

# Human-Computer Interface using a Web Camera

Loftie Ellis

University of Stellenbosch



Supervisor: Ben Herbst

Co-supervisor: Johan du Preez

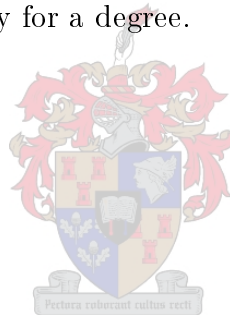
November 26, 2007

# Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature:

Date:



Copyright © 2007 Stellenbosch University

All rights reserved

# Abstract

In this thesis we present a human-computer interface (HCI) system for disabled persons using only a basic web camera. Mouse movements are simulated by small movements of the head, while clicks are simulated by eye blinks. In this study, a system capable of face tracking, eye detection (including iris detection), blink detection and finally skin detection and face recognition has been developed. A detection method based on Haar-like features are used to detect the face and eyes. Once the eyes have been detected, a support vector machines classifier is used to detect whether the eye is open or closed (for use in blink detection).

Skin detection is done using K-means clustering, while Eigenfaces is used for face recognition.

It is concluded that using a web camera as a human-computer interface can be a viable input method for the severely disabled.

# Opsomming

Die doel van hierdie studie is om 'n stelsel te ontwerp om 'n rekenaar te beheer deur slegs die gebruik van 'n basiese web kamera. Muis bewegings word gesimuleer deur klein bewegings van 'n persoon se oë en kop, terwyl 'n muis-kliek gesimuleer word deur middel van 'n oogknip. So 'n stelsel kan dan gebruik word om gestemde persone toegang tot die gebruik van 'n rekenaar te bied.

'n Metode gebaseer op Haar kenmerke word gebruik om die gesig en oë te vind, terwyl SVM's (support vector machines) gebruik word om 'n oogknip te identifiseer.

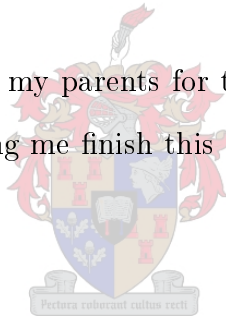
Om 'n spesifieke persoon te erken word gebruik gemaak van K-means om die gesig van die agtergrond te onderskei, met Eigenfaces om die klassifisering te bepaal.

Die gevolgtrekking word dan gemaak dat 'n internet kamera gebruik kan word as gebruikbare 'n koppelvlak tussen die rekenaar en 'n gestemde persoon.

# Acknowledgments

I would like to thank my supervisor Prof BM Herbst and fellow students for their support throughout the development of my Thesis.

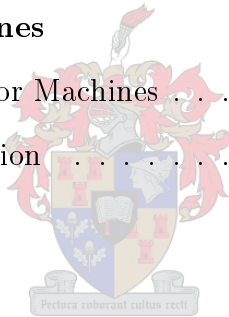
I would also like to thank my parents for their proofreading and continuous encouragement in helping me finish this report.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Description . . . . .	1
1.2	Solution . . . . .	2
1.3	Study Overview . . . . .	2
1.4	Related Research . . . . .	3
1.4.1	Gaze Tracking Systems . . . . .	4
1.4.2	Non-intrusive Techniques . . . . .	5
1.4.3	Research Summary . . . . .	7
<b>2</b>	<b>Image Processing Fundamentals</b>	<b>8</b>
2.1	Morphological Image Processing . . . . .	8
2.1.1	Erosion . . . . .	8
2.1.2	Dilation . . . . .	9
2.1.3	Closing . . . . .	10
2.1.4	Opening . . . . .	11
2.2	Color Space . . . . .	12

<i>CONTENTS</i>	vi
2.2.1 RGB Color Space . . . . .	12
2.2.2 HSV Color Space . . . . .	12
2.2.3 Grayscale images . . . . .	13
<b>3 Haar-like Features</b>	<b>14</b>
3.1 Fast Feature Computation . . . . .	16
3.2 Training . . . . .	18
3.3 Cascade of Classifiers . . . . .	19
3.3.1 Training the Cascade of Classifiers . . . . .	21
<b>4 Support Vector Machines</b>	<b>22</b>
4.1 Linear Support Vector Machines . . . . .	22
4.2 Non-linear classification . . . . .	25
<b>5 K-Means Clustering</b>	<b>29</b>
<b>6 Detection</b>	<b>32</b>
6.1 Face Detection . . . . .	32
6.2 Eye Detection . . . . .	33
6.2.1 Structure of the human eye . . . . .	33
6.2.2 Detection . . . . .	34
6.3 Blink Detection . . . . .	37
6.3.1 Blink Detection with Support Vector Machines . . . . .	38
6.4 Pupil Detection . . . . .	40



<i>CONTENTS</i>	vii
<b>7 Face Recognition</b>	<b>44</b>
7.1 Skin Detection . . . . .	44
7.2 Eigenfaces . . . . .	48
7.2.1 Calculating Eigenfaces . . . . .	49
7.2.2 Recognition Procedure . . . . .	52
<b>8 Implementation</b>	<b>53</b>
8.1 Tools and environments used . . . . .	53
8.2 Translating head movements to cursor movements . . . . .	54
<b>9 Results</b>	<b>58</b>
9.1 Time Performance Analysis . . . . .	58
9.2 Eye and Iris Detection Accuracy . . . . .	59
9.2.1 Description of the test databases . . . . .	59
9.2.2 Experimental Results . . . . .	60
9.3 Skin Detection . . . . .	63
9.4 Usability Analysis . . . . .	63
<b>10 Conclusions and Recommendations</b>	<b>67</b>
10.1 Future Work . . . . .	67
10.2 Suggested Applications . . . . .	68
10.3 Conclusions . . . . .	69
<b>A Code Structure</b>	<b>74</b>



# List of Figures

1.1	FreeGaze Gaze tracking system. . . . .	4
2.1	Erosion with a $3 \times 3$ square with origin at its center. . . . .	9
2.2	Dilation with a $3 \times 3$ square . . . . .	10
2.3	Closing with a $3 \times 3$ square. . . . .	10
2.4	Opening with a $3 \times 3$ square. . . . .	11
3.1	Extended set of Haar-like features . . . . .	15
3.2	Upright and 45 degrees rotated rectangle . . . . .	16
3.3	The first and second features selected by AdaBoost in the case of face detection. The first feature selected by AdaBoost focuses on the property that the region of the eyes are often darker than the region around the nose and cheeks, while the second feature relies on the property that the eyes are darker than the bridge of the nose. . . . .	19
3.4	Schematic depiction of the detection cascade . . . . .	20

4.1	The optimal hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ separates the classes with maximal margin. . . . .	23
4.2	Non-separable training set . . . . .	24
4.3	Non-linear classification . . . . .	26
4.4	Classes separated by a non-linear dividing line . . . . .	26
5.1	Demonstration of the K-Means Clustering Algorithm . . . . .	30
6.1	Frame image extracted from the web camera and detected face.	34
6.2	Structure of the human eye . . . . .	35
6.3	Detected Face Rectangle, expected eye area and detected eyes.	35
6.4	Positive eye training images . . . . .	36
6.5	Negative eye training images . . . . .	36
6.6	Image where a user's eyebrow was incorrectly identified as an eye. . . . .	36
6.7	Example Eye Images . . . . .	37
6.8	A vector space representation of a raster image array. . . . .	38
6.9	Iris/Pupil Detection . . . . .	40
7.1	Image regions for skin detection . . . . .	45
7.2	K-Means clustering with $k = 4$ . . . . .	47
7.3	Final skin map . . . . .	48
7.4	Mean image . . . . .	50
8.1	Translating head movements to cursor movements. . . . .	55

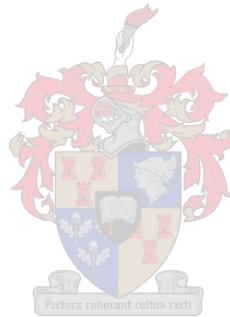
9.1 Relations and relative error. . . . . 60

9.2 Sample iris detection results . . . . . 61

9.3 Sample images where the iris was incorrectly identified . . . . 62

9.4 Sample skin detections. The last image is an example where  
bad lighting from the side resulted in poor skin detection. . . . 64

9.5 Sample images where the user is positioned above the screen.  
In the left image the user’s eyes are open but is looking down,  
while in the right image the eyes are closed. . . . . 65



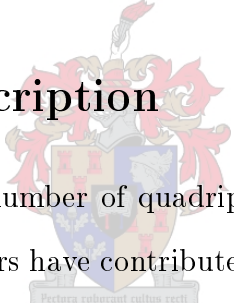
# List of Tables

3.1	Number of features inside a $24 \times 24$ window for each template.	16
9.1	Time analysis performance . . . . .	59
9.2	Results on the BioID database . . . . .	60
9.3	Results on the XM2VTS database . . . . .	62
A.1	Python source files (excluding modified libraries) . . . . .	74
A.2	Python libraries used . . . . .	75

# Chapter 1

## Introduction

### 1.1 Problem Description



There are currently a large number of quadriplegics<sup>1</sup> in the world. During the last two decades computers have contributed substantially to the quality of life of these people. However most of the interfaces allowing interaction with computers are expensive, intrusive and can be hard to obtain and use. Current available systems include EagleEyes, a system using electric probes attached to a user's forehead and FreeGaze, which utilizes a high definition infrared camera. The cost of such systems are typically in the range of tens of thousands of dollars.

Nowadays a basic USB web camera can be bought for under \$50, and many new personal computers come installed with web cameras. By lever-

---

<sup>1</sup>Quadriplegia is paralysis of both the arms and legs.

aging these devices and face detection techniques, it should be possible to create a system where the user can control the computer simply by looking into the camera.

## 1.2 Solution

We are proposing a system that uses only a standard current generation computer and entry level USB web camera to allow quadriplegics to interface and control the computer. The system we propose uses small movements of the head to control the mouse cursor, while blinks controls mouse clicks. (A left blink equals a left mouse click and vice versa.)

Such a system when coupled with other technology such as an on-screen keyboard can allow the user to effectively control nearly any task on the computer.

The efficiency of the head and eye-tracker is very important to control the computer in real time. The system must also be able to work reliably under different camera and lighting conditions.

## 1.3 Study Overview

The goal of this study is a proof of concept: to prove the viability of using an entry level USB web camera and a current generation computer to provide a reliable input method for quadriplegics to control the computer. It should be

possible to control the cursor using only movements of the head coupled with eye blinks. The system must also provide a fully automatic login system to recognize each user's settings without manual intervention.

This study will briefly discuss some basic image processing techniques in Chapter 2. Chapter 3 describes the workings of an object detection algorithms that we use to detect both the face and eyes. Chapter 4 describes the classification method of support vector machines which we use for blink detection. In Chapter 5 the K-Means clustering algorithm which we use for skin detection is detailed. Chapter 6 will then detail how these concepts are implemented to detect the face, eyes and blinks. Chapter 7 describes a method for automatically recognizing and remembering users of the system, while the system interface is briefly discussed in Chapter 8. The system is evaluated in Chapter 9 by examining its accuracy and usability. Finally recommendations on possible improvements on and applications for the developed system is discussed in Chapter 9.

## 1.4 Related Research

Several methods of eye detection, iris detection and eye tracking have been proposed in recent years. They can be classified mainly in several subcategories such as feature-based schemes, template matching, active contours etc.

Here we present some of these approaches.



Figure 1.1: FreeGaze Gaze tracking system.

### 1.4.1 Gaze Tracking Systems

Several very accurate gaze tracking systems using external devices have been developed for purposes such as marketing research or help for people with disabilities. These systems commonly use infrared cameras to create a corneal reflection, and then use triangulation to determine the fixation point. The setup of these systems can vary greatly; some use high resolution cameras placed in front of the computer screen (Figure 1.1), some are head-mounted and some use a different method such as electrodes attached to a user's head.

Since these gaze tracking systems use infrared cameras or other specialized hardware to track the eyes and we are interested in a non-invasive application, we will not consider this approach.



### 1.4.2 Non-intrusive Techniques

T. N. Bashkar et al. [6] describe a method for detecting blinks using frame differencing coupled with optical flow for eye blink detection. Frame differencing is used to quickly determine possible motion regions, and if any are detected, optical flow is computed within those regions. If the dominant motion is downwards in a pair of detected regions, their positions are noted. These would represent eye closure during a blink. The eye locations are computed as the bounding boxes of the regions where a blink is deemed to have occurred. Their method depends on finding dominant downward or upward motion in consecutive frames to detect the eyes. In our experience however the frame rate of some of the lower quality web cameras that we use is too low to reliably detect these motions. (The eye might be open in one frame, and completely closed in the next without any intermediate frames).

Singh et al. [7] describe an approach for real-time detection of driver fatigue. The system monitors the driver's eyes in order to detect micro-sleeps (short periods of sleep). As driver fatigue increases, the blink rate of the driver increases, and blinks tend to last longer. The system uses skin-color information in order to search for the face in the current frame. In order to locate the position of the pupil, they use gray scale model-matching. Since this approach has been developed in order to detect driver fatigue, it is not really accurate in situations with different gaze orientation or head rotation due to the fact that the method uses a template containing an open eye with the iris located in the center of the eyes.

Micheal Chau et al. [5] describe a method to detect and track blinks for use as a binary switch human-computer interface using USB cameras. The system uses a thresholded difference image of each frame and the previous frame to construct a binary image showing the regions of movement that occurred between the two frames. If a blink has occurred (and little other movement) then the system yields two connected components that are taken as the eye positions. Once the eye positions have been determined, template matching is used to track the position of the eyes allowing slight movements by the user. The correlation score between the online template of the user's eye and the current frame is also used for blink detection. As the user's eye is in the open state, very high correlation scores of about 0.85 to 1.0 are reported. As the user blinks, the scores fall to values of about 0.5 to 0.55. Scores below 0.45 indicate that the tracker has lost the location of the eye and must be reinitialized. The system works well when the movement of the user is constrained (blink detection accuracy as high as 95.3% is reported) but cannot effectively deal with head rotation or user movement.

Sandra Trejo Guerrero [8] describes a method to extract the eye motion from a video stream containing a human face and applying the eye motion into a virtual character. The system uses vertical edge detection using Sobel operators to find the position of eyes in the face image. For blink detection he uses a correlation coefficient generated by the eye image of the current frame and an open eye template. When the user is closing his/her eye correlation decreases. The first frame in the video sequence is used as the open eye

template, and thus it is assumed that in the first frame the eyes are open and with sight fixed into the camera.

### 1.4.3 Research Summary

The majority of methods dealing with iris or skin detection assumes high quality images under controlled lighting and/or little head movement from the user.

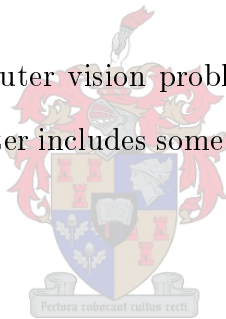
Our system follows a similar approach to Sandra Trejo Guerrero [8], in that we first locate the face in an image using the Haar-based detector, and then proceed to find the eyes and other features. Instead of using Sobel operators to locate the eyes we again employ the Haar-detector (along with some simple heuristics).

Most of the systems used for blink detection (such as [5] or [8]) use template matching and correlation to recognize eye blinks. Such systems perform well under controlled lighting and when the movement of the user is constrained, but cannot effectively deal with rapid head movements or inconsistent lighting (for example if the light source is from the side of the user). Since we expect the user to often make large and rapid head movements to control the mouse cursor, we chose a different approach based on support vector machines to detect blinks.

# Chapter 2

## Image Processing Fundamentals

Almost all solutions to computer vision problems involve image processing and manipulation. This chapter includes some of the basic concepts that will be used in this study.



### 2.1 Morphological Image Processing

#### 2.1.1 Erosion

Erosion is one of the two basic operators in mathematical morphology (the other being dilation) typically applied to binary images. The effect of erosion on a binary image is to erode away the boundaries of regions of foreground (white) pixels.

The erosion operator takes two pieces of data as inputs: The image to be eroded, and a set of coordinate points (known as a structuring element or

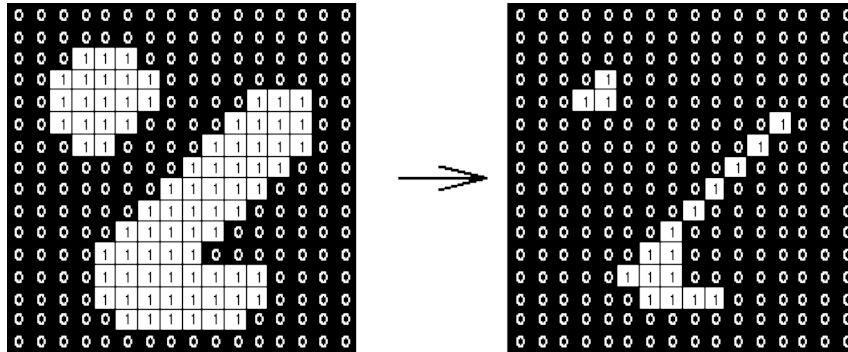


Figure 2.1: Erosion with a  $3 \times 3$  square with origin at its center.

kernel) that determines the price effect of the erosion on the input image.

Erosion is defined as follows:

Let  $A$  denote a binary image and  $B$  a structuring element

Let  $B_z$  denote the translation of  $B$  so that its origin is at  $z$ .

Then the erosion of  $A$  by  $B$  is simply the set of all points  $z$  such  $B_z$  is a subset of  $A$ :

$$A \ominus B = \{z | B_z \subset A\} \tag{2.1}$$

Erosion of an image with a  $3 \times 3$  square is shown in Figure 2.1.

### 2.1.2 Dilation

The basic effect of the dilation operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels.

The dilation of  $A$  by  $B$  is the set of all points  $z$  such that the intersection of  $B_z$  with  $A$  is non-empty:

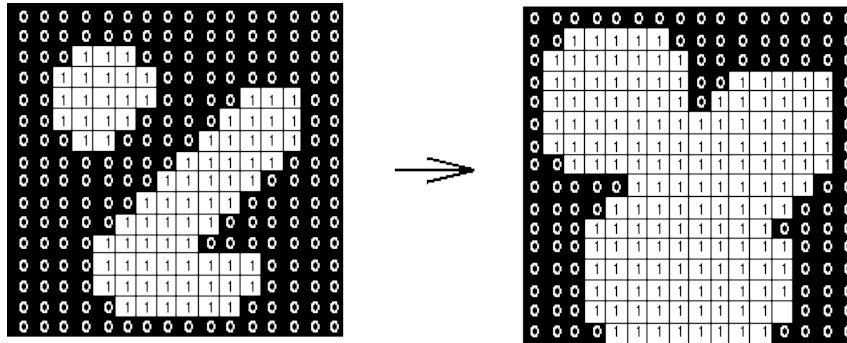


Figure 2.2: Dilation with a  $3 \times 3$  square

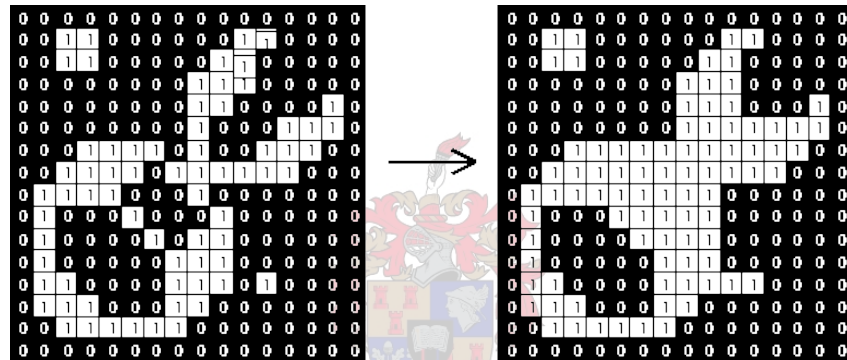


Figure 2.3: Closing with a  $3 \times 3$  square.

$$A \oplus B = \{z | B_z \cap A \neq \emptyset\} \tag{2.2}$$

Dilation of an image with a  $3 \times 3$  square is shown in Figure 2.2.

### 2.1.3 Closing

Closing is an important operator in the field of mathematical morphology. It is similar to dilation in that it tends to enlarge the boundaries of foreground regions in an image, but it is less destructive of the original boundary shape.

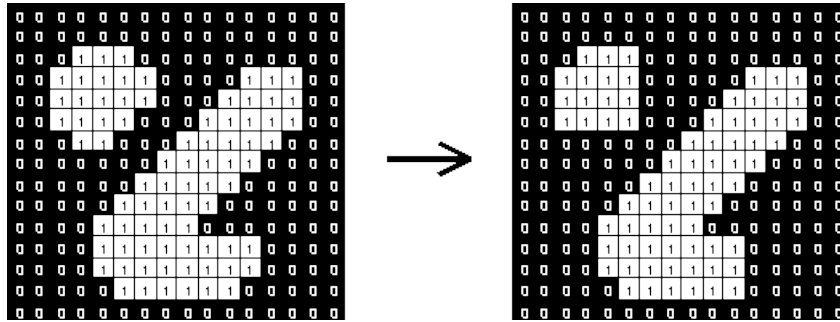


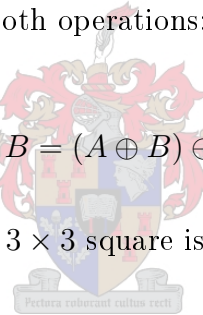
Figure 2.4: Opening with a  $3 \times 3$  square.

It is often used to fill in small background holes in images, e.g. ‘pepper noise’.

Closing is defined as simply a dilation followed by an erosion using the same structuring element for both operations:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.3}$$

Closing of an image with a  $3 \times 3$  square is shown in Figure 2.3.



### 2.1.4 Opening

Opening is the dual of closing, i.e. opening the foreground pixels with particular structuring element is equivalent to closing the background pixels with the same element.

Opening is defined as an erosion followed by a dilation using the same structuring element for both operations.

$$A \circ B = (A \ominus B) \oplus B \tag{2.4}$$

Opening of an image with a  $3 \times 3$  square is shown in Figure 2.4.

## 2.2 Color Space

A color model is an abstract mathematical model to describe the way colors can be represented as numbers for use with computers. Typically colors are represented as three components, such as with the RGB or HSV model.

For the skin detection information about the color of the user's face is processed.

### 2.2.1 RGB Color Space

Color images are usually represented in RGB. RGB is an additive model in which red, green and blue are combined in various ways to reproduce other colors. The color values are usually written as numbers in the range 0 to 255. Each component can then be stored as an 8-bit integer, thus up to  $2^{24}$  different colors can be generated.

### 2.2.2 HSV Color Space

The HSV model (also known as HSB) defines a color space in terms of three components:

1. Hue, the color type (such as red, blue or yellow)
2. Saturation, the intensity of the color




3. Value, the brightness of the color

The HSV color model is often used in skin detection, as the distribution of skin colors is better clustered in HSV than in RGB.

### 2.2.3 Grayscale images

A grayscale image is composed of one matrix in which the colors are shades of gray. Grayscale images are very common in image processing algorithms as they contain enough information for many tasks. Grayscale intensity is stored as an 8-bit integer giving 256 possible shades of gray from black (0) to white (255).

The conversion formula from RGB to grayscale images (as implemented in the PIL image library<sup>1</sup>) is:


$$L = \frac{299}{1000}R + \frac{587}{1000}G + \frac{114}{1000}B \quad (2.5)$$

where  $R$ ,  $G$ , and  $B$  are the red, green and blue components of the full color image.

In our algorithm both the face and eye detection is performed using grayscale images.

---

<sup>1</sup>Python Image Library. Note: All software implementation is done in Python.

# Chapter 3

## Haar-like Features

In this study we identify objects using simple Haar-like features. Such a system was initially proposed by Viola et al [3] and improved by Rainer Leinhardt [4], and this discussion is based in their work.

The detection algorithm slides a window at all scales over the input image, and at each stage the window is classified based on the value of simple features. For our implementation 14 feature sets are used as shown in Figure 3.1. Each feature consists of two or three “black” and “white” rectangles as shown in figure 3.1. The value of each feature is calculated as the weighted sum over each complete rectangle. The white and black regions are assigned different weights and of opposite signs. The absolute values of the weights are inversely proportional to the areas and can be computed as  $-w_{white}Area_{white} = w_{black}Area_{black}$ . Since only relative magnitudes are important we can set  $w_{white} = -1$  and  $w_{black} = \frac{Area_{white}}{Area_{black}}$ . For example the black

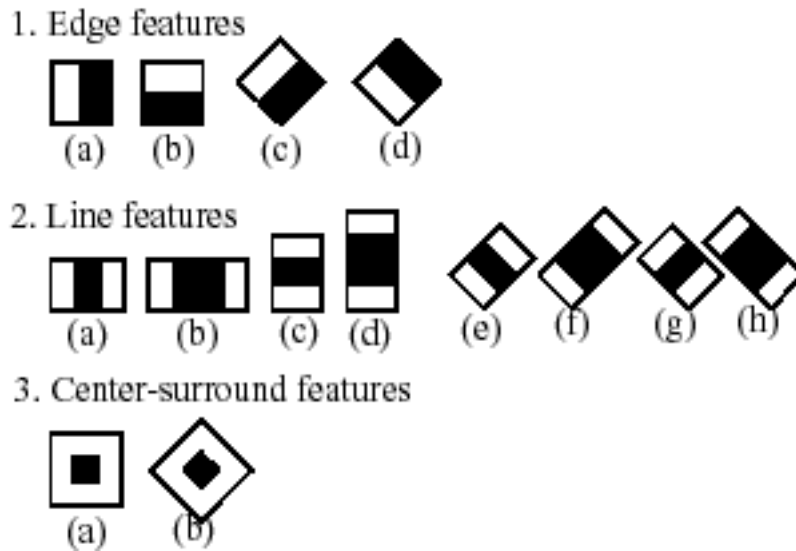


Figure 3.1: Extended set of Haar-like features

feature 3(a) in Figure 3.1 has a weight  $w_{black} = \frac{8}{1}$

Each of these features are scaled independently in both the vertical and horizontal direction in order to generate a rich, over-complete set of features. Each feature can therefore be described by the template, its coordinate relative to the search window origin and the scale factors of the feature. The number of features derived from each template is huge and computed as follows. Let  $W$  and  $w$  be the width of the image and the width of the feature respectively. Similarly let  $H$  and  $h$  be the height of the image and the feature set, as in Figure 3.2. The maximum scaling factors in the x and y direction can be defined as  $X = \frac{W}{w}$  and  $Y = \frac{H}{h}$ . An upright feature of size  $w * h$  then generates  $XY(W + 1 - w\frac{X+1}{2})(H + 1 - h\frac{Y+1}{2})$  features, while a  $45^\circ$  rotated feature generates  $XY[W + 1 - (w + h)\frac{X+1}{2}]H + 1 - (w + h)\frac{Y+1}{2}]$  features.

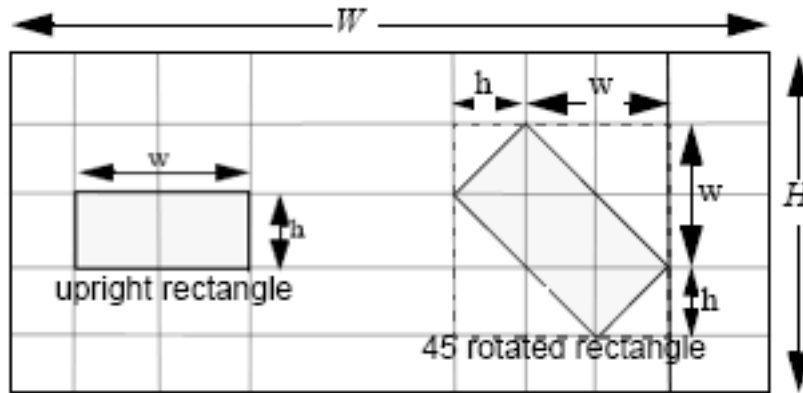


Figure 3.2: Upright and 45 degrees rotated rectangle

The total number of features for a small image of  $24 \times 24$  pixels is 117941, as calculated in Table 3.1.

Feature Template	w	h	X	Y	Number of features
1a; 1b	2; 1	1; 2	12; 24	24; 12	43200
1c; 1d	2; 1	1; 2	8	8	8464
2a; 2c	3; 1	1; 3	8; 24	24; 8	27600
2b; 2d	4; 1	1; 4	6; 24	24; 6	20736
2e; 2g	3; 1	1; 3	6	6	4356
2f; 2h	4; 1	1; 4	4	4	3600
3a	3	3	8	8	8464
3b	3	3	3	3	1521
Total					117941

Table 3.1: Number of features inside a  $24 \times 24$  window for each template.

### 3.1 Fast Feature Computation

All the features can be computed rapidly by means of two auxiliary images called integral images. For upright rectangles the integral image at location

$x, y$  contains the sum of the pixels values above and to the left of  $x, y$  inclusive:

$$\iota(x, y) = \sum_{x', y'} i(x', y') \quad (3.1)$$

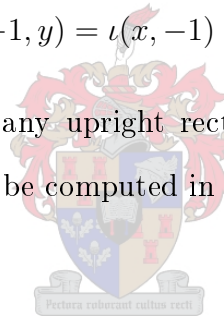
where  $\iota(x, y)$  is the integral image and  $i(x, y)$  is the original image. The integral image can be computed in one pass over the original image by means of

$$\iota(x, y) = \iota(x, y - 1) + \iota(x - 1, y) + i(x, y) - \iota(x - 1, y - 1) \quad (3.2)$$

with

$$\iota(-1, y) = \iota(x, -1) = 0. \quad (3.3)$$

From this the pixel sum of any upright rectangle  $r = \{(x, y), x_0 \leq x \leq x_0 + w, y_0 \leq y \leq y_0 + h\}$  can be computed in four array references using the corners of the rectangle:



$$\text{RecSum}(r) = \iota(x_0 + w, y_0 + h) - \iota(x_0 + w, y_0) - \iota(x_0, y_0 + h) + \iota(x_0, y_0) \quad (3.4)$$

For rotated rectangles a separate “rotated” integral image must be used.

Using the integral image, any feature can be completed in constant time for any sized image with just a few array references, regardless of the feature size or position.

## 3.2 Training

Given a set of positive and negative images, a variant of AdaBoost was used to train the classification function. AdaBoost is a powerful machine learning algorithm that can train a strong classifier based on a large set of weak classifiers by re-weighting the training samples. Since there are well over 100,000 rectangle features associated with each image sub-window it is prohibitively expensive to compute the complete set, but Viola and Jones [3] have shown that a small number of these features can be combined to form an effective classifier.

To find this small subset of features, the weak training algorithm is designed to select the single rectangle feature that best separates the positive and negative training samples. For each feature, the weak trainer determines the optimum threshold classification function such that the minimum number of examples are misclassified. A weak classifier  $h_j(x)$  therefore consists of a feature  $f_j$ , a threshold  $\Theta_j$  and a parity  $p_j$ :

$$h_j(x) = \begin{cases} 1 & p_j f_j < p_j \Theta_j \\ 0 & \text{otherwise} \end{cases} \quad (3.5)$$

The parity  $p_j$  is used to indicate the direction of the inequality sign and can be  $\pm 1$ .

Here  $x$  is a  $24 \times 24$  pixel sub-window of an image. For the task of face detection, the first two features selected by AdaBoost are shown in Figure 3.3.

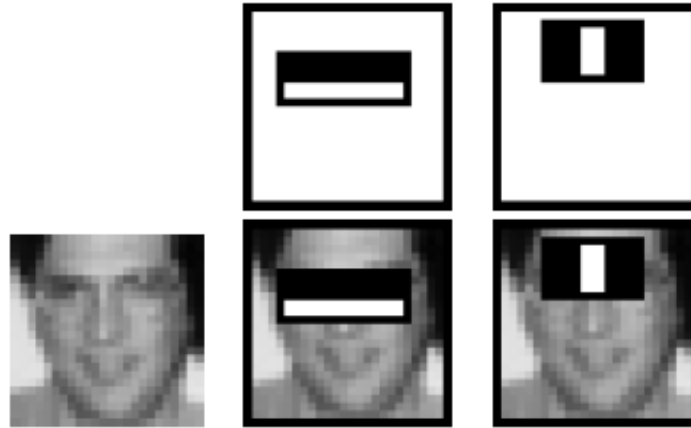


Figure 3.3: The first and second features selected by AdaBoost in the case of face detection. The first feature selected by AdaBoost focuses on the property that the region of the eyes are often darker than the region around the nose and cheeks, while the second feature relies on the property that the eyes are darker than the bridge of the nose.

### 3.3 Cascade of Classifiers

To achieve the desired detection performance a large number of features must be used which can have an adverse effect on computation time. In order therefore to reduce the computation time and still achieve good detection performance, Viola and Jones use a sequence of increasingly complex classifiers called a cascade. At each stage the classifier is trained to detect almost all objects of interest while rejecting a certain fraction of non-object features. (The threshold of a boosted classifier can be adjusted so that the false negative rate is close to zero). With this setup simpler classifiers are used to reject the majority of the sub-windows before more complex classifiers (that require the evaluation of more features) are called upon to achieve

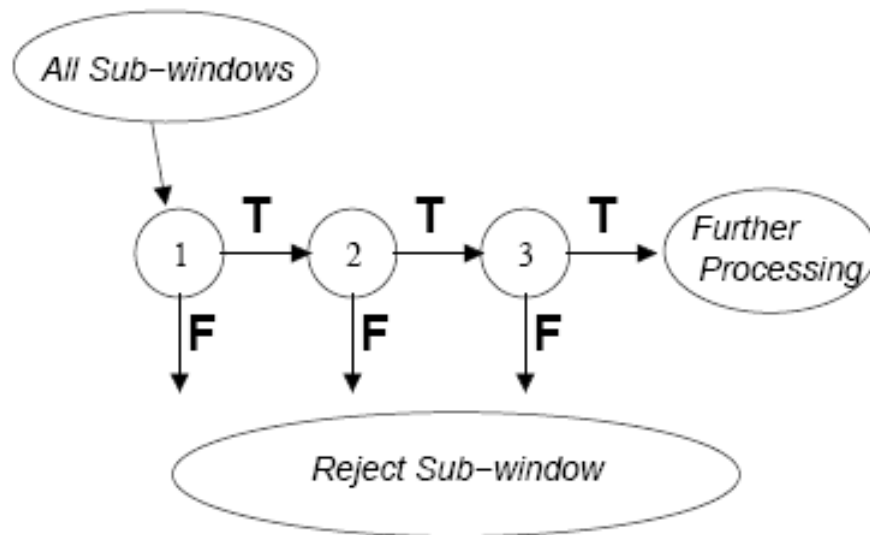


Figure 3.4: Schematic depiction of the detection cascade

low false positive rates.

The overall form of the detection process is that of a degenerate decision tree. A positive result at any stage leads to evaluation of the next stage, whereas a negative outcome at any point leads to the immediate rejection of the sub-window. The structure of the cascade reflects the fact that within any single image an overwhelming majority of sub-windows are negative. The cascade therefore attempts to reject as many of the negatives as possible at the earliest possible time. Positive instances will trigger the evaluation of every classifier in the cascade but this is a rare event. Figure 3.4 illustrates the cascade.



### 3.3.1 Training the Cascade of Classifiers

Training the cascade involves a trade off between the number of features per classifier and the computation time. In most cases classifiers with more features will achieve higher detection rates and lower false positive rates, but also require more time to compute. Determining the optimal balance between the number of classifier stages, the number of features in each stage and the threshold of each stage can be difficult.

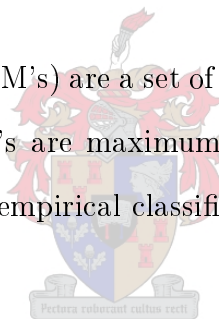
In practice a simple framework is used to produce an effective yet efficient classifier. Since each stage in the cascade reduces the false positive rate and decreases the detection rate, a target is selected for the minimum reduction in false positives and the maximum decrease in detection. Each stage is then trained by adding features until the target rates, determined by testing the detector on a validation set, are met. Stages are added until the overall target for detection and false positives is met.

The complete face detection cascade (as implemented in the OpenCV [2] library) has 24 stages, where each stage has up to 200 features.

# Chapter 4

## Support Vector Machines

Support Vector Machines (SVM's) are a set of supervised learning algorithms used for classification. SVM's are maximum margin classifiers, since they simultaneously minimize the empirical classification error and maximize the geometric margin.



### 4.1 Linear Support Vector Machines

Given data that can belong to two different classes we need to find a way to separate the classes. For example in the case of linearly separable data set, the optimal hyperplane is the one giving the largest margin of separation between the classes. (see Figure 4.1)

A relatively small number of data points lie exactly on the margin, and these points are called the support vectors (indicated as the shaded vectors

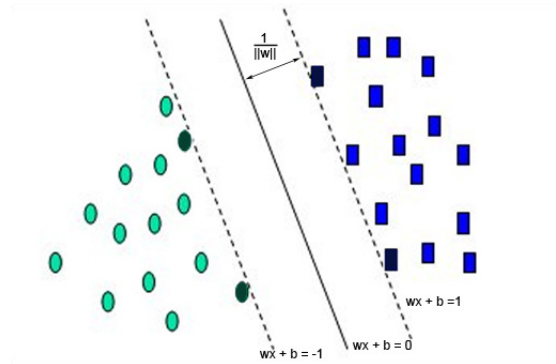


Figure 4.1: The optimal hyperplane  $\mathbf{w} \cdot \mathbf{x} + b = 0$  separates the classes with maximal margin.

in figure 4.1). The optimal hyperplane is completely determined by these points.

The support vectors lie on the two hyperplanes  $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$ . The margin is therefore  $\frac{2}{\|\mathbf{w}\|}$ . This margin is maximized by minimizing  $\|\mathbf{w}\|$ , subject to the constraint that there can be no points between them. To exclude data points, we need to ensure that for all  $i$  either

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \text{ or}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1$$

This can be written as:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, i = 1, \dots, n \quad (4.1)$$

where  $y_i = 1$  or  $y_i = -1$  depending on  $\mathbf{x}_i$ 's class.

Most data sets are not linearly separable (see for example Figure 4.2), so this minimization problem is modified to allow for misclassified data points.

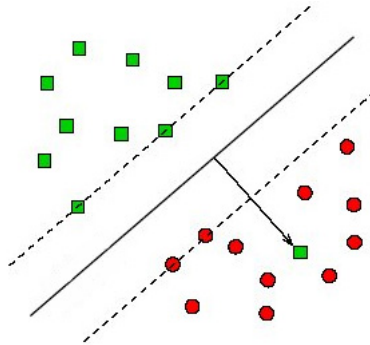


Figure 4.2: Non-separable training set

If there exists no hyperplane that can split the "yes" and "no" examples, the Soft Margin method will choose a hyperplane that splits the examples as cleanly as possible, while still maximizing the distance to the nearest cleanly split examples. This method introduces error variables  $\xi_i \geq 0$ , and the minimization is then as follows:



$$\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 + \kappa \sum_{i=1}^n \xi_i \quad (4.2)$$

$$\text{Subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \dots, n \quad (4.3)$$

This results in a hyperplane that maximizes the margin for correctly classified data, while minimizing the sum of deviations,  $\xi_i$

This is a quadratic programming problem that can be solved using stan-

standard optimization techniques. In practice, the dual form of this optimization problems is using to a set of Lagrange multipliers  $\alpha$ .

The dual form of the classification can be shown to be [1]:

$$\text{Maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (4.4)$$

$$\text{subject to } \sum_{i=1}^n y_i \alpha_i = 0 \text{ and } 0 \leq \alpha \leq \kappa, i = 1, \dots, n \quad (4.5)$$

The error variables  $\xi_i$  constrains the range of Lagrange multipliers  $\alpha_i$  from 0 to  $\kappa$ . This penalty parameter,  $\kappa$  aims to prevent outliers from affecting the optimal hyperplane.

This means that  $\mathbf{w}$  is a linear combination of the support vectors:

$\mathbf{w} = \sum_{i=1}^n y_i \alpha_i \mathbf{x}_i$  where  $\alpha_i = 0$  for any training vector that is not a support vector.

The decision function of the optimal hyperplane is thus:

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^n y_i \alpha_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right) \quad (4.6)$$

## 4.2 Non-linear classification

If the classes are separated by a non-linear boundary as shown in Figure 4.4, a non-linear dividing line is required for classification.

Rather than fitting nonlinear curves to separate the classes, the training vectors are mapped into a higher (maybe infinite) dimensional space by a

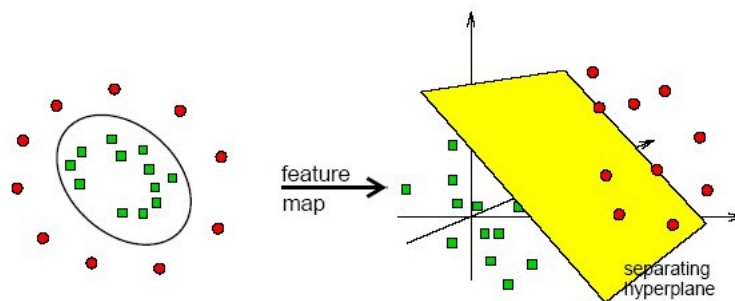


Figure 4.3: Non-linear classification

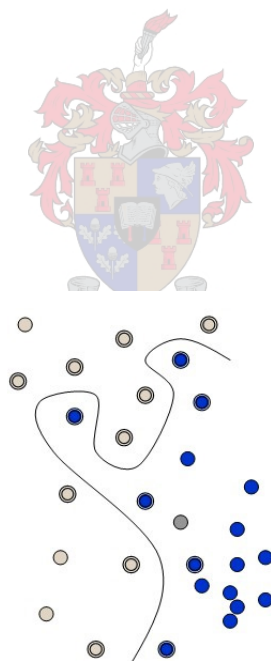


Figure 4.4: Classes separated by a non-linear dividing line

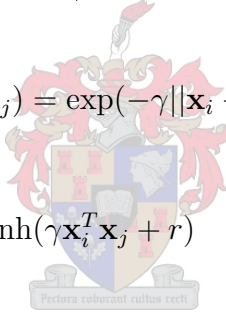
function  $\phi$ . Then SVM then finds a linear separating hyperplane with the maximal margin in this higher dimensional space (see Figure 4.3). Furthermore  $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i^T)\phi(\mathbf{x}_j)$  is called the kernel function.

Many kernel mapping functions can be used, but a few have been found to work well in a wide variety of applications:

- linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \cdot \mathbf{x}_j$
- polynomial:  $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \gamma > 0$
- radial basis functions (RBF's), for example:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$$

- sigmoid:  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$



here  $\gamma$ ,  $r$  and  $d$  are kernel parameters.

The resulting algorithm for the optimization problem is formally similar, except that every dot product is replaced by a non-linear kernel function:

$$\text{Maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (4.7)$$

$$\text{subject to } \sum_{i=1}^n y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq \kappa, i = 1, \dots, n \quad (4.8)$$

the decision function is now:

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^n y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b\right) \quad (4.9)$$





# Chapter 5

## K-Means Clustering

K-Means is an algorithm to cluster objects into  $k$  partitions based on certain attributes. The goal of the K-Means algorithm is to minimize the squared error function,

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2 \quad (5.1)$$

where  $\|x_i^{(j)} - c_j\|^2$  is a chosen distance measure between data point  $x_i^{(j)}$  (the squares in Figure 5.1) and the cluster mean  $c_j$  (the circles in Figure 5.1),  $k$  is the number of clusters and  $n$  the number of feature vectors.

The algorithm is composed of the following steps:

1. Place  $k$  points<sup>1</sup> into the space represented by the objects that are being clustered. These points can be chosen either at random or using some heuristic data, and represent the initial cluster mean. These points

---

<sup>1</sup>Note that one has to decide in advance the number of partitions into which the objects are clustered.

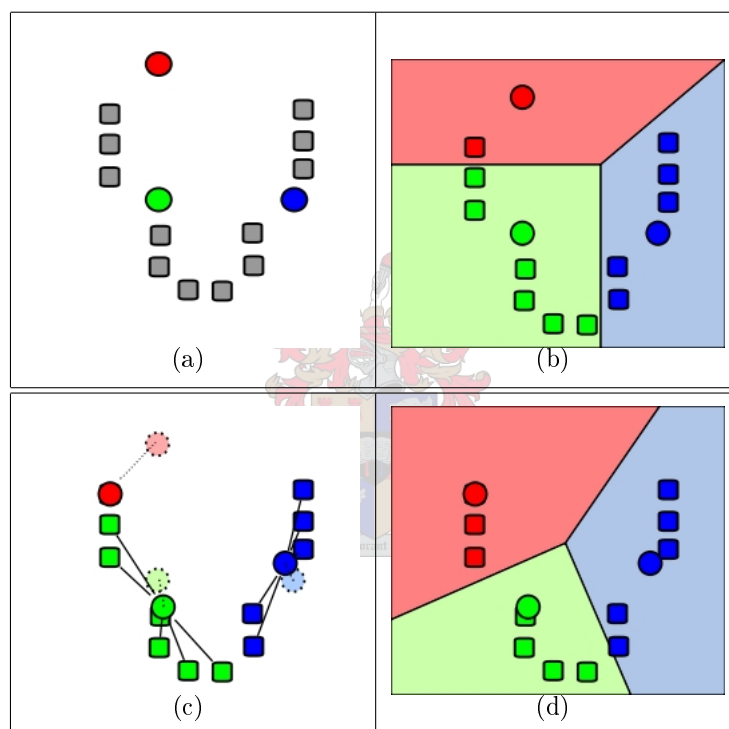
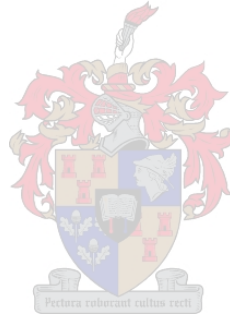


Figure 5.1: Demonstration of the K-Means Clustering Algorithm

are represented by the colored circles in Figure 5.1 (a), while the gray squares represent the objects that are being clustered.

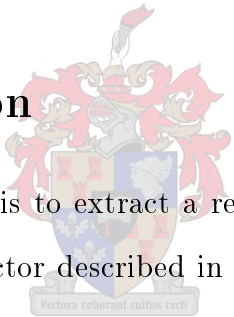
2. Points are then associated to the cluster that has the closest mean. (Figure 5.1 b)
3. When all points have been assigned, the means are moved to the mean of their respective clusters. (Figure 5.1 c)
4. Steps 2 & 3 are repeated until the centroids no longer move or a suitable level of convergence has been reached. (Figure 5.1 d)



# Chapter 6

## Detection

### 6.1 Face Detection



The first step in our system is to extract a region of the image containing the face using the Haar-detector described in section 3. Such a detector is already designed and implemented in OpenCV with the function `cvHaarDetectObjects`. The OpenCV [2] library also contains default training data for use with face detection. According to the OpenCV Manual [2],

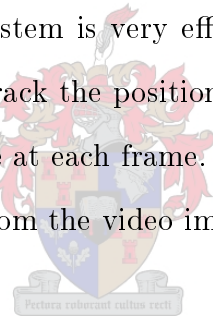
*The function `cvHaarDetectObjects` finds rectangular regions in the given image that are likely to contain objects the cascade has been trained for, and returns those regions as a sequence of rectangles. The function scans the image several times at different scales. Each time it considers overlapping regions in the image and applies the classifiers to the regions using `cvRunHaarClassifierCascade`. It may also apply some heuristics to reduce the num-*

ber of analyzed regions, such as Canny pruning. After it has proceeded and collected the candidate rectangles (regions that passed the classifier cascade), it groups them and returns a sequence of average rectangles for each large enough group.

Using this function, we obtain the coordinates of each face in the image. If multiple faces are detected, then we must choose the face that belongs to the actual user in control of the system. To do this, we assume that the actual user in control of the system is most likely the user closest to the camera, and therefore the largest face in the image.

Since the face detection system is very efficient there is no need for an actual tracking algorithm to track the position of a person's face, instead it detects the position of the face at each frame.

The detected face region from the video image is shown in Figure 6.1.



## 6.2 Eye Detection

After the face has been identified in the image the location of the eyes must be found.

### 6.2.1 Structure of the human eye

In order to detect the position of the eyes, we process the information given by three basic parts of the eyes: (see Figure 6.2)

1. The Pupil is located in the center of each eye, and generally appears

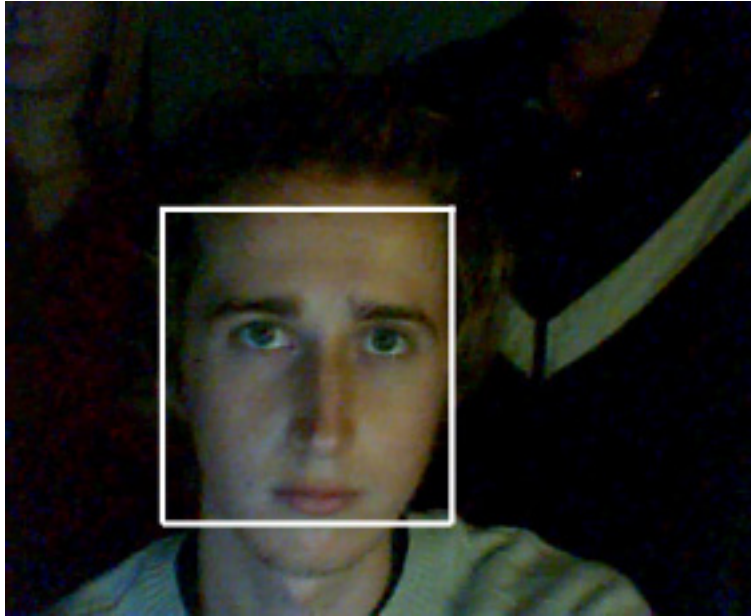


Figure 6.1: Frame image extracted from the web camera and detected face.

to be the dark center of the eye.

2. The Iris is the colored part of the human eye.
3. The Sclera is the white outer part of the eye.

### 6.2.2 Detection

Since the location of the face is known, the search area to find eyes can be narrowed down to just those areas where you would expect eyes on a face.

After running several tests this region was found to be the rectangle

$$r = \{(x, y), \frac{w}{9} \leq x \leq \frac{8w}{9}, \frac{h}{4} \leq y \leq \frac{h}{2}\}, \quad (6.1)$$

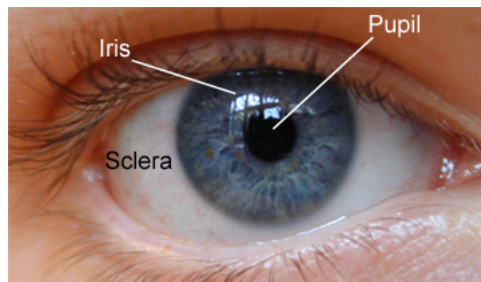


Figure 6.2: Structure of the human eye

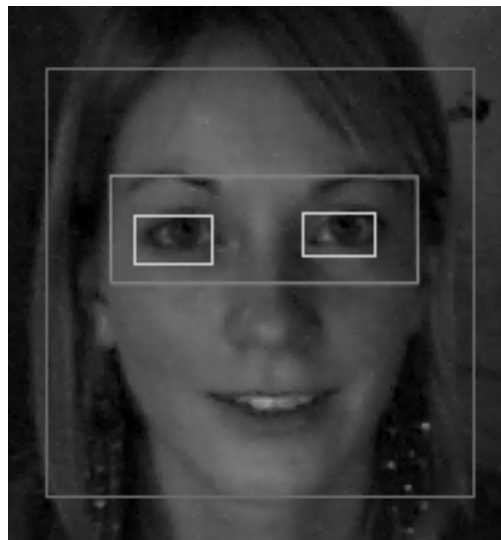


Figure 6.3: Detected Face Rectangle, expected eye area and detected eyes.

where  $w$  is the width and  $h$  the height of the detected face image, see Figure 6.3.

To find the eyes in this region, the Haar detector from Chapter 3 is again used. The Haar detector was trained with 1800 positive and 2000 negative samples. The positive eye samples were extracted from the xm2vts [9] database, a set of normalized face images all manually aligned so that the eyes in each image are at the same coordinates. Since the eye detector has

to find both open and closed eyes (for example a mouse drag operation can require the user to close his one eye and then move his head), the positive eye samples included both open and closed eyes. Negative samples were obtained by taking several samples around the face area away from the eyes. Sample positive and negative images for training are shown below.

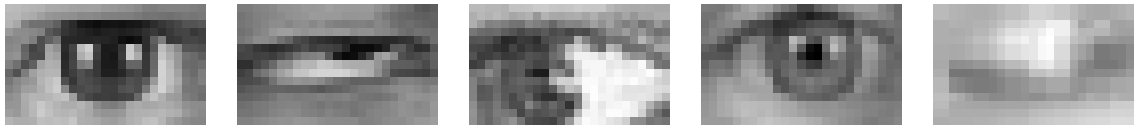


Figure 6.4: Positive eye training images



Figure 6.5: Negative eye training images

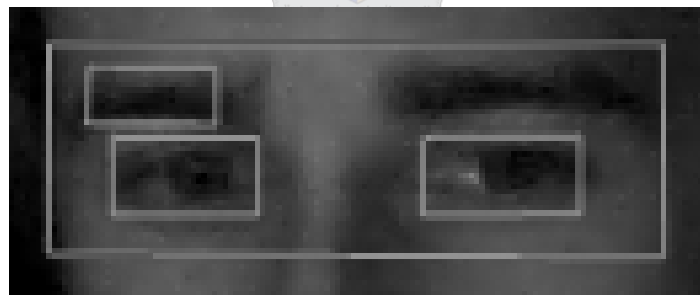


Figure 6.6: Image where a user's eyebrow was incorrectly identified as an eye.

**Dealing with false positives:** If a user has dark eyebrows and his/her head is positioned in such a way that the eyebrows falls inside the search



window for the eyes, then the Haar detector will often wrongly detect the eyebrow as an eye (Figure 6.6). In such a situation the correct location can easily be found by inspecting the coordinates returned by the Haar detector and choosing the location with the highest y coordinates (the lowest result in the image).

If the detector does not return a result for both eyes (for example if the users head is turned too far) then the previous known positions of the eyes are used.

### 6.3 Blink Detection

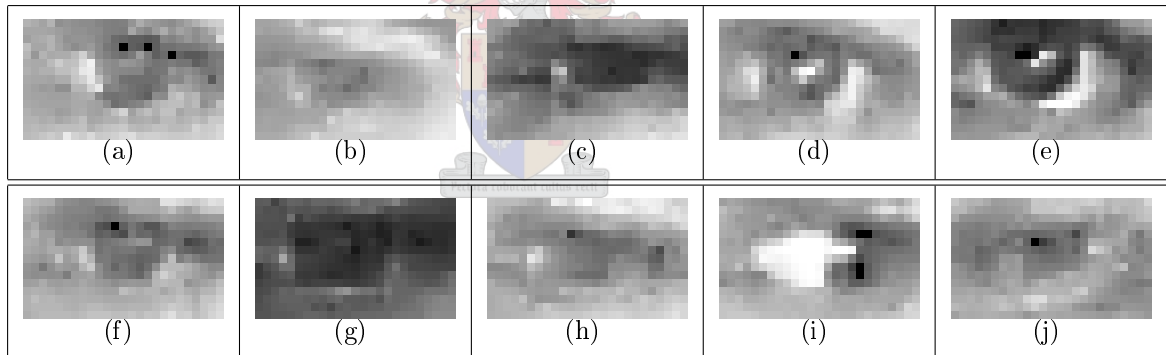


Figure 6.7: Example Eye Images

In order to simulate mouse clicks we include the process of blink detection. In this way a left eye blink equals a left button click and similarly a right eye blink equals a right mouse click. Furthermore the system must be able to distinguish between voluntary and involuntary blinks.

Since the blink information is used to control mouse-click events it is

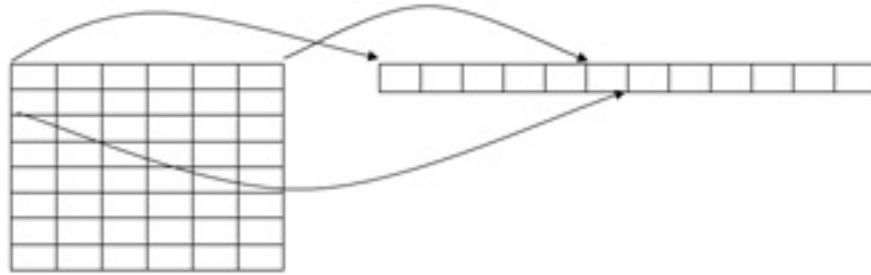


Figure 6.8: A vector space representation of a raster image array.

important that the detection system be robust, especially against false positives. If the system detects a blink then a mouse click is generated, and any false clicks can cause havoc to a system. The blink detection is complicated by the fact that the user may freely move his head during using the system, even while keeping one or both eyes closed. This can cause the eye images to look blurry and in some situations can cause the lighting on one or both eyes to change significantly, especially on lower quality web cameras. For example, all the images in Figure 6.7 are all from the same camera, taken just a few seconds apart.

### 6.3.1 Blink Detection with Support Vector Machines

To detect the blinks we use support vector machines described in section 4.

For the kernel function we use the RBF (Radial Basis Function). Unlike the linear kernel, the RBF kernel nonlinearly maps samples into a higher dimensional space so it can handle the case when the relation between class labels and attributes is nonlinear. The RBF kernel also has less hyperpa-

rameters than the polynomial kernel which can make it easier to select the model parameters, and it has less numerical difficulties. (In polynomial kernels kernel values may go to infinity or zero in contrast to the RBF kernel where  $0 < K_{ij} \leq 1$  .)

For the training of a default classifier we use 1000 images of each class (eyes open and closed). The images were taken from different cameras and in different lighting conditions to be a representative sample of what can be expected in real-world use. The system was also setup such that it could be retrained for each user using images from the user's own eyes.

To train the SVM classifier, the training images must be converted to a vector space representation. This can be done in several ways, such as using edge features or histogram analysis, but the simplest way is to use the pixel values of the grayscale image directly. To do this, all positive and negative training images must have the same dimensions, in our case all images are resized to 25x15 pixels. The image matrix is then 'flatten' to obtain a vector space representation (see Figure 6.8) and values are normalize to lie within  $(0, 1)$ .

**Parameter selection:** RBF kernels need two parameters,  $C$  and  $\gamma$  (see Section 4.2). Optimal values are not available a priori. In order to find suitable values, we used a grid-search on  $C$  and  $\gamma$  using cross-validation. (It only has to be done once, and can be done efficiently since SVM's are fast to train). Exponentially growing sequences of  $C$  and  $\gamma$  in the range  $2^{-5}$  to

$2^8$  (for example,  $C = 2^{-5}, 2^{-4}, \dots, 2^8, \gamma = 2^{-5}, 2^{-4}, \dots, 2^8$ ) were tried, and the pair with the best cross-validation accuracy are picked. For the default training data in our system the selected parameters are:  $C = 2$  and  $\gamma = 2^{-3}$

To reduce the number of false classifications that can result from blurry images due to fast head movement, we only log a blink if the classifier returns the same result for at least 50ms or the previous three consecutive frames, whichever is longer. If the classifier detects both eyes as closed, then it is assumed to be an involuntary blink and no mouse click is generated.

## 6.4 Pupil Detection

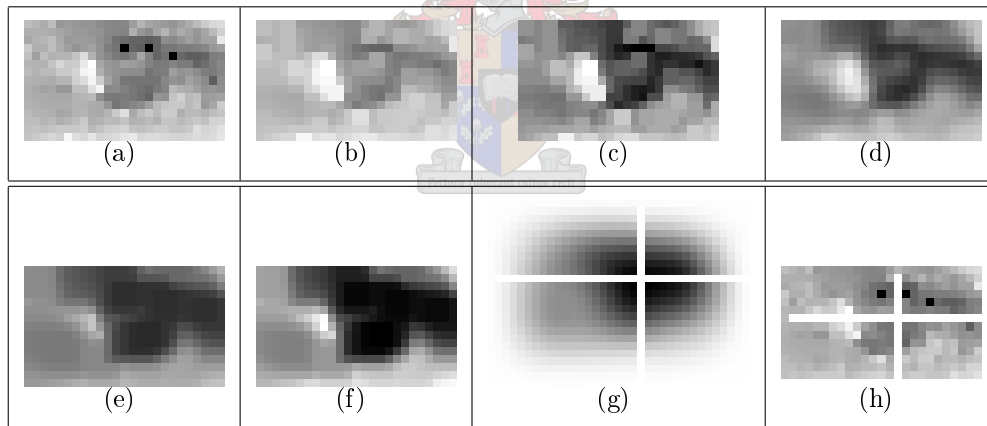


Figure 6.9: Iris/Pupil Detection

The Haar detector returns an approximate location for the eyes, but for greater accuracy the locations of the pupils are computed. In the low resolution eye images it is often impossible to differentiate between the pupil and the iris (for example Figure 6.9a), and in such cases finding the location of

the pupil is the same as finding the center of the iris. To locate the iris in an image two of its properties are used, namely that the iris is roughly circular in form, and generally darker than the surrounding eye (the sclera) and skin.

Iris Detection is performed in the following steps:

1. The eye region is extracted using the results from the Haar detector (section 3). A typical image is shown in (Figure 6.9 a). The resulting image typically has a low resolution (around  $25 \times 15$  for a  $352x \times 288$  resolution camera if the user is at a distance of around 50cm from the camera) and can have several artifacts (single pixels that are much darker or lighter than it should be), usually as a result of poor lighting.
2. To remove the dark pixel artifacts, a maximizing filter is applied to the image. The maximizing filter replaces a value with the maximum value in the neighborhood of a  $2 \times 2$  matrix and serves to remove dark pixel artifacts. (Figure 6.9 b)
3. After the maximizing filter the image is normalized for all pixel values to lie within  $[0, 255]$  (Figure 6.9 c)

$$\text{Image} = \frac{255(\text{Image} - \min(\text{Image}))}{\max(\text{Image} - \min(\text{Image}))} \quad (6.2)$$

4. Next a Gaussian Smoothing filter is applied to the image in order to reduce noise and other naturally occurring jitter from the image. The Gaussian filter is chosen as it has several desired features for smoothing

(e.g. it is linearly separable, and it produces no sharp edges). The filter is applied by convolving the image with the Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (6.3)$$

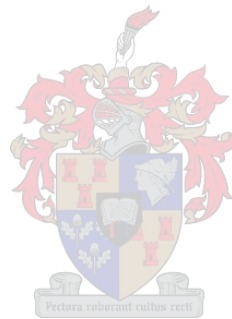
$\sigma$  is the standard deviation of the distribution, and for our implementation was chosen as  $\sigma = 1$ . The resulting image after the Gaussian blur is shown in Figure 6.9 (d).

5. Next the image is passed through a minimum filter. The minimum filter replaces a value with the minimum value in the neighborhood of a  $2 \times 2$  matrix and has the effect of suppressing the light areas in the image. (Figure 6.9 e)
6. After the minimum filter the image is again normalized (Figure 6.9 f)
7. Finally the image is convolved with a function, emphasizing the circular regions of dark pixels.

$$W(x, y, r_0) = \frac{\sin\left(\frac{x^2+y^2}{r_0}\right)}{\frac{(x^2+y^2)}{r_0}} \quad (6.4)$$

This function emphasizes the circular regions at the center of the convolution. The parameter  $r_0$  controls the radius of the expected circular area and is chosen according to the expected iris size. The output and minimum values after the convolution is shown in Figure 6.9 (g).

The final result is shown in Figure 6.9 (h).



# Chapter 7

## Face Recognition

For the system to be usable as an alternative means of communication, it is necessary to automatically recognize the person in front of the camera. This enables a person to login to a computer by just sitting in front of the camera.

Face recognition is done in two steps: First the skin part of a face must be extracted from the image to remove all elements that are not relevant to recognition, such as the background and hair/clothes. After the skin map has been constructed, the second step is the recognition, which is done using eigenfaces.

### 7.1 Skin Detection

The simplest method of skin detection is to define skin color to have a certain range or values in some coordinates of a color space. This can easily be



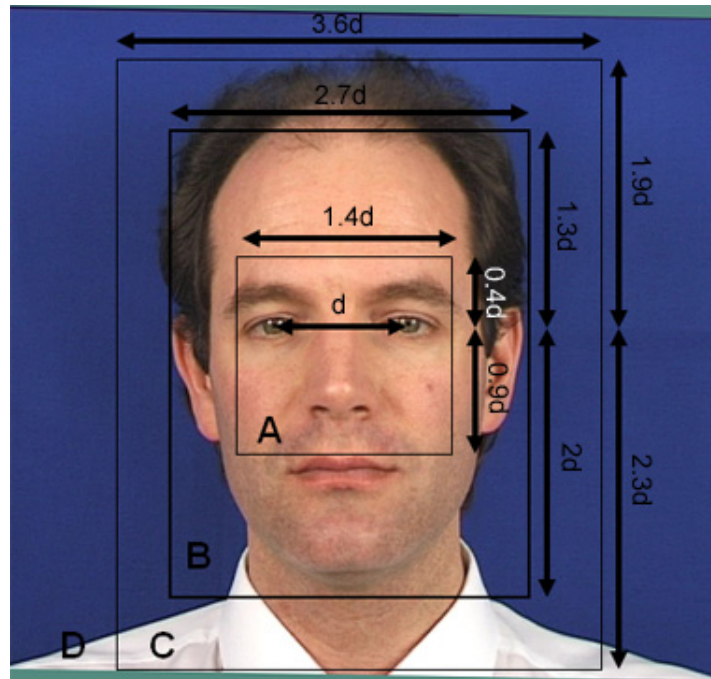


Figure 7.1: Image regions for skin detection

implemented as a look-up table or as threshold values or by using parametric methods such as multimodal Gaussian mixtures.

For our system however such methods have proved to be very ineffective. The color representation from the cheap web cameras can be very poor, especially in non-ideal lighting conditions. To identify any areas as skin under these conditions the threshold for skin detection must be set very low, resulting in an unacceptably large number of false positive detections.

Since the location of a face in the image is known in our system, a different method for skin detection based on K-Means clustering can be used.

The first step in constructing the skin map with K-Means is to rotate the

face to an upright position, using the previously detected pupil positions. Then, using the distance between the eyes as a size guide, the face image is broken up into 4 different segments as shown in Figure 7.1.

- Region A is the rectangle  $A = \{(x, y), i_{xl} - 0.2d \leq x \leq i_{xr} + 0.2d, i_y - 0.4d \leq y \leq i_y + 0.9d\}$ , where  $d$  is the distance between the eyes,  $i_y$  the y-coordinate of the iris and  $i_x$  the x-coordinate. This is the region that we can reasonably assume will include mostly skin-colored pixels. (it also contains other features such as eyebrows, but that is assumed to be the minority)

- Region B is the rectangle  $B = \{(x, y), i_{xl} - 0.85d \leq x \leq i_{xr} + 0.85d, i_y - 1.3d \leq y \leq i_y + 2d\}$

This is the region where we expect the skin boundary to lie.

- Region C is the rectangle  $C = \{(x, y), i_{xl} - 1.3d \leq x \leq i_{xr} + 1.3d, i_y - 1.9d \leq y \leq i_y + 2.3d\}$

This area will usually contain only background or hair features, but it can still contain skin depending on the shape of the face.

- Region D is the rest of the image, and is the area of the image that gets discarded when we proceed with skin detection.

The K-Means algorithm is then run on the image in section C. Once the image is clustered we still have to identify the specific cluster that represents

the skin color. To identify this we look for the biggest cluster in region A, where we can expect mostly skin.

The result from the K-Means with  $k = 4$  is shown below.



Figure 7.2: K-Means clustering with  $k = 4$

**Choosing  $k$**  A weakness of the K-Means algorithm is that you have to specify the number of clusters,  $k$ . If the image has a uniform background color then the image can be broken up into 3 different colors (Skin, hair and background). For a non-uniform background however it can be better to set  $k$  to a higher value.

To choose the best value for  $k$  automatically, we calculate the K-Means algorithm for values of  $k$  from 3 to 8, and then look at the ratio of skin color detected in region A (where we expect mostly skin) to skin detected in region between C and B (where we expect to find little skin). We then choose  $k$  to be the value that produces the highest ratio of skin in region A vs region (C-B).

**Finalizing the skin map** After the K-Means clustering, we choose only the skin areas on the image to construct a binary skin map. To remove any small clusters or pixels we perform a morphological closing on the image. Next we remove all areas that are not connected to the main area, region A, and finally a Gaussian blur with  $\sigma = 3$  is applied on the image to remove any rough edges that might have been incorrectly identified.

The final skin map is shown below:

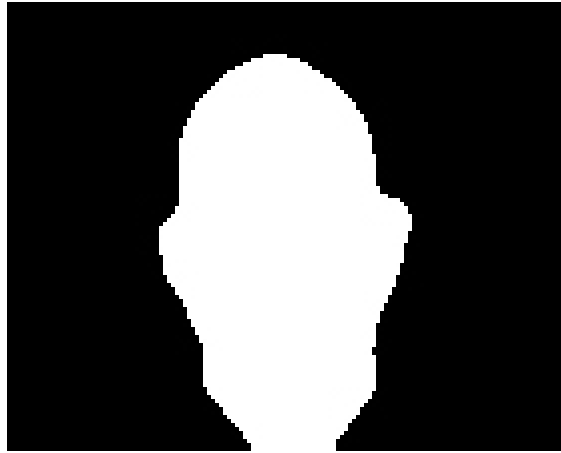


Figure 7.3: Final skin map

## 7.2 Eigenfaces

Once the skin has been extracted from the face image, we use Eigenfaces to recognize the user.

### 7.2.1 Calculating Eigenfaces

**Definitions:** An  $N \times N$  matrix  $A$  is said to have an eigenvector  $X$ , and corresponding eigenvalue  $\lambda$  if

$$AX = \lambda X$$

A matrix is called symmetric if it is equal to its transpose,

$$A = A^T$$

it is termed orthogonal if its transpose equals its inverse,

$$A^T A = AA^T = I$$

finally, a real matrix is called normal if it commutes with its transpose,

$$AA^T = A^T A$$

**Calculating Eigenfaces:** The first step is to obtain a set  $S$  with  $M$  face images.

$$S = \{\Gamma_1, \Gamma_1, \Gamma_1, \dots, \Gamma_M\} \quad (7.1)$$

After the set have been obtained the mean image  $\Psi$  is calculated.

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n \quad (7.2)$$

The difference between the input image and the mean image is then:

$$\Phi_i = \Gamma_i - \Psi. \quad (7.3)$$



Figure 7.4: Mean image

This set of large vectors is then subject to principal component analysis, which identifies a set of  $M$  orthonormal vectors,  $u_n$ , which best describes the distribution of the data. The  $k$ th vector,  $u_k$ , is chosen such that

$$\lambda_k = \frac{1}{M} \sum_{n=1}^M (u_k^T \Phi_n)^2 \quad (7.4)$$

is a maximum, subject to

$$u_l^T u_k = \delta_{lk} = \begin{cases} 1 & \text{if } l = k \\ 0 & \text{otherwise} \end{cases} . \quad (7.5)$$

The vectors  $u_k$  and scalars  $\lambda_k$  are the eigenvectors and eigenvalues, respec-

tively of the covariance matrix

$$C = \frac{1}{M} \sum_{n=1}^M \Phi_n \Phi_n^T = AA^T. \quad (7.6)$$

where the matrix  $A = [\Phi_1 \Phi_2 \dots \Phi_M]$ . The covariance matrix  $C$  however is  $N^2 \times N^2$  real symmetric matrix, and computing  $N^2$  eigenvectors is not a computationally feasible task for typical image sizes.

If the number of data points in the image space is less than the dimension of the space ( $M < N^2$ ), there will be only  $M - 1$  rather than  $N^2$  meaningful eigenvectors, and the remaining eigenvectors will have associated eigenvalues of zero. In this case we can solve for the  $N^2$  dimensional eigenvectors by first solving the eigenvectors of an  $M \times M$  matrix and then taking appropriate linear combinations of the face images  $\Phi_i$ .

Consider the eigenvectors  $v_i$  of  $A^T A$  such that  $A^T A v_i = u_i v_i$ .

Multiplying both sides by  $A$  we have  $AA^T A v_i = u_i A v_i$  from which we see that  $A v_i$  are the eigenvectors of  $C = AA^T$ . (This computation however is mathematically unstable and not used in practice, see Golub et al [15].)

Following these analysis, we construct the  $M \times M$  matrix  $L = A^T A$ , where  $L_{mn} = \Phi_m^T \Phi_n$ , and find the  $M$  eigenvectors,  $v_l$ , of  $L$ . The eigenfaces is then

$$u_l = \sum_{k=1}^M v_{lk} \Phi_k, l = 1, \dots, M. \quad (7.7)$$

### 7.2.2 Recognition Procedure

To recognize an unknown face image, the new face is first transformed into its eigenface components. For our system we chose the number of eigenfaces  $M$  equal to the number of images in the training set.

A new face image  $\Gamma$  is transformed into its eigenface components by a simple operation,

$$w_k = u_k^T(\Gamma - \Psi) \text{ for } k = 1, \dots, M \quad (7.8)$$

The weights form a feature vector,

$$\Omega^T = [w_1 \ w_2 \ \dots \ w_M] \quad (7.9)$$

We now determine which face class provides the best description for the input image by minimizing the Euclidean distance

$$\varepsilon_k = \|\Omega - \Omega_k\|^2. \quad (7.10)$$

The unknown face then belongs to the class with the minimum  $\varepsilon_k$ .



# Chapter 8

## Implementation

This section briefly discusses the tools and methods used to implement a prototype of the proposed design.

Note that Appendix A contains details on the classes and files related to the system.



### 8.1 Tools and environments used

**Programming language:** The system was implemented using Python 2.5 [20] and totals around 2000 lines of code, excluding libraries. Python is a dynamically typed language and was chosen for rapid development nature and wealth of scientific libraries that are available.

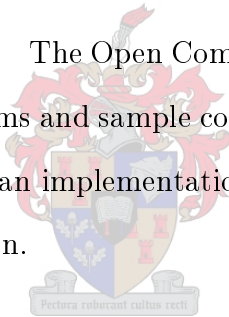
**Image manipulation library:** The PIL [21] image library was used in many places that required image manipulation such as rotation or flood fill.

The PIL library also allows for easy conversions between images and Python arrays.

**Scientific libraries:** Scipy [22] in combination with Numpy [23] is used extensively where matrix manipulation such as convolutions or filters are required. Scipy also provides a k-means clustering algorithm that was used for the skin detection.

LibSVM [24] is an integrated software for support vector classification, with support to work with Python lists.

**Computer Vision Library:** The Open Computer Vision Library (OpenCV [25]) is a collection of algorithms and sample code for various computer vision problems. OpenCV provides an implementation of the Haar-based classifier used for face and eye detection.



**Graphics Library:** wxPython [26] was used to build the interface. wxPython is a GUI toolkit for the Python programming language that allows for programs with a robust, highly functional graphical user interface.

## 8.2 Translating head movements to cursor movements

In our system the cursor position is controlled by making small head movements, while blinks are used to simulate mouse clicks. To track the position

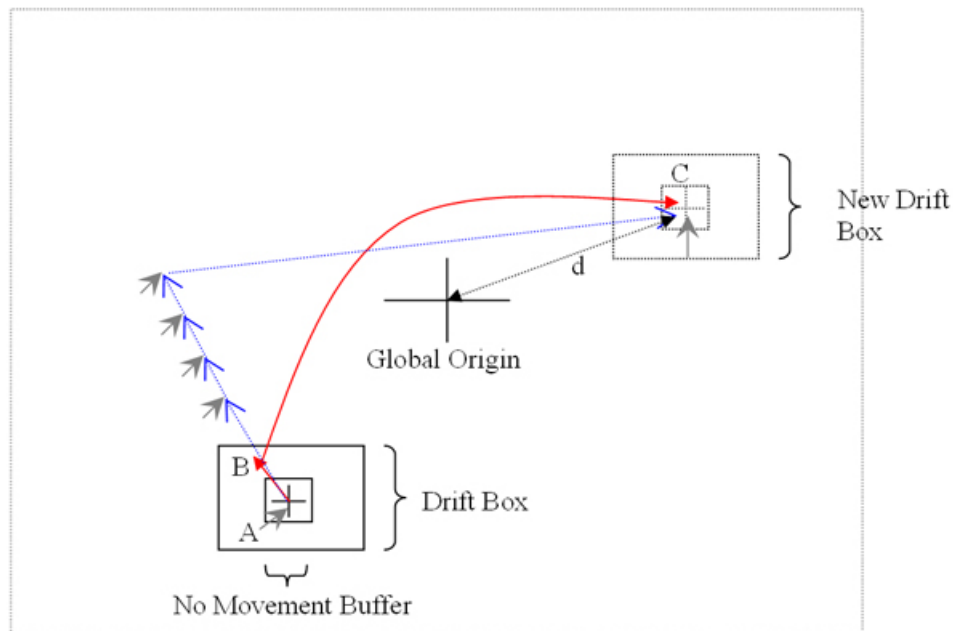


Figure 8.1: Translating head movements to cursor movements.

of the head, we use the location of the eye pupils. By default the pupil of the right eye is used for tracking, but during a right eye blink the tracking point is switched to the left eye.

To translate head movements to movements of the mouse cursor there are two simple methods that can be considered:

1. Head movements can be directly mapped to coordinates on the screen. This way when the user turns his/her head towards a certain part of the screen, the mouse cursor is immediately moved to that position. Such a system is good for large mouse movements as the cursor can quickly be moved to different parts of the screen, but it can be difficult to accurately make small movements such as clicking on a button.

2. The mouse cursor can drift head movements relative to some neutral position. In this setup, when the user turns his/her head to one side, the cursor slowly drifts in that direction. The further a person's head is turned, the faster the cursor will drift in that direction. With this system the mouse cursor can be controlled very accurately for small movements (since the cursor can be slowly drifted to the correct location) but large movements is not as fast.

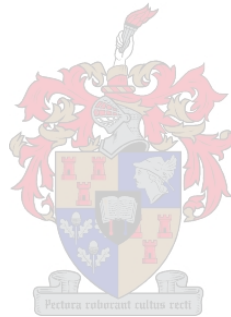
For our system we use a combination of the two methods. When the system first recognizes a user the coordinates of his/her face are noted. This point is then used as the absolute position for the direct mapping technique when the user makes a large head movement. For small head movements however the drifting technique is used, but rather than using the absolute position as neutral point for the drift method, the neutral point is updated after every large head movement. This means that a user can make a large head movement to quickly position the cursor close to the desired point on the screen, and then using further small movements can drift the cursor accurately to the exact point.

Figure 8.1 explains this method graphically:

1. The mouse cursor and head position is initially both at position A. It is also the neutral position for the drift box.
2. The user then moves his head slightly to position B. Since this movement is within the drift box, the mouse cursor is slowly drifted in the

direction of B (indicated as the blue arrows). If the user moves his head back to A, the cursor will stop drifting.

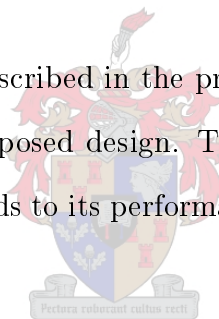
3. The user then moves his head to position C. Position C is outside the drift box, so the cursor is immediately moved (without drifting), and the drift box is now centered around C.



# Chapter 9

## Results

In this chapter the system described in the previous sections is evaluated to prove the viability of the proposed design. The quality of the implemented system is explored with regards to its performance, recognition accuracy and usability.



### 9.1 Time Performance Analysis

The system was tested using a 1.73GHz Intel(R) Core processor with 1GByte of RAM and a  $352 \times 258$  resolution web camera. The average values of the time analysis performance is shown in table 9.1.

The total time of 51ms equals a frame rate of 19.6 frames per second (fps), which allows operation in real-time.

Table 9.1: Time analysis performance

	time (ms)
Face detection	18
Eye detection	9
Blink detection	2
Iris detection	22
Total	51

## 9.2 Eye and Iris Detection Accuracy

### 9.2.1 Description of the test databases

The system has been tested on two databases, the BioID [11] and the extended M2VTS (XM2VTS) [9].

The XM2VTS database contains 2360 color images, each one showing the face of one out of 295 different persons. The images were taken under controlled lighting conditions with a uniform background, making the task of face and eye detection much easier.

The BioID database consists of 1521 images ( $384 \times 288$  pixel, grayscale) of 23 different persons. The pictures were taken under various lighting conditions in a complex background, and as such this database is considered as the more difficult one for the detection task.

Both databases contain faces with glasses and in a few images of the BioID, people have their eyes shut or pose in various expressions.

Some results from the BioID database are shown in Figure 9.2 and Figure 9.3.

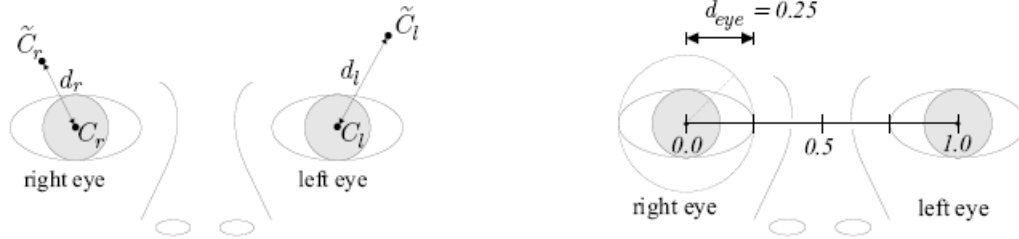


Figure 9.1: Relations and relative error.

Table 9.2: Results on the BioID database

	T=0.25			T=0.125		
	Successful	Failed	Percentage	Successful	Failed	Percentage
People without glasses	1016	49	95.4%	972	93	91.3%
People with glasses	371	29	92.3%	346	54	86.6%
People with closed eyes	51	4	92.7%	47	8	85.5%
Total	1438	82	94.6%	1365	155	89.8%

## 9.2.2 Experimental Results

For the pupil/iris detection, the criterion of calculating the correct detection rates are the following [19]:

$$d_{eye} = \frac{\max(d_l, d_r)}{\|C_l - C_r\|} < T \quad (9.1)$$

where  $d_l$  and  $d_r$  are the distances between the true eye centers  $C_l, C_r$  and the eye centers found by the algorithm  $\tilde{C}_l, \tilde{C}_r$  as depicted in Figure 9.1.

$T$  is the threshold used to declare a successful detection. For an average face a threshold of  $T = 0.25$  equals the distance of half an eye width, while  $T = 0.125$  equals half the width of an iris, as shown in Figure 9.1.

Results for  $T = 0.25$  and  $T = 0.125$  are shown in Tables 9.2 and 9.3.



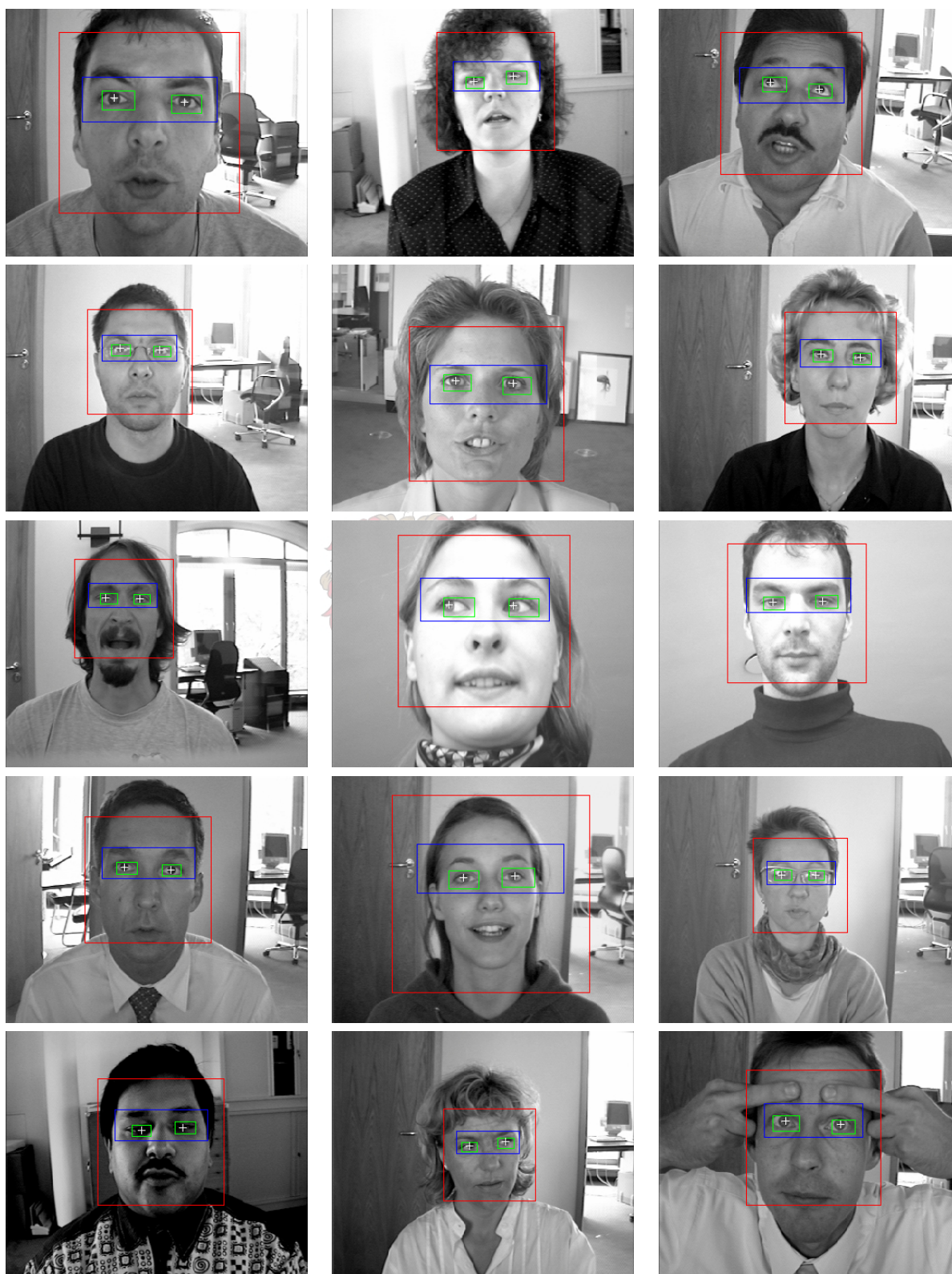


Figure 9.2: Sample iris detection results

Table 9.3: Results on the XM2VTS database

	T=0.25			T=0.125		
	Successful	Failed	Percentage	Successful	Failed	Percentage
People without glasses	1497	28	98.2%	1486	39	97.4%
People with glasses	796	39	95.3%	764	71	91.5%
Total	2293	67	97.2%	2250	110	95.3%

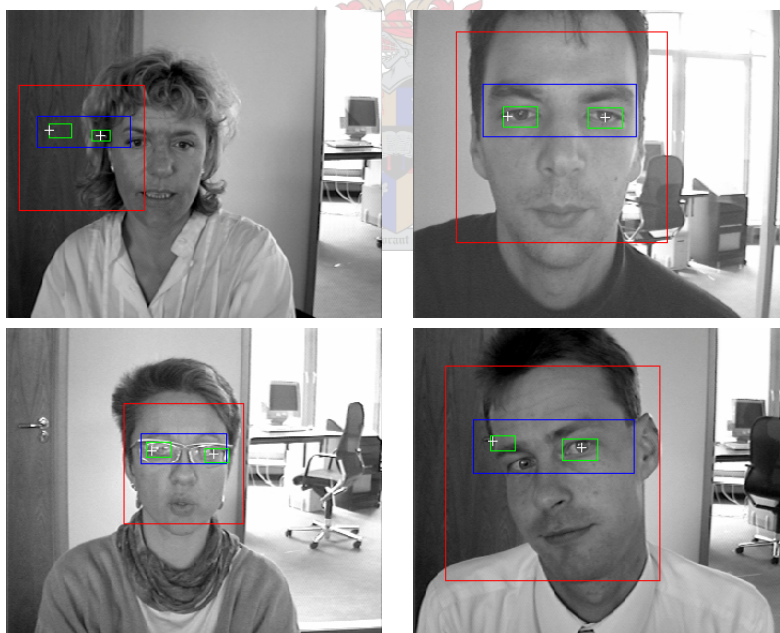


Figure 9.3: Sample images where the iris was incorrectly identified

### 9.3 Skin Detection

The skin detector was tested on several images in different lighting conditions. It performed reasonably well under most situations, with the exception of strong lighting from the side. When the lighting conditions illuminates one side of a users face more than the other, the K-Means clustering algorithm would often classify only one side as skin, resulting a poor skin map.

The Gaussian filter that is applied as the final step on the skin detection also posed a problem. Under good lighting conditions the K-Means algorithm usually does a great job of classifying the skin regions, but much of the detail is lost after the image has been filtered. This is especially evident in the regions around the ears where the skin map has sharp edges.

Some results are shown in Figure 9.4.

### 9.4 Usability Analysis

To test the usability of the system, six users were asked to try out the system. The users were asked to give feedback on three parts of the system, namely the login system, the blink detection and their overall impression. The tests were then repeated under different camera setups and lighting conditions.

**Login System:** All six users were added to the database and then asked to use the login system. To eliminate any false identifications that may result from head rotation, the users were asked to look directly at the camera when





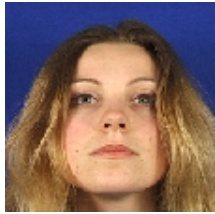







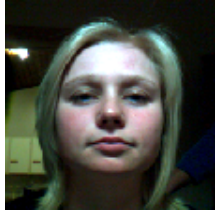


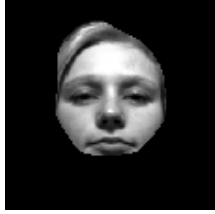
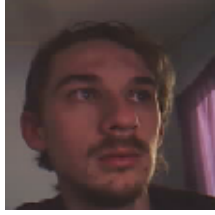
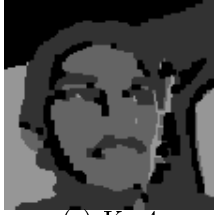


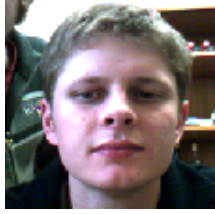
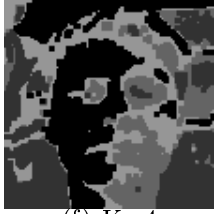

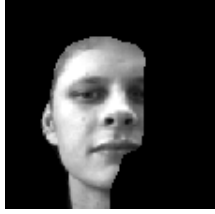
Image	K-Means Clustering	Binary Skin Image	Resulting Image
	 (a) K=4		
	 (b) K=3		
	 (c) K=3		
	 (d) K=6		
	 (e) K=4		
	 (f) K=4		

Figure 9.4: Sample skin detections. The last image is an example where bad lighting from the side resulted in poor skin detection.



Figure 9.5: Sample images where the user is positioned above the screen. In the left image the user's eyes are open but is looking down, while in the right image the eyes are closed.

using the login system.

- Under good lighting conditions, the system was able to reliably identify each of the six users.
- Under bad lighting (for example if the light source is to the side of the user such as in the last image in Figure 9.4) the system could still identify the correct user over 70% of the time, as long as the lighting remained consistent during use.
- Since the face image is scaled with relation to the detected iris positions, the system is extremely sensitive to incorrect iris detections.

**Blink Detection:** Each user was asked to test out the blink detection system twice: First using the default training data, and then again after the classifier was specifically trained for the user's eyes.

Results:

- The position of the user in relation to the screen and camera is very important. If the user's head is positioned higher than the screen the

system performed poorly. This is because in such a setup it is almost impossible to distinguish between a blink and when the user is simply looking down to the bottom of the screen. (see Figure 9.5)

- Three out of 6 users could reliably use the system with the default training data.
- After the classifier was trained for each user, all 6 users could control the system. One of the users had difficulty in keeping his other eye open during a blink, and several attempts was required to generate reliable training data for this user.

**Overall Impression:** The users initially had trouble controlling the mouse cursor, but quickly learned after some practice.

Some of the comments made:

- Users initially expected the system to track where you look at the screen (gaze tracking), and it took them a while to get used to controlling the computer with head movements.
- Some users got confused when they changed their sitting position in front of the camera. Changing your sitting position can cause the mouse to get stuck in one of the corners, and the users expected the system to automatically realize that their neutral position has changed.
- Many believe the system shows promise and could be used as a viable means of input to control the computer.

# Chapter 10

## Conclusions and Recommendations



This chapter discusses some suggestions for further research and improvements that could be made. Some suggested applications are also mentioned and the study is concluded in a summary.

### 10.1 Future Work

Users of the system often got confused when moving around in front of the camera, since the system expects a global neutral point for the head position. An possible enhancement of the system could be to automatically recognize when a user has shifted his/her position, and then update the neutral position.

The system is also not fully automatic for all users. For about 50% of the users the blink detection has to be trained by the user. To achieve a better ratio the default training set for blink detection can be expanded to include more real world examples. Other methods of representing the image as a vector, such as histogram of gradients, can also be explored.

Furthermore the login system will always recognize an unknown user using the closest match it can find, even if the unknown user is not part of the system. To only allow registered users to login to the system a more sophisticated classifier can be created by choosing a maximum threshold value for the euclidean distance,  $\varepsilon_k$ . If the input image is above the threshold, the image can be determined as an unknown face. Care will have to be taken to make sure the threshold works with different camera and lighting setups.

## 10.2 Suggested Applications

In this section we point out some specific applications for system.

- Help for disabled persons: The system was developed to help disabled persons in wheelchairs control the computer. The system should be able to help even severely disabled persons (unable to control their head movements) by using only the blink detection as a switch based input method.
- Computer games: Eye motion can be used as an additional input in computer games.

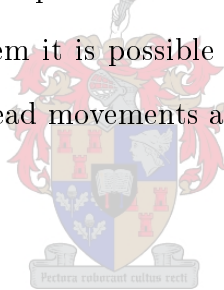


- Extra security for login systems: The face recognition system can be extended to provide better security for login systems. (To be used in combination with an existing system such as passwords.)

### 10.3 Conclusions

This study explored the viability of using only a web camera to control the computer.

A system capable of face tracking, eye detection, blink detection and finally face recognition was developed and found to be usable when performing simple tasks. Using the system it is possible to control the mouse position and clicks using only small head movements and eye blinks.



# Bibliography

- [1] Shaogang Gong, Stephen J McKenna and Alexandra Psarrou. “Dynamic Vision - From Images to Face Recognition”, Imperial College Press (2000).
- [2] OpenCV. “Open Computer Vision Library Manual.” Available: <http://sourceforge.net/projects/opencvlibrary/>
- [3] Paul Viola and Michael Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, Computer Vision and Pattern Recognition Conference, volume 1, pp 511-518 (2001).
- [4] Rainer Lienhart and Jochen Maydt, “An Extended Set of Haar-like Features for Rapid Object Detection”, IEEE ICIP 2002, volume 1, pp. 900-903 (2002).
- [5] Michael Chau and Margrit Betke, “Real Time Eye Tracking and Blink Detection with USB Cameras”. Dept. of Computer Science, Boston University, Boston 02215 (2005).

- [6] T. N. Bhaskar, Fo Tun Keat, Surendra Ranganath and Y. V. Venkatesh, “Blink Detection and Eye Tracking for Eye Localization”. TENCON 2003, volume 2, pp 821-824 (2003).
- [7] 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 (1997).
- [8] Sandra Trejo Guerrero. “Model-Based Eye Detection and Animation”. Department of Electrical Engineering, Image Coding Group, Linkopings University (2006).
- [9] XM2VTS - Database of Human Faces by the XM2VTSDB multi-modal face database project, <http://www.ee.surrey.ac.uk/CVSSP/xm2vtsdb>
- [10] DTREG - “Introduction to Support Vector Machine (SVM) Models.” [Online]. Available: <http://www.dtreg.com/svm.htm>
- [11] The BioID face database. [Online]. Available: <http://www.bioid.com/downloads/facedb/facedatabase.html>
- [12] Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin. “A Practical Guide to Support Vector Classification”. Department of Computer Science, National Taiwan University, Taipei 106 (2003).
- [13] The EagleEyes Project, Boston College. Available: <http://www.cs.bc.edu/~eagleeye>.

- [14] HIPR Image Processing Learning Resources. [Online]. Available: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/>
- [15] Gene H. Golub and Charles F. van Laan. “Matrix Computations”, Johns Hopkins University Press, Baltimore, MD (1996).
- [16] Wikipedia. [Online]. Available: <http://en.wikipedia.org/>
- [17] Ilker Atalay. “Face Recognition Using Eigenfaces”. Istanbul Technical University, Institute of Science and Technology (1996).
- [18] Drexel University. “Eigenfaces Tutorial”. [Online]. Available: <http://www.pages.drexel.edu/~sis26/Eigenface%20Tutorial.htm>
- [19] Oliver Jesorsky, Klaus J. Kirchberg and Robert W. Frischholz. “Robust Face Detection Using the Hausdorff Distance”, 3rd International Conference on Audio- and Video-Based Biometric Person Authentication 2001, Halmstad, Sweden, pp 90-95 (2001).
- [20] Python Programming Language. Available: <http://www.python.org/>
- [21] PIL Image Library. Available: <http://www.pythonware.com/products/pil/>
- [22] Scipy. (Scientific Python Library). Available: <http://www.scipy.org/>
- [23] Numpy. (Numerical Python Library). Available: <http://numpy.scipy.org/>

- [24] LibSVM. (SVM Library for Python). Available:  
<http://csie.ntu.edu.tw/~cjlin/libsvm/>
- [25] OpenCV. (Open Computer Vision Library). Available:  
<http://sourceforge.net/projects/opencvlibrary/>
- [26] wxPython. (Python Graphics Library). Available:  
<http://www.wxpython.org/>



# Appendix A

## Code Structure

The developed system was implemented using Python 2.5, and consists of 7 main python files plus a few auxiliary files. We list the main files and a short description in Table A.1.

To implement the various parts of the system (such as the face recognition and blink detection) we used 7 external libraries. These libraries, together with their version numbers are shown in Table A.2.

Table A.1: Python source files (excluding modified libraries)

Filename	Line count	Description
conversions.py	79	conversion between different image and matrix formats
webcam.py	38	web camera routines
NewMember.py	190	new member management
facedetect.py	580	Face and feature detection routines
Frame1.py	760	Main system interface
App1.py	32	System starting piont
imatch.py	180	SVD image recognition library

Table A.2: Python libraries used

Library	Version	Url
wxPython	2.8.1.1	<a href="http://www.wxpython.org/">http://www.wxpython.org/</a>
scipy	0.5.2	<a href="http://www.scipy.org/">http://www.scipy.org/</a>
numpy	1.0.1	<a href="http://numpy.scipy.org/">http://numpy.scipy.org/</a>
OpenCV	1.0	<a href="http://sourceforge.net/projects/opencvlibrary/">http://sourceforge.net/projects/opencvlibrary/</a>
PIL	1.1.6	<a href="http://www.pythonware.com/products/pil/">http://www.pythonware.com/products/pil/</a>
LibSVM	2.83	<a href="http://www.csie.ntu.edu.tw/~cjlin/libsvm/">http://www.csie.ntu.edu.tw/~cjlin/libsvm/</a>
VideoCapture	0.9	<a href="http://videocapture.sourceforge.net/">http://videocapture.sourceforge.net/</a>