movements in Human-Computer Interaction Techniques*. ACM Transactions on Information Systems (1991).

[23] P. Viola and M. Jones. *Rapid Object using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Patter Recognition 2001.

[24] P. Peer. *CVL Facial Database*. Computer Vision Laboratory, University of Ljubljana, Faculty of Computer and Information Science. Ljubljana, Slovenia (1999).

[25] PIE - Database of Human Faces by Terence Sim, Simon Baker and Maan Bsat.
Robotics Institute, Carnegie Mellon University (January 2001).

[26] Picture extracted from Image Coding Group web page. Department of Electrical Engineering, Linköping (September 2000). Available on http://www.bk.isy.liu.se/candide. University, Sweden.

[27] H. Kolb, E. Fernández and R. Nelson. *WebVision, The Organization of the Retina and Visual System*. University of Utah (October 2000). St Luke's Cataract and Laser Institute. Available on http://www.stlukeseye.com/Anatomy.asp.

[28] B. Fisher, S. Perkins, A. Walker and E. Wolfart. *Hypermedia Image Processing Reference*. Department of Artificial Intelligence, University of Edinburgh, United Kingdom. Available on http://www.cee.hw.ac.uk/hipr/html/sobel.html.

Appendix A

Functions used from OpenCV

A.1 CvHaarDetectObjects

The declaration of the function used to detect the face in the input image is the following:

$CvSeq*cvHaarDetectObjects(constIplImage*img, CvHidHaarClassifierCascade* cascade, CvMemStorage*storage, doublescale_factor = 1.1, intmin_neighbors = 3, intflags = 0)$;

img: image to detect objects in.

cascade: Haar classifier cascade in internal representation.

storage: Memory storage to store the resultant sequence of the object candidate rectangles.

scale_factor: The factor by which the research window is scaled between the subsequent scans, for example 1.1 means increasing window by 10%.

min_neighbors: Minimum number (minus 1) of neighbor rectangles that makes up an object. All the groups if a smaller number of rectangles that min_neighbors-1 are rejected. If min_neighbors is 0, the function does not any grouping at all and returns all the detected candidate rectangles, which may be useful if the user wants to apply a customized grouping procedure.

flags: Mode of operation. Currently the only flag that may be specified is CV_HAAR_DO_CANNY_PRUNING. If it is set, the function uses Canny edge detector to reject some image regions that contain to few or too much edges and thus can not contain the searched object. The particular threshold values are tuned for face detection and in this case the pruning speeds up the processing.

According to [23], *the detector is previously trained in order to be capable of*

processing images extremely rapidly and achieving high detection rates using a cascade of boosted classifiers for rapid object detection. This trainer is based in two concepts: on one hand the trainer classifies the objects by selecting a small number of important features from a large set, using AdaBost. Within an image the total number of Harr-like features is very large, far larger than the number of pixels. In order to ensure fast classification, the searching process must reject a large majority of the available features and focus on a small set of features. The detector is trained for detecting special face features. As was introduced by Tieu and Viola, who presented a simple modification of the AdaBoost procedure, each classifier returned can depend on only a single feature, thus each stage of the boosting process, which selects a new weak classifier, can be viewed as a feature selection process.

On the other hand, it combines more complex classifiers in a cascade *which allows background regions of the image to be quickly discarded. The cascade can be understood as an object with a specific focus-of-attention, ensuring that the object of interest is not included in the discarded regions. Those sub-window which are not rejected by the initial classifier are processed by the sequent classifiers, each one more complex than the last. The structure of the cascade detection process is essentially that of a degenerate decision tree, what we call a* cascade.

A.2 cvQuerryFrame

$IplImage * cvQueryFrame(CvCapture * capture)$;

Function that grabs and returns a frame from camera or file.

capture: video capturing structure.

The function cvQueryFrame grabs a frame from camera or video file, decompresses and returns it. This function is just a combination of cvGrabFrame and cvRetrieveFrame in one call. The returned image should not be released or modified by user.

A.3 cvGrabFrame

$intcvGrabFrame(CvCapture * capture)$;

Functions which grabs frame from camera or file.

capture: video capturing structure.

The function cvGrabFrame grabs the frame from camera or file. The grabbed frame is stored internally. The purpose of this function is to grab frame fast that is important for synchronization in case of reading from several cameras simultaneously. The grabbed frames are not exposed because they may be stored

in compressed format (as defined by camera/driver). To retrieve the grabbed frame, cvRetrieveFrame should be used.

A.4 cvRetrieveFrame

$IplImage * cvRetrieveFrame(CvCapture * capture)$;
capture: video capturing structure.

The function cvRetrieveFrame returns the pointer to the image grabbed with cvGrabFrame function. The returned image should not be released or modified by user.

A.5 cvCreateVideoWriter

$typedef struct CvVideoWriter CvVideoWriter$;
$CvVideoWriter * cvCreateVideoWriter$
$(const char* filename, int fourcc, double fps, CvSize frame_size, int is_color = 1)$;

filename: Name of the output video file.

fourcc: 4-character code of codec used to compress the frames. For example, CV_FOURCC('P','I','M','1') is MPEG-1 codec, CV_FOURCC('M','J','P','G') is motion-jpeg codec etc. Under Win32 it is possible to pass -1 in order to choose compression method and additional compression parameters from dialog.

fps: Framerate of the created video stream.

frame_size: Size of video frames.

is_color: If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

The function cvCreateVideoWriter creates video writer structure.

A.6 cvWriteFrame

$int cvWriteFrame(CvVideoWriter * writer, const IplImage * image)$;
The function cvWriteFrame writes/appends one frame to video file.

writer: video writer structure.

image: the written frame

A.7 cvGetCaptureProperty

Gets video capturing properties

$double cvGetCaptureProperty(CvCapture * capture, int property_id);$

capture: video capturing structure.

property_id: property identifier. Can be one of the following:

CV_CAP_PROP_POS_MSEC	film current position in milliseconds or video capture timestamp
CV_CAP_PROP_POS_FRAMES	0-based index of the frame to be decoded/captured next
CV_CAP_PROP_POS_AVI_RATIO	relative position of video file (0 - start of the film, 1 - end of the film)
CV_CAP_PROP_FRAME_WIDTH	width of frames in the video stream
CV_CAP_PROP_FRAME_HEIGHT	height of frames in the video stream
CV_CAP_PROP_FPS	frame rate
CV_CAP_PROP_FOURCC	4-character code of codec
CV_CAP_PROP_FRAME_COUNT	number of frames in video file

A.8 cvSetCaptureProperty

Sets video capturing properties

$int cvSetCaptureProperty(CvCapture* capture, int property_id, double value);$

capture: video capturing structure.

property_id: property identifier. Can be one of the following:

CV_CAP_PROP_POS_MSEC	film current position in milliseconds or video capture timestamp
CV_CAP_PROP_POS_FRAMES	0-based index of the frame to be decoded/captured next
CV_CAP_PROP_POS_AVI_RATIO	relative position of video file (0 - start of the film, 1 - end of the film)
CV_CAP_PROP_FRAME_WIDTH	width of frames in the video stream
CV_CAP_PROP_FRAME_HEIGHT	height of frames in the video stream
CV_CAP_PROP_FPS	frame rate
CV_CAP_PROP_FOURCC	4-character code of codec

A.9 cvFilter2D

Convolves image with the kernel.

$void cvFilter2D(const CvArr* src, CvArr* dst, const CvMat* kernel, CvPoint anchor = cvPoint(-1, -1));$

src: The source image.

dst: The destination image.

kernel: Convolution kernel, single-channel floating point matrix. If you want to apply different kernels to different channels, split the image using cvSplit into separate color planes and process them individually. anchor The anchor of the kernel that indicates the relative position of a filtered point within the kernel. The anchor should lie within the kernel. The special default value (-1,-1) means that it is at the kernel center.

The function cvFilter2D applies arbitrary linear filter to the image. In-place operation is supported. When the aperture is partially outside the image, the function interpolates outlier pixel values from the nearest pixels that is inside the image.

A.10 cvSmooth

Smooths the image in one of several ways

src: The source image.

dst: The destination image.

smoothtype: Type of the smoothing

1. CV_BLUR_NO_SCALE (simple blur with no scaling) - summation over a pixel $param1 \times param2$ neighborhood. If the neighborhood size may vary, one may precompute integral image with cvIntegral function.
2. CV_BILATERAL (bilateral filter) - applying bilateral $3x3$ filtering with color $sigma = param1$ and space $sigma = param2$.

   param1: The first parameter of smoothing operation.

   param2: The second parameter of smoothing operation. In case of simple scaled/non-scaled and Gaussian blur if param2 is zero, it is set to param1.

   param3: In case of Gaussian parameter this parameter may specify Gaussian sigma (standard deviation).

Appendix B

Description of the Graphical Interface

In this appendix we describe the graphical interface developed in order to provide to the user an easy use of the application. The graphical interface offers a set of buttons and choices to execute the application.

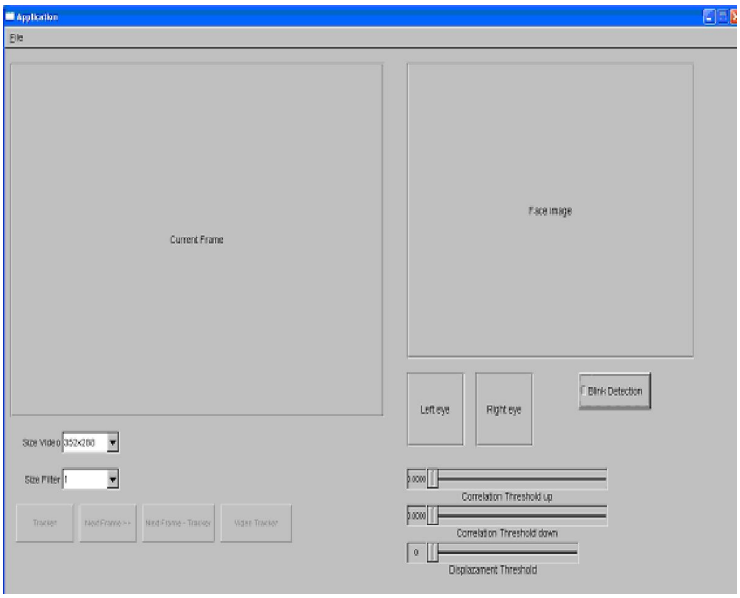


Figure B.1: Graphical Interface

Buttons

1. *Tracker*: Tracks the current frame of the video.
2. *Next Frame*: Gets the next frame of the video sequence.
3. *Next Frame Tracker*: Gets and tracks the next frame of the video sequence.

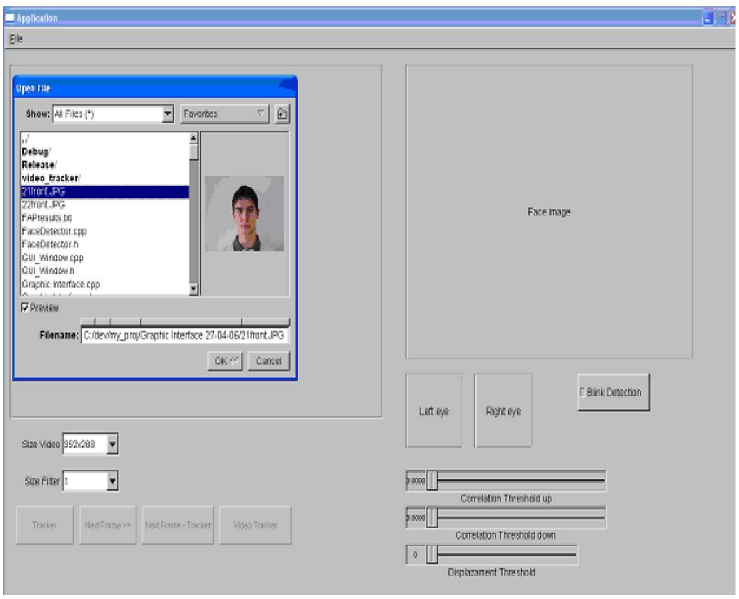


Figure B.2: Graphical Interface - File Chooser

4. *Video Tracker*: Tracks the whole video sequence and records a new video where in every frame the position of the eyes is pointed out by placing crosses in the eye centers.

5. *Blinking detection*: displays out when the blinking is detected.

Windows

1. *Current Frame*: Shows the current video frame.

2. *Face Image*: Shows the image extracted from the current video frame in which the background has been extracted.

3. *Left and right eye*: Shows both image containing the eyes.

In these images the position of the eyes is pointed out by placing crosses in the eye centers.

Slides item

The slides provide to the application the needed information in order to detect the blinking in the video sequence.

1. *Correlation Threshold up*: defines the upper threshold of the correlation coefficient.

2. *Correlation Threshold down*: defines the lower threshold of the correlation coefficient.

3. *Displacement*: defines the threshold of the flow movement.

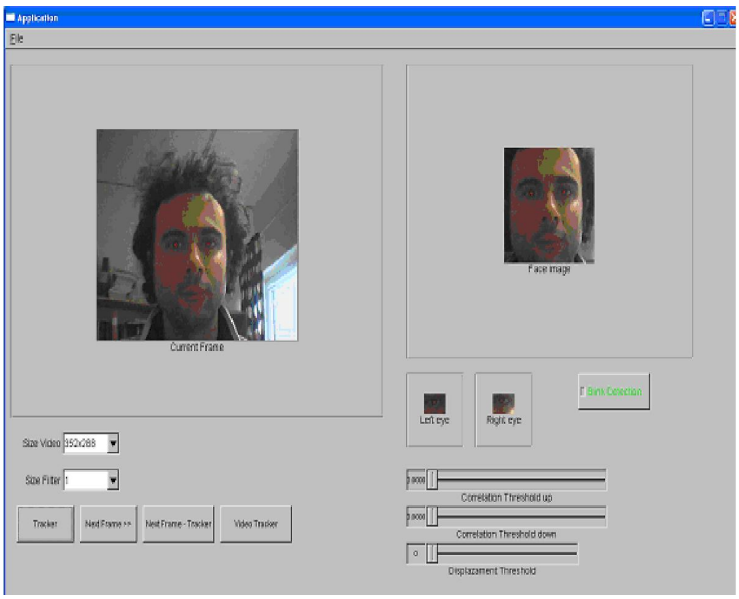


Figure B.3: Graphical Interface - Processing Image

Choice item

1. *Size Video*: allows the user to choose the size of the video to be processed.
2. *Size filter*: In case the user wishes to apply a temporal low pass filter during the processing of the video sequence, this choice item allows to select the size of this filter.

file chooser:

Menu item which allows to the user to select the video stream to be processed by the application.

Quit

Menu item which allows the user to quit the application.