Thesis Committee

Drs. Dr. L. J. M. Rothkrantz	Delft University of Technology
Dr. Ir. C.A.P.G. van der Mast	Delft University of Technology
Dr. K. van der Meer	Delft University of Technology
MSc. D. Datcu	Delft University of Technology

Contact: Wai Shung Wong w.s.wong@ewi.tudelft.nl
William Chan w.chan@ewi.tudelft.nl

Delft University of Technology
Faculty of Information Technology and Systems
Man Machine Interaction group
September 2005

Revised: 28 September 2005.

Acknowledgements

It is common to be asking questions during any research, but finding the right answers is much more of a challenge. We have learnt that the key to successful scientific research is an aggressive and persistent attitude and patience to conduct research. There were times of stress and also times of relieve. We have found ourselves lost in the enormous amount of information, and also stuck at the little information we can lay our hands on, especially in the beginning which is very crucial in every research. But a good beginning is just as important as a successful ending. We would like to acknowledge the help and advice we get during the engagement of this thesis.

We especially would like to thank Professor Leon Rothkrantz for supervising, guiding and advising us throughout the whole project. The continuous feedback and discussions about the ideas were really pleasant and motivating. We would also like to thank Dragos Datcu for his cooperation and valuable help. He is also the one we have been turning to for countless brainstorm sessions. To everyone, especially our family and friends who have been supporting us we would like to express our sincere gratitude. Last but not least we like to thank each other for the mutual support, the hard work and the laughter during the long hours.

Delft, 16 September 2005

William Chan
Wai Shung Wong

Abstract

Computer vision has become one of the most challenging subjects nowadays. The need to extract information from images is enormous. Face detection and extraction as computer-vision tasks have many applications and have direct relevance to the face recognition and facial expression recognition problem. Potential application of face detection and extraction are in human-computer interfaces, surveillance systems, census systems and many more.

In this thesis the focus is on the realization of a fully automatic emotion recognition system. The exploited approach splits the system into four components. Face detection, facial characteristic point extraction, tracking and classification. Face detection is employed by boosting simple rectangle features that give a decent representation of the face. These features also allow the differentiation between a face and a non-face. The boosting algorithm is combined with an Evolutionary Search to reduce the overall search time. Facial characteristic points (FCP) are extracted from the detected faces. The same technique applied on faces is utilized for this purpose. Additionally, FCP extraction using brightness distribution has also been considered. Finally, after retrieving the required FCPs the emotion of the facial expression can be determined. The Relevance Vector Machine (RVM) is the classification method that is used where a classifier is required.

Index terms - face detection, facial feature extraction, facial characteristic point extraction, facial expression recognition, Relevance Vector Machine, corner detection, AdaBoost, evolutionary search, hybrid projection.

CHIM group

The Computer Human Interaction Machine (CHIM) group aims at finding solutions to specific problems in the area of machine vision and machine learning. The ultimate goal is to use the models so as to enhance the capabilities of the intelligent systems. The group is focused on the research in the fields of sensor data extraction and processing, data fusion, reasoning and information presentation. The efforts are concentrated on the development of functional components for applications that perceive the environmental world, extract the meaning and construct proper feedback in specific contexts.

Current ongoing projects include: Face detection, Facial feature detection, Facial expression recognition, Pattern recognition and AIBO. More information about the CHIM group can be found at: http://mmi.tudelft.nl/.

Table of Contents

List of Figures and Tables

Part I

Introduction and Problem Definition

1

Introduction

For the past decades, many projects have been started with the purpose of learning the machine to recognize human faces and facial expressions. Computer vision has become one of the most challenging subjects nowadays. The need to extract information from images is enormous. Face detection and extraction as computer-vision tasks have many applications and have direct relevance to the face-recognition and facial expression recognition problem. Potential application of face detection and extraction are in human-computer interfaces, surveillance systems, census systems and many more. It is not so hard to imagine the importance of face detection in the means of face and emotion recognition. The importance of this subject can be ratified by the recent terrorism bombings in London. In London, monitoring of people especially in the public places is done by closed-circuit cameras and televisions, which are linked via cables and other direct means (see Figure 1). These can also be found in casinos and banks for instance.

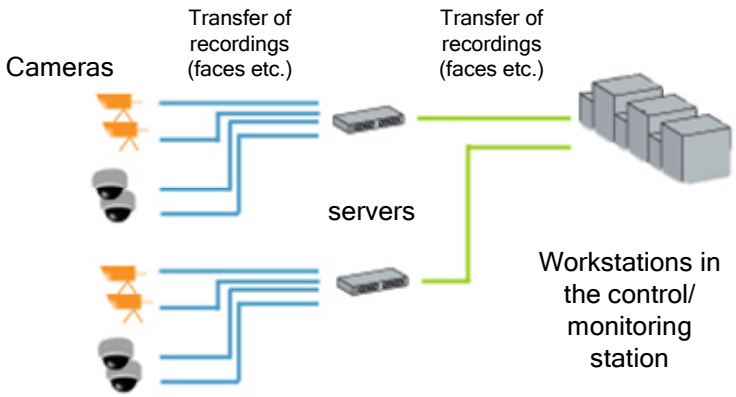


Figure 1: Closed-circuit television (CCTV) network.

The CCTV systems [Dick03] transmit their digital images over the network and the images are analyzed with face- and behavioural-recognition software to identify unusual patterns. After the incidents the authorities were able to identify the attackers with the help of the recordings of these cameras. As they have done in London, video shots of faces allow for the prompt identification of suspects soon after security events happen. The set-up is very simple. Some cameras exist to capture the faces of people as they pass through critical locations. Other cameras like the complementing overview cameras are able to detect a threat, which clearly failed in doing its job to prevent the terror.

The surveillance systems like the CCTV have the same main principle. First, a face is detected. Then, the detected face can be tracked and enables important features to be extracted for analysis. The type of features that is extracted depends strongly on what the system wants to achieve. Features can be obtained for either the recognition of a face (identification) or the recognition of an emotion/expression. Face identification is relevant in retrieving a person's identity and emotion recognition has its contribution in prevention of crime and calamities for instance. In the latter it concerns aggression detection, unusual or nervous behavioural detection. That is also why extraction and recognition of facial expression has been a hot topic last decades. It is important to note that face detection and facial expression recognition are distinct subjects. In face detection the different expressions are considered as noise, where as in facial expression recognition the identity is considered as noise. The latter implies that different persons have different neutral faces with different feature shapes (big/small eyes, big/small mouth, etc.).

Facial expressions are crucial in human communication. Human communication is a very complex phenomenon as it involves a huge number of factors: we speak with our voice, but also with our hands, eyes, face and body. The interpretation of what is being said does not only depend on the meaning of the spoken words. Our body language i.e. gestures modify, emphasize, and contradict what we say. Facial expressions provide sensitive cues about emotional responses and play an important role in human communication. Therefore, it is valuable if this aspect of human communication can also be applied for more effective and friendly methods in man-machine interaction. In verbal communication, the conversation becomes very difficult if neither participant understands the language the other is speaking. The same applies for nonverbal communication: both parties must have the same interpretation of the nonverbal signals. Like language, nonverbal signals are not universal. Moreover, they are context and culture dependant. Research has shown that the ability to communicate nonverbally is something that has to be learnt. According to (Eckman & Friesen 1972) people are born with the ability to generate and interpret only six facial expressions: happiness, anger, disgust, fear, surprise and sadness. All

other facial expressions have to be learned from the environment the person grows up. Humans are capable of producing thousands of expressions that vary in complexity, intensity, and meaning. Subtle changes in a facial feature such as tightening of the lips are sufficient to turn the emotion from happy to angry. And to think that the eyes and eyebrows can also take on different shapes, one may imagine how complex the problem gets.

With the ability to recognize facial expressions and thus getting information about the psychological state of a person, a lot of applications can be considered. Systems can be made smarter and safer. Consider for example the Driver Vigilance Monitoring System [Dikk04]. The idea is that the system will alert the driver when it sees that he/she is in a state of somnolence. As the name already indicates, the system is installed in the car for monitoring the driver's facial expression continuously. The input to this system is a sequence of images of the driver's face captured by a camera. The system will then make an assessment based on the movements of parts of the face, especially the eyes and eye lids. Another interesting application that emphasizes the importance of nonverbal communication is the Facial Expression Dictionary (FED). This is an online dictionary that allows us to find the meaning of certain facial expressions. In fact, given a facial expression, the system is able to extract and recognize the given facial expression. More about this system will follow in the next chapters, as this thesis project is an extension to the FED framework.

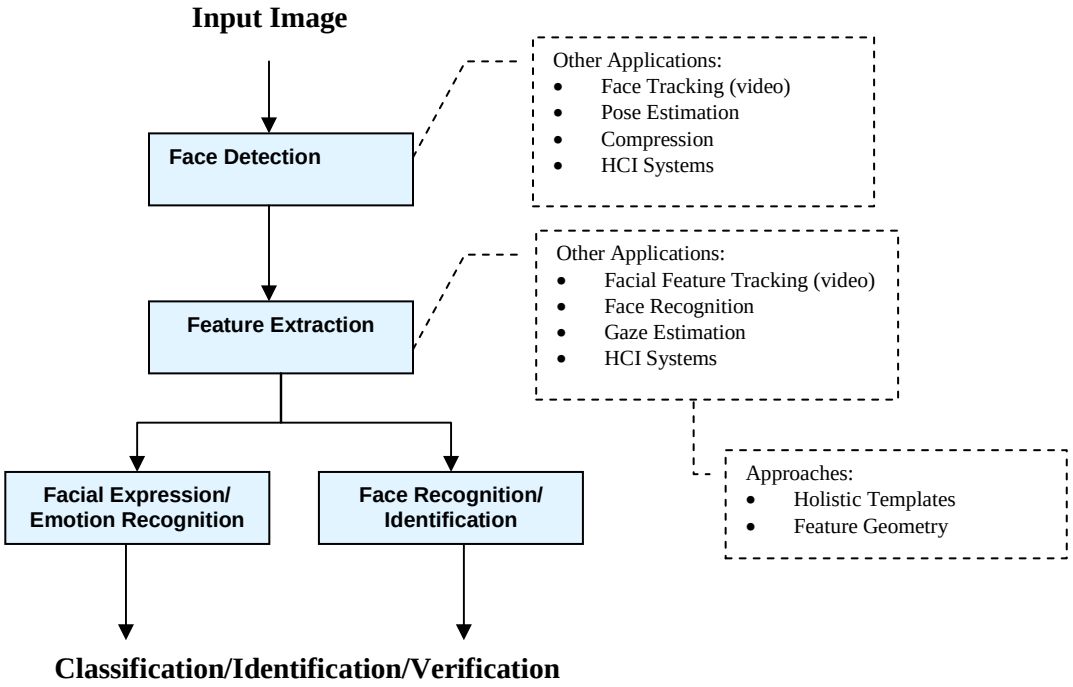


Figure 2: Generic emotion recognition system.

For a system to recognize a facial expression the system first needs to detect and locate the faces in the image or video (see Fig. 1 for a generic face recognition system). Depending on the used system, the face in the image needs to satisfy some constraints like full frontal view, silhouette view, rotations of the frontal face within some boundaries and certain light conditions. The constraints all depend on the face model that the system uses. Having found a face, the next step is to extract the facial features: eyes/eyebrows, nose/nostrils, mouth/lips, cheeks/forehead, chin, etc. Not all of these features are of equally importance for facial expression recognition. The final step in this face analysis process is to pass the obtained data to an expert system that determines in what kind of psychological state the person was.

In the past, Morishima et al. [Mori93] implemented a five-layered manual-input neural network which is used for recognition and synthesis of facial expressions. In [Zhao96] they explained a singular emotional classification of facial expressions using a three-layered manual-input backpropagation neural network. [Kearney and McKenzie] developed a manual-input memory-based learning expert system, which interprets facial expressions in terms of emotion labels given by college students without formal instruction in emotion signals. Rothkrantz et al. [Roth00] proposed a point-based face model composed of two 2D facial views, namely the frontal- and the side view. Based on a point-based face model, expression-classification rules can be converted straight-forwardly into the rules of an automatic classifier. In [Chan04] we tried to detect a face and extract facial features using the Relevance Vector Machine classifier. In order to recognize facial expressions, the additional step to do is to find the facial feature points which will be used for analyzing the facial expression. This facial expression recognition method will be examined and explained in this report.

A critical step in detecting a face with its facial features is to distinguish the face and non-faces. This is done by using a classifier. There are different kinds of classifying methods. Some well known examples are K-Nearest Neighbours (KNN), Tree-Augmented Naive-Bayes (TAN) and Support Vector Machines (SVM). The latter, being a state-of-the-art classification method, is based on some rather simple ideas and provides a clear intuition of what learning from examples is about. Practical applications have already shown outstanding high performances of this classification method. Some examples of recent applications of SVM are in handwritten digit recognition [Vapn96, Burg97], face detection in images [Osun97] and text categorization [Duma98, Joac97]. However, despite its success, there are some significant and practical disadvantages in the SVM learning methodology. A recently introduced classification method based on the idea of the Support Vector Machine is the Relevance Vector Machine (RVM) [Tipp00]. RVM is a Bayesian framework for regression and classification with analogous sparsity

properties to the SVM. It can be seen as a probabilistic version of SVM but without the disadvantages and simultaneously providing a number of additional advantages (see Chapter 4 for more details).

1.1 Related works

In [Chan04] an extensive overview is given of related works in the area of face detection and facial expression recognition. To provide a context for our problem definition we review some of them globally. Section 1.1.1 to 1.1.3 discuss about some existing face detection work. Section 1.1.4 to 1.1.6 describes some of the projects done on facial expression recognition.

1.1.1 Applying Support Vector Machine to face detection

A Support Vector Machine (SVM) is introduced for detecting human faces in grey-level images [Osun97]. First, face-like patterns are scanned at many possible scales and then SVM is used to classify them into the appropriate class (face/non-face). The SVM is trained with a second degree polynomial as kernel function and an upper bound C=200. This upper bound is the expected value of the ratio between the number of support vectors and the total number of data points on the generalization error. Also a database is used consisting of face and non-face 19x19 pixel patterns, assigned to classes +1 and -1 respectively. Once the SVM has been trained it is primarily used over images that do not contain faces. Misclassifications are stored for use as negative examples in subsequent training phases. Images with many different texture patterns are good resources for false positives. This way of reusing misclassified examples is called the 'bootstrap' method which was successfully used by Sung and Poggio [Sung98]. This method will also reduce the size of the non-face class which is much broader and richer than the face class. After the SVM is fully trained it is incorporated as a classifier in the system for pattern recognition of face/non-face.

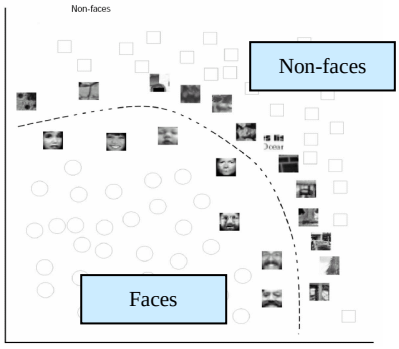


Figure 3: Geometrical interpretation of how the SVM separates the face and non-face classes. The patterns are real support vectors after training the system.

This SVM face detection system is compared to Sung and Poggio's on two sets of images. On test set A, which contains 313 high-quality images with the same number of faces, SVM has a detection rate of 97.1% with 4 misclassifications, while Sung and Poggio's has a detection rate of 94.6% with 2 misclassifications. On test set B containing 23 images of mixed quality with a total of 155 faces, SVM has a slightly poorer performance. While having a same detection rate, SVM has 20 misclassifications against 11 of Sung and Poggio's.

1.1.2 Neural-network-based face detection

Rowl et al. [Rowl98] present a neural network-based face detection system for upright frontal views of faces. In their work only gray-scale images are considered. The algorithm works by applying one or more neural networks directly to parts of the input image, and judging their results. Each network is trained to output the presence or absence of a face. The algorithms and training methods are designed to be general, with little customization for faces. The neural networks are trained with images containing faces and images not containing any faces. It is also using the "bootstrap" method so as to reduce the size of the training set that is needed for images not containing faces.

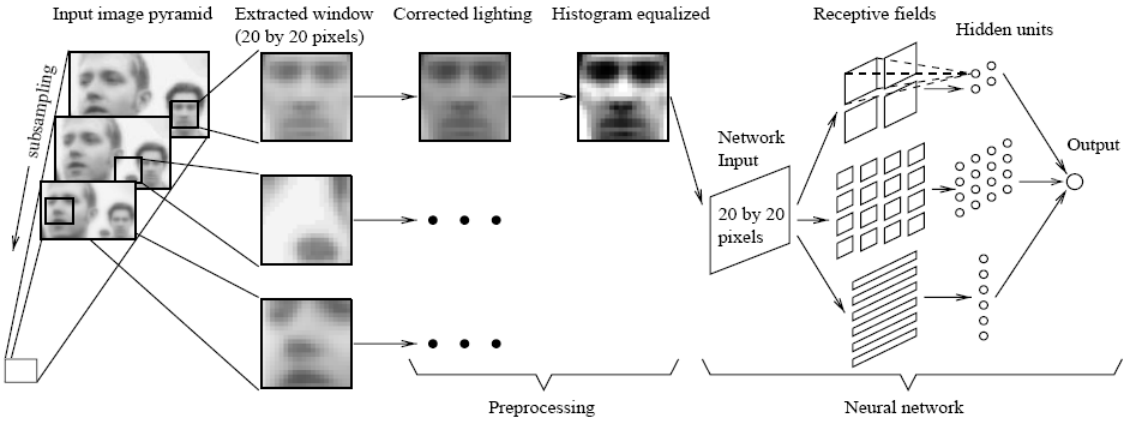


Figure 4: Face detection algorithm in [Rowl98].

The system operates in two stages. The first component uses a neural network-based filter which receives as input a 20x20 pixel region of the image. The output will be positive or negative for the presence, respectively absence of a face in the sub-window. Because a face can appear in every part of an image, the filter is applied at every pixel position in the image. In order to detect faces larger than the window size, the input image is repeatedly reduced in size by sub sampling. Before the 20x20 pixels are passed to the neural networks, it is preprocessed with lighting corrections and histogram equalization. Experiments show that the raw output from a single network can contain a number of false detections. The second stage of the system consists of

ways to deal with this problem. Two strategies for improving the reliability of the neural network's outputs are presented: merging overlapping detections from a single network and arbitrating among multiple networks.

The authors also compared the performance of their system to other illustrious face detection systems. Amongst them are Sung and Poggio's system [Sung98] and the SVM face detection system of Osuna, Freund and Girosi [Osun97]. The support vector machine has a number of interesting properties, including the fact that it makes the boundary between face and non face images more explicit. In the comparison using the same set of 23 images, Sung and Poggio's system results in a slightly poorer performance than Rowley's system with a difference of 6 faces out of 155.

Another neural network-based face detection system is that of Kah-Kay Sung and Tomaso Poggio [Sung98]. They developed a generic human face detection system that finds vertically oriented and un-occluded frontal views of human faces in gray-level images. Their system starts with passing a small 19x19 sub-window over all portions of the image. The system is using a clustering method with six "face" and six "non-face" clusters. Each cluster is a multi-dimensional Gaussian with a centroid location and a covariance matrix that describes the local data distribution. These clusters measure the "difference" between the sub-window and some prototype distribution. The last step is to use a neural network to classify the sub-window as face or non face using this 'difference'.

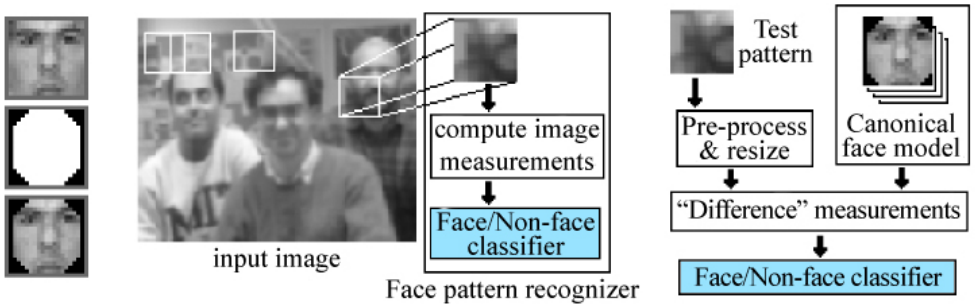


Figure 5: Face detection system as in [Sung98].

The most critical part of their system is the learning algorithm for classifying window patterns as faces or non-faces. The key components of this algorithm are: 1) this system uses a distribution-based face model. So, they used a normalized window of 19x19 pixels to model the distribution of canonical frontal face patterns. 2) In order to classify new patterns the "difference" between each sub-window (after preprocessing and resizing) and the face distribution model is measured and the result of these measurements are passed to a trained neural network which determines

whether or not the new window pattern contains a face. This training procedure is done in the 'bootstrap' fashion as described before. The system is trained with 4150 positive and nearly 43000 negative examples.

They tested their system on two test databases. For the first database, consisting of 301 high quality digitized images with frontal and near-frontal faces of 71 different people, the system correctly finds 96.3 percent of all the face patterns and it makes only three false detects. For the second database, consisting of 23 images of mixed quality with 149 face patterns in complex backgrounds, the system achieves a 79.9 percent detection rate with five false detections. They have not compared their system to other face detection systems.

1.1.3 Real-time object detection

P. Viola and M. Jones [Viol01] presented a face detection system based on their real-time object detection framework. This framework involves three key contributions to the very positive experimental results. The first one is the introduction of a new image representation called *integral image*. An integral image makes it possible for the detection procedure, which classifies images based on the value of simple features, to process an image very fast. Once an integral image is computed, any of its features can be extracted at any scale or location in constant time. The second is a learning algorithm, based on AdaBoost. This learning algorithm's goal is to construct classifiers by selecting a small number of important features. The third contribution is a method for constructing complex classifiers by combining different classifiers in a cascade structure. This increases the speed of the image processing process dramatically; it allows the detector to discard background regions very quickly and consequently use more time for more promising regions of the image. In the cascade structure, a series of classifiers are applied to every sub-window. They will first pass through the initial classifier which eliminates a large number of negative examples with very little processing. Then it will be passed to the subsequent layers which will in turn eliminate additional negatives but require additional computation. In the end the number of sub-windows will be reduced. This process is based on the fact that within any single image a great majority of negative sub-windows exists.

The training dataset that is used for this face detection system consisted of 4916 hand labeled faces scaled and aligned to a base resolution of 24 by 24 and 10000 24x24 non-faces. The final detector is a 32 layer cascade of classifiers which included a total of 4297 features. This system was compared to Rowley's face detection system described above [Rowl98], which was by that time widely considered as the fastest. The result was really promising. While having comparable face detection performance, it was about 15 times faster.

1.1.4 Expert system for automatic analysis of facial expressions

Rothkrantz et al. [Roth00] is working on the development of an intelligent automated system for the analysis of non-verbal communication. The result was implemented as Integrated System for Facial Expression Recognition (ISFER). In contrast to existing facial feature detectors which utilize single image processing technique, a hybrid approach to facial feature detection is presented. Instead of fine-tuning existing facial feature detectors, they combine multiple feature detection techniques that are applied in parallel. The system deals with static face actions, which mean that only the end-state of the facial movement is measured and compared to the neutral position. The face model is defined as a point-based model composed of two 2D facial views, namely the frontal- and the side view. The features defined by the model are extracted automatically from the inputs which are still full-face/profile images. Different techniques are used for extracting the facial features. The used algorithm first locates all facial features with namely ANN for eye-, nose-, and mouth-detection. Once the windows containing the features are found, different ANNs and other algorithms are applied for defining the facial characteristic points for each feature. The changes in position of these points are observable and so the validity of the model can be visually inspected.

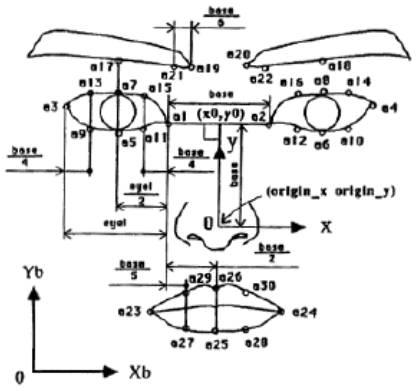


Figure 6: Kobayashi and Hara face model.

1.1.5 Automatic feature extraction

In [Koba97] the authors developed an animated 3D face robot for real-time interaction with human beings. The aim is to let the robot produce realistic human-like responses. In order to react appropriately, the robot must first recognize the facial expression of the human. Then it has to make a proper decision for an action, and finally it has to perform the action. The face model used for this system is described in [Hara97]. The authors proposed a face model with 19 facial characteristic points (see Figure 6). Movement of groups of facial characteristic points indicates a

certain facial expression. These points are used in their system for facial expression recognition and facial expression production.

Also different image processing algorithms are presented for extracting features of facial organs and face contour. The facial organs include eyes, eyebrows, nose and mouth. Given face image data, the irises can be positioned with an uncertainty of 3 mm in their location. This iris allocation algorithm uses a correlation technique using brightness distribution data. Referencing to the iris positions, the image processing procedure will continue with finding sub-areas containing the facial organs described above with 100% certainty. For each found sub-area, there is an algorithm to extract the contour lines of the facial organ. Following, the algorithm for finding the contour of an eyebrow is described. 1) Reinforcement of the horizontal edge, (of the original image) 2) template matching for eyebrow positioning, 3) and 4) lower and upper edges of eyebrow are approximated by quadratic curves, 5) inner and outer parts of the closed contour of eyebrow are binarized in terms of black and white, 6) find the maximum dark area by labeling technique.

1.1.6 Feature point tracking by optical flow in facial expression

Kanade et al. [Kana98] developed and implemented an optical-flow based approach (feature point tracking) to capture the full range of emotion expression. This approach is sensitive to subtle changes in facial expressions. Because face position in an image sequence may be slightly transformed a transformation is adequate to normalize the face position. The positions of the feature points are normalized by automatically mapping them to a standard face model. This model is based on three facial feature points: the medical canthus of both eyes and the uppermost point of the philtrum. In the facial feature point tracking phase, key feature points are manually marked in the first digitized frame with a mouse around facial landmarks. The facial feature points within 13x13 pixel windows are tracked by using a hierarchical optical flow method. The displacement of each feature point is calculated by subtracting its current normalized position by its normalized position in the first frame.

1.2 Thesis overview

The structure of this thesis report is chosen so that chapter 2 discusses the problem definition and the thesis assignment. In the following chapter (chapter 3) models and algorithms, which gives an important overview of our models, will be presented. In chapter 4 and 5 the discussion continues with the face detector. All techniques, methods and algorithms used will be considered in more detail, for instance the Adaboost algorithm and RVM classification model. After these chapters the second part of this project will be presented. It starts with the examination of corner detectors

and then using these corners detectors to extract the characteristic points or landmarks in chapter 6. Further classification of the detected corners is needed, which leads to the extraction of relevant corner feature combined with the RVM classificator in chapter 7. Chapter 8 proceeds with a technique called the projection method. This is used to track other remaining corners that cannot be found with the corner detector algorithm. The analysis, design and implementation of the prototype which demonstrates the result of our findings is included as chapter 9. Chapter 10 is engaged with the test results of the prototype. We will conclude this thesis report with conclusions, discussions and recommendations for future works.

2

Problem Definition and Thesis Assignment

This chapter describes the outline of our thesis. This thesis-project can be considered as the extension of a previous work, named FED (an online Facial Expression Dictionary [Jong02]) concerning a nonverbal dictionary. As the name already points out, FED is an online application and it can be accessed via the World Wide Web. First, FED will be described in section 2.1. After that, the objectives and the scope of our research will be defined in section 2.2 and 2.3. In the last section, we will define our thesis assignment.

2.1 Facial Expression Dictionary – FED

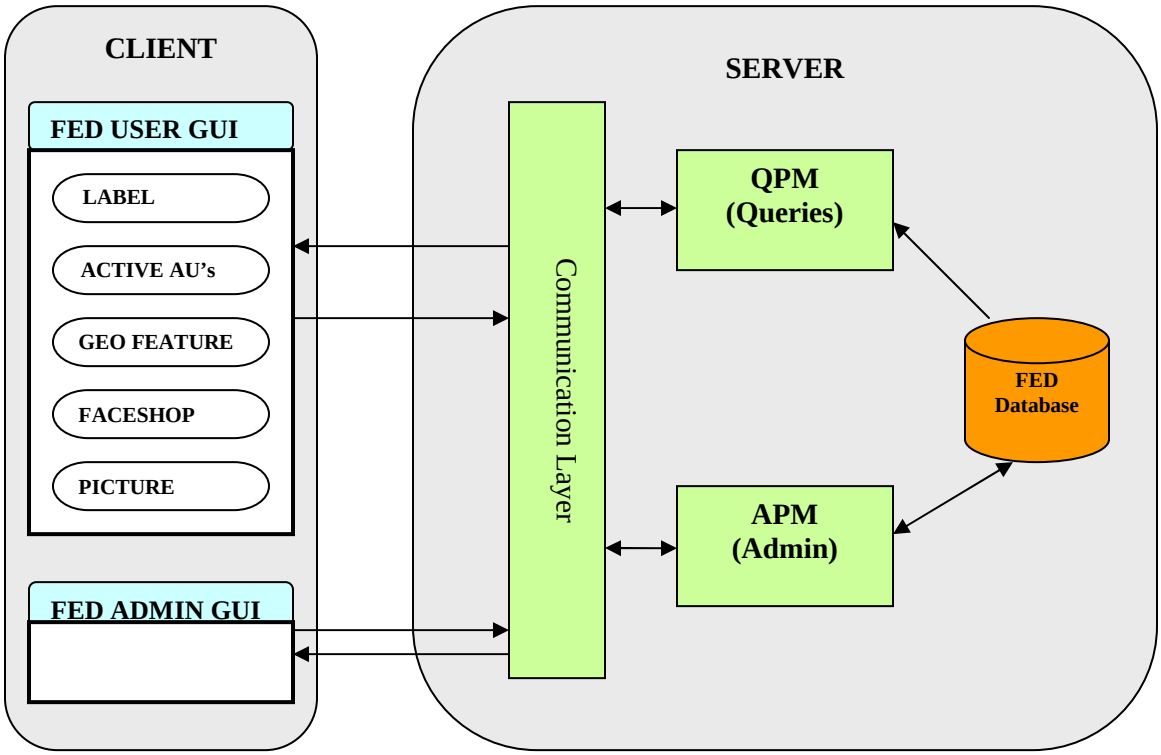


Figure 7: Global design of the FED system

A verbal dictionary can be described as a tool that aims to provide a partial solution for the problem where two persons neither understand the language the other is speaking but still want to communicate. One can just look up the meaning of the words of another language. A nonverbal dictionary has the same concept of a verbal dictionary, but it differs in the type of information that is stored. Instead of words, a nonverbal dictionary contains information about all the ways people communicate with each other nonverbally: facial expressions, gestures, posture, eye movement and contact, speech rate, loudness, pitch, tone of voice and the placing of inflections.

The FED system is an online nonverbal dictionary. Figure 7 illustrates how the FED system works. Currently, there are several ways to find an entry in FED. It is possible to have a label, active action units, geometric features, a FaceShop-generated facial expression or an image as input. A label is the specification of the expression by a keyword. Examples of label queries are *happiness*, *stressed*, *surprised*, and so on. The keyword will be matched on three parts: the facial expression label, the label synonyms and the description. Analogous to the label queries, a user can look for FED entries by specifying which action units are active with the facial expressions he is looking for. Also possible is to specify some specific geometric features like *mouth open*, *eyes closed* or *mouth open AND eyes closed*. If a certain facial expression is unknown, the user can use FaceShop to sketch the facial expression. The facial characteristic points (FCP) are determined automatically while sketching. Another option to query FED entries is to submit a picture of a face. The user has to mark the FCPs manually. The results for all queries are of the same form and are illustrated in Figure 8.

Figure 8: Result of FED. The result panel displays the result in the same panel with the original image and other details.

FED handles query requests via client-server architecture. Figure 7 shows the global design of the FED system. Below a short description of the individual components of FED are given.

- The FED user GUI (Graphical User Interface) enables the users to issue queries into FED. It consists of a number of HTML pages and Java applets for handling each of the query alternatives.
- The FED administrative GUI provides the GUI for the management part of the FED system.
- The communication layer of the FED system resides on the server and handles all data traffic between the client and the server.
- The Query Processing Module (QPM) takes care of all kind of queries that are issued by the user.
- The Admin Processing Module (APM) implements the functionality needed to manage the FED system.
- The FED Database contains all the entries in the dictionary, admin user information, and log info. The PostGreSQL database management system is used to implement the database.

With this description of the FED, we can continue with the problem definition to see what our research project will change in FED.

2.2 Research question and objectives

Before we define our research question and objectives, we summarize the idea of a specific part of FED. We only focus on that part of FED, which allows the user to send a picture. This image input will be labelled by emotional word (happiness, sad, etc.). FED requires the user to manually locate the face and facial characteristic points (FCPs). The FCPs are predefined conform the Kobayashi and Hara face model. After manually selecting and submitting the points an emotional word will be output. Thus, FED lacks the ability of automatic extraction of facial characteristic points that are needed for the facial expression recognition process. In the current situation user interaction is needed to complete the whole procedure (see Figure 9).

This research is a first step towards a fully automatic emotion recognition system. We define a fully automatic emotion recognition system as: given an input image containing one or more human faces the corresponding emotions are output without any further interaction from the user.

Since this research can be seen as an elaboration of the FED we will concentrate only on the specific parts of the framework to make fully automatic emotion recognition as much as possible.

Our goal is thereby to fully automate the face detection and the extraction of facial characteristic points. These characteristic points depend on the face model that is used in the emotion analyzing module in FED. In this case the Kobayashi and Hara face model is used. Further, we want to mention that this fully automatic emotion recognition system is built on a relative new and promising classifier RVM. The use of RVM for face detection is described in our previous work [Chan04]. In the following we will define the research question and the research and implementation objectives of this thesis project.

Research question:
Our research question is defined as follows:

> "*How to realize a fully automatic facial expression recognition system using a sparse learning Relevance Vector Machine?*"

Research objectives:

- Face detection as a first step to automatic emotion recognition; it is important that it is fast and robust. How can this be achieved? What are the requirements for robust face detection?
- What are the difficulties for detecting/extracting facial characteristic points from images?
- RVM is applied in the different phases of the project. What can be said about the performance of RVM?
- What other techniques should be combined with RVM to make automatic face and FCP detection possible? How is the performance of the final system?

Implementation objectives:

- The final system can be built within the FED framework. Figure 10 illustrates the new working/scheme of FED after the implementation. Users would no longer be required to select feature characteristic points manually. The final software consists of three modules that can be integrated into the FED framework: face detection module, facial point extraction module and classification module. Since the FED was built with java, we will also implement the system/module in java/java2. We attempt to build the module as compatible as possible to the FED so that minimal adjustment to the existing FED code is needed. Note that the actual integration of these modules within the FED framework will not be done by us during this project.

- Build a prototype to test the modules.

- The face detection module is able to detect 80 % of all frontal views of human faces in the input image. This is constrained by the minimal resolution of the face which must be at least 19x19 in the original input. For computational cost reduction a dimension of 24x24 pixels of the original input is considered.

- Characteristic point extraction module should be able to detect at least 80% of all characteristic points.

2.3 Scope of the research

This thesis project focuses on the realization of a fully automatic recognition system based on the FED framework. This includes all aspects that are related to face detection and facial characteristic point extraction. For face detection, in our case it means the selection of a suitable face database, the training and testing algorithms and optimizing RVM's performance by tuning its parameters. The same facets are also applicable for facial characteristic point extraction.

After the facial characteristic points (FCPs) are extracted from the face image, a facial expression can be determined. The latter is done by the existing FED emotion classification module according to the FED framework. This also means that this project is limited to automatic face detection and facial characteristic points' extraction.

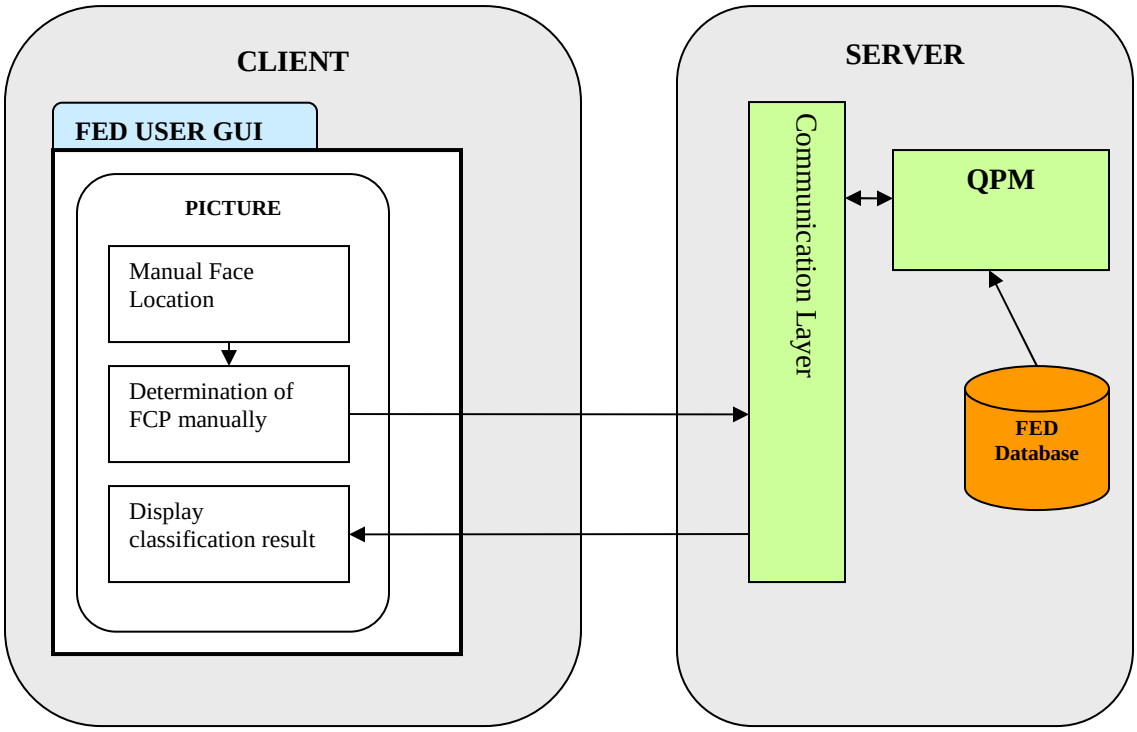


Figure 9: FED system.

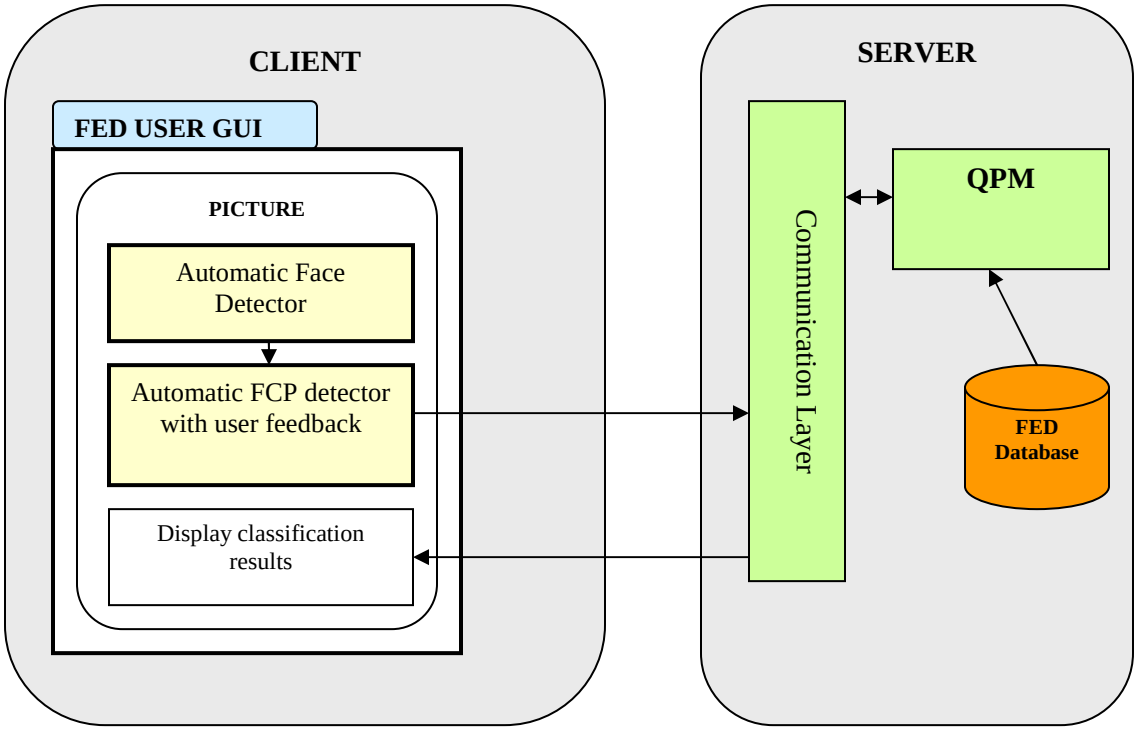


Figure 10: New scheme for the FED system. Manual face detection and FCP location is no longer required.

2.4 Thesis assignment

To conclude this chapter, we define our thesis assignment. It consists of different parts and the summary of these are listed below:

- *Literature survey*: a research on related works on the topics of face detection, facial characteristic point detection, facial expression recognition and classification methods.

- *Model design*: design a model as a solution to the problem of automatic facial characteristic point detection. This model is built of multiple methods and algorithms.

- *Prototype*: implement the designed model.

- *Tests*: write a test plan to test the prototype and depicts the results.

- *Scientific paper*: summarize this thesis project.

3

Models and Algorithms

In this chapter, we present the models that we designed to detect faces and facial characteristic points automatically. This chapter functions as an overview of the methods and algorithms that are used in our model. Detailed explanation of these will be given in the appropriate sections of the next chapters. Section 3.1 illustrates the WUXTRAP model which is the training model for face detection and FCP detection. It generates data that is needed for the FLEX application, which will be explained in section 3.2. In the last section, the face model used for WUXTRAP and FLEX is described.

3.1 WUX-values Training Application (WUXTRAP)

For the purpose of face detection we studied the object detection system of Viola and Jones [Viol01]. It is presented as a very fast and robust real-time object detection system. According to the test results it outperforms many other systems on accuracy and speed. The main drawback of this system is the training time, which is extremely long. To avoid this problem, we have implemented the genetic algorithm (Evolutionary Search) that was described in [Trep03]. For the extraction of the FCPs we used a corner detection method. Haar features are extracted from the detected corners and passed to the classifier to determine whether the detected corner is one of the desired FCP.

The model described in this chapter results in two applications: WUXTRAP and FLEX. WUXTRAP is the training application (WUX-values Training Application): it contains the training model that includes AdaBoost, Evolutionary Search and the RVM classification model. WUXTRAP selects the proper features that will be used in FLEX (Facial Landmark Extraction) to classify the given input images. WUX stands for three kinds of data that are needed to build FLEX's classifiers. These are generated during the training phase:

- W stands for the weights.

- U stands for the used samples (relevance vectors).
- X stands for the input training data.

These WUX values are used by the relevance vector machine when performing its classification task. WUXTRAP consists of two independent modules. Both modules aim for the same kind of results which are Haar features and the corresponding WUX values. As a result, the training algorithms used in both modules are the same. The modules are combined together and shown in Figure 11. It depicts the general scheme of WUXTRAP.

In the face detection module, a labelled set of face images and a labelled set of non-face images function as input. Both sets need to be converted to the integral image representation, which offers the advantage of very fast feature evaluation. The images are evaluated against a huge set of generated Haar-like features (this evaluation procedure will be explained in detail in section 4.3). Note that the set of Haar features for faces and corners are not the same. AdaBoost in combination with the relevance vector machine trains and selects the best features that can distinguish faces from non-faces. Because of the huge number of Haar-like features that needs to be trained and evaluated, a genetic search algorithm is incorporated to improve the speed of this procedure. The emphasis in Evolutionary Search (ES) lies on natural selection and survival of the fittest (see section 4.5). The combination of these three processes is named EABoost. The features selected by EABoost will be evaluated against the test set. If the features perform well and achieve the proper detection rate, they will be added to the final set of features. The result of this face detection training module will be a strong set of features that is able to distinguish faces from non-faces. The WUX values belonging to these features will be stored.

In the FCP training module (also shown in Figure 11) different sets of images of facial characteristic points serve as input. To be more precise, for each FCP a set of images of that FCP is needed. In Figure 11 these sets are named REIC (right eye inner corner), MLC (mouth left corner), NLC (nose left corner), etc, conform to their position in the face. These sets of FCP images are manually extracted from the BioID and Carnegie Melon face database. For each FCP there is also a set of non-FCP images. Note that only one FCP can be trained at a time. So there is actually one RVM for one FCP.

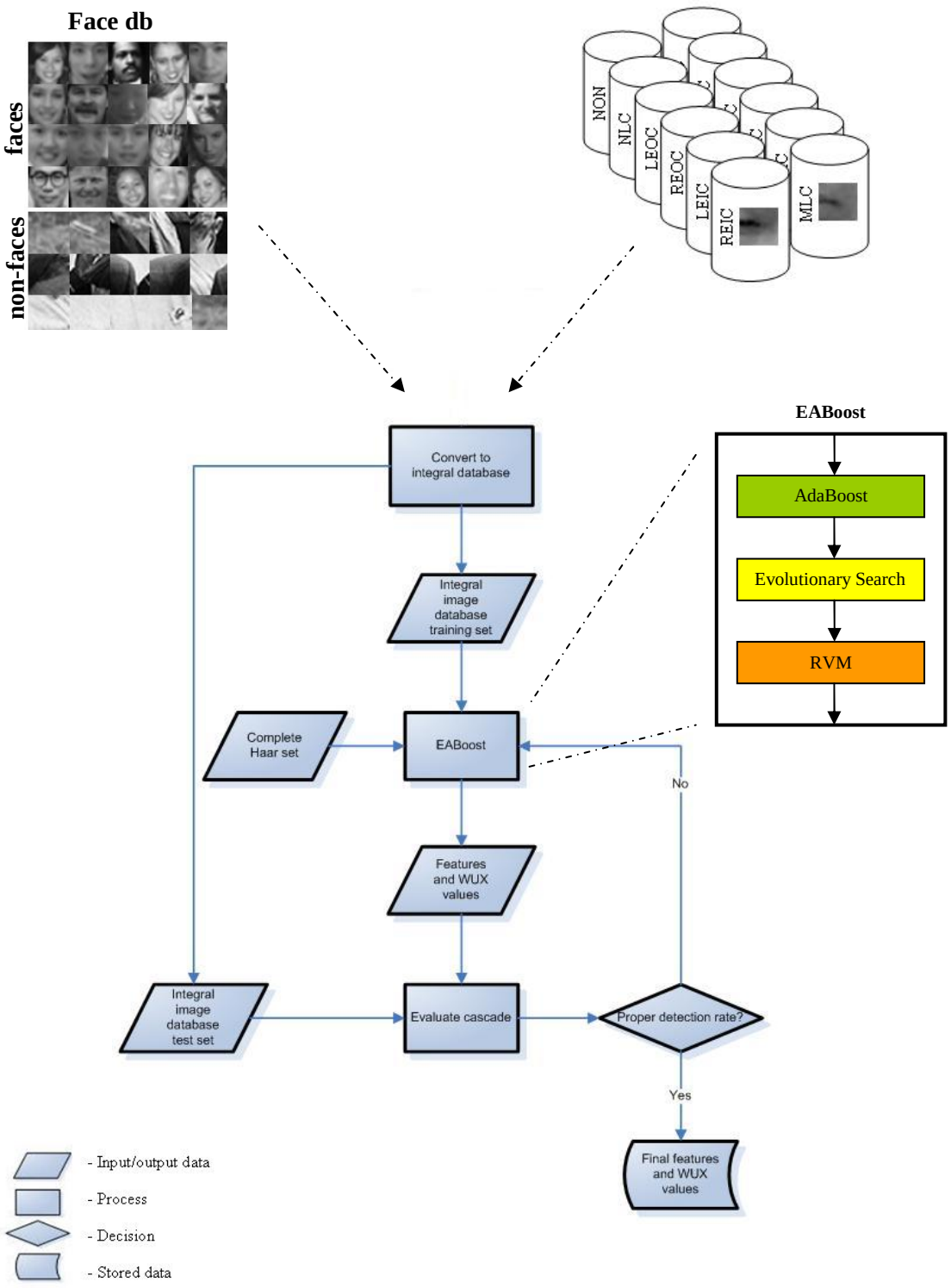


Figure 11: General scheme of WUXTRAP.

3.2 Facial Landmarks Extraction (FLEX)

Figure 12 depicts the general scheme of FLEX. FLEX is the running application that is able to detect faces and detect FCP from face images. An input image can be selected from the graphical user interface. After selection, the image processing component checks whether or not it is necessary to resize the image or to convert the image into gray-scale. Then the image is scanned on different resolutions for face detection. It does so by using the features and WUX values obtained from WUXTRAP. It scans by applying the strong Haar-features on a small part of the image and evaluating it with RVM. If faces are found, the proper ones will be passed to the FCP detection component for FCP extraction and FCP detection. With the proper ones, faces with a resolution greater than 64x64 pixels are meant. There are two running processes in this component. A corner detection algorithm, either Harris (see section 6.2) or Sojka (see section 6.3) or both will be applied on a face. An image of a region containing the corner point in the centre will be cut out. The image around this corner will be classified with RVM to determine whether or not the detected corner is a FCP. Besides the corner detectors, a hybrid projection method (HPM, see chapter 8) will also be used to extract FCP candidates from faces. This HPM method must be applied on a facial feature (eye, eye brow, etc) which is extracted from the face using a RVM trained with proper facial features. FLEX ends with showing its results in the graphical user interface.

3.3 Face model

FLEX's objective is to extract FCPs from a face. FCP extraction can be defined as the process of finding the facial features of a face model. The face model outlines the facial features of a generic face. There exist several face models like 3D wire-frame models and 2D face models. 3D wire-frame models are known too be very complex and very time-consuming in the construction of the model. 2D face models on the other hand are rather simple but not very efficient given the fact that 3D information of the face is lost. To overcome this problem there exist several systems which defines the face model as a point-based model composed of two 2D facial views, namely the frontal and the side view. Combining a dual view into a single model yields a more realistic representation of 3D face. The FCPs that FLEX has to extract is defined by the face model of Kobayashi and Hara [Hara97]. The reason for this face model is that it was already used in FED.

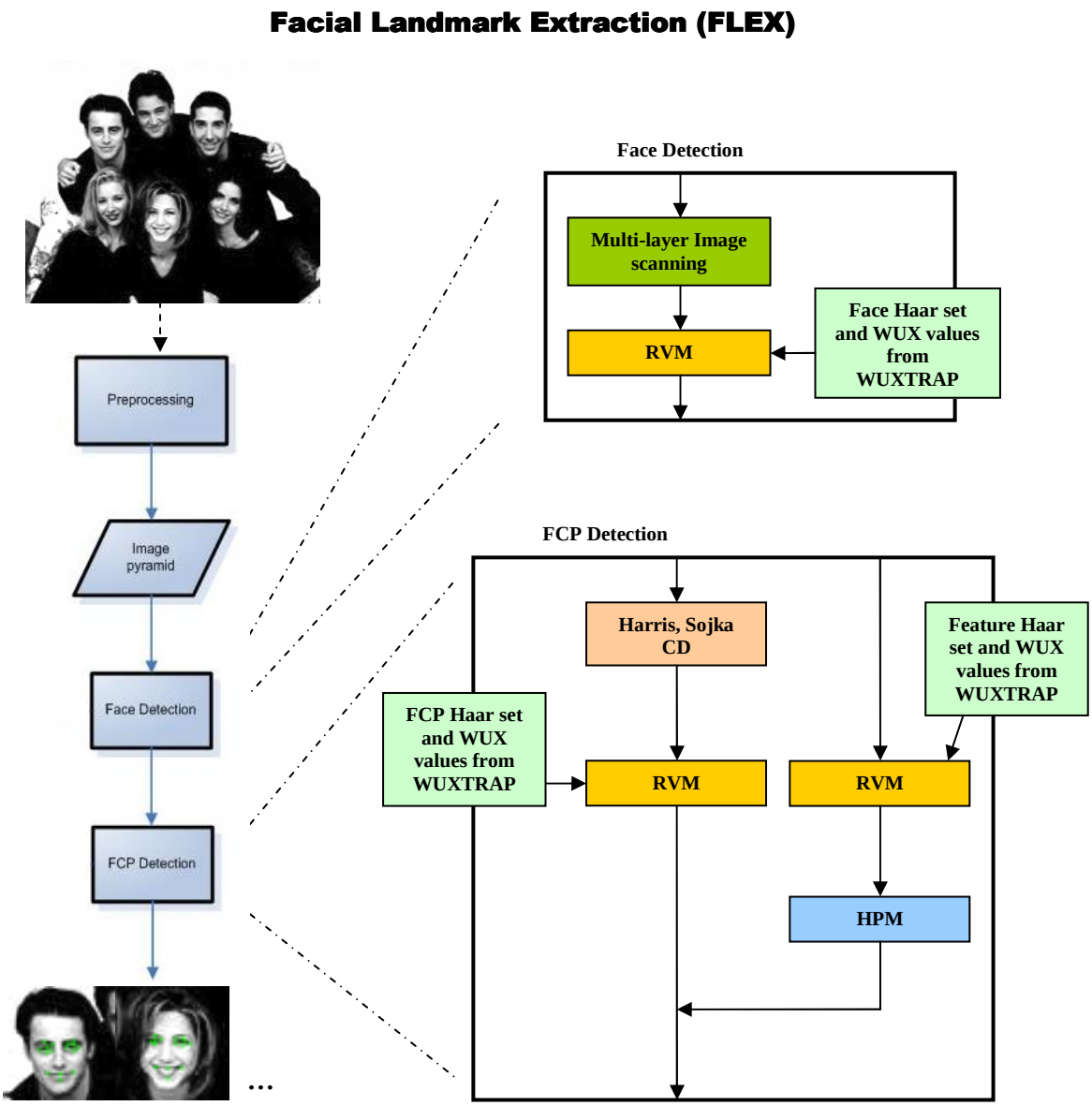


Figure 12: General scheme of FLEX.

Kobayashi and Hara model the face through 30 FCPs. These 30 FCPs correspond to 30 of the 44 Action Units (AUs) of the FACS [Ekma78] system. The intention of FACS was to objectively represent facial expression information. The 30 AUs chosen by Kobayashi and Hara are related to the contours of the eyes, eyebrows and mouth. It was not needed to use for example the points around the cheek and chin, because experiments have shown that people only pay attention to the position and size of the eyes, eyebrows and mouth when classifying facial expressions. Figure 13

shows the position of these 30 FCPs. The vertical lines in the figure are the so-called *haralines*. The positions of the haralines are fixed and depend on the position of the FCPs a1, a2, a3, a4 (corners of the eyes) and FCP a19 and a20 (inner corners of the eyebrows). The x-coordinates of all the other FCPs are fixed depending on the position of the haralines. This is a property of the face model of Kobayashi and Hara.

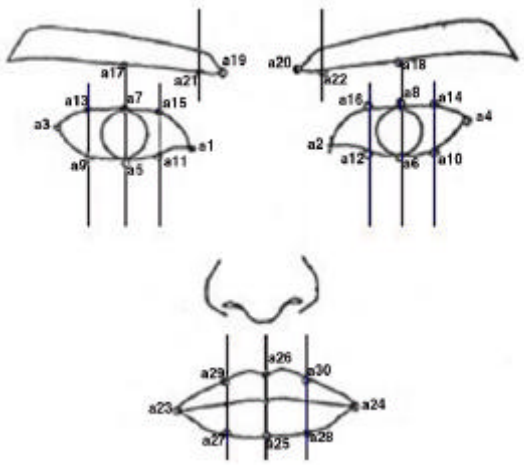


Figure 13: The Kobayashi and Hara face model used in the FED.

Table 1: Description of the face model points.

	Point description	State		Point description	State
A1	Left eye inner corner	stable	A19	Left eyebrow inner corner	non-stable
A2	Right eye inner corner	stable	A20	Right eyebrow inner corner	non-stable
A3	Left eye outer corner	stable	A23	Left corner of the mouth	non-stable
A4	Right eye outer corner	stable	A24	Right corner of the mouth	non-stable
A5	Bottom of the left eye	non-stable	A26	Top of the upper lip	non-stable
A6	Bottom of the right eye	non-stable	A25	Bottom of the lower lip	non-stable
A7	Top of the left eye	non-stable			
A8	Top of the right eye	non-stable			
-	Left nostril centre	non-stable	-	Left eyebrow outer corner	non-stable
-	Right nostril centre	non-stable	-	Right eyebrow outer corner	non-stable

Part II

Face Detection

4

Face Detection – Methods and Tools

This chapter explains the components of our model introduced in the previous chapter for face detection in detail. The relation between the different components was shown in Figure 11 of the previous chapter. First the theory of RVM will be described in section 4.1. RVM is the main classificator that is used throughout the whole paper. Section 4.2 discusses our initial ideas and attempts to detect faces using RVM. Unfortunately, it does not work as we hoped. So, there is a need to look further for other techniques and methods. These are combined in our WUXTRAP model, which are the Haar-like features set and integral image representation, the AdaBoost learning algorithm, the genetic algorithm (Evolutionary Search) and the cascade construction of classifiers. They will be explained in the sections 4.3 to 4.6.

4.1 Relevance Vector Machine (RVM)

[Tipp01] RVM is a Bayesian approach to pattern recognition in the context of regression and classification problems. It can be seen as a probabilistic version of the Support Vector Machine (SVM). SVM is known as a very good classifier. It has a lot of applications like face detection and handwriting recognition. On the other hand, RVM has the compelling feature that, while capable of generalization performance comparable to an equivalent SVM, the number of relevance vectors used by RVM is in most cases dramatically smaller than the number of support vectors used by SVM to solve the same problem (see Figure 14). This means that the computation costs are reduced. On the same time, RVM offers a number of additional advantages, which include the benefits of probabilistic predictions, automatic estimation of parameters and the facility to use arbitrary basis functions, which are not necessary ‘Mercer’ kernels. This chapter starts with the RVM regression model upon which the RVM classification model is based. Then the modifications required in the case of classification will be described.

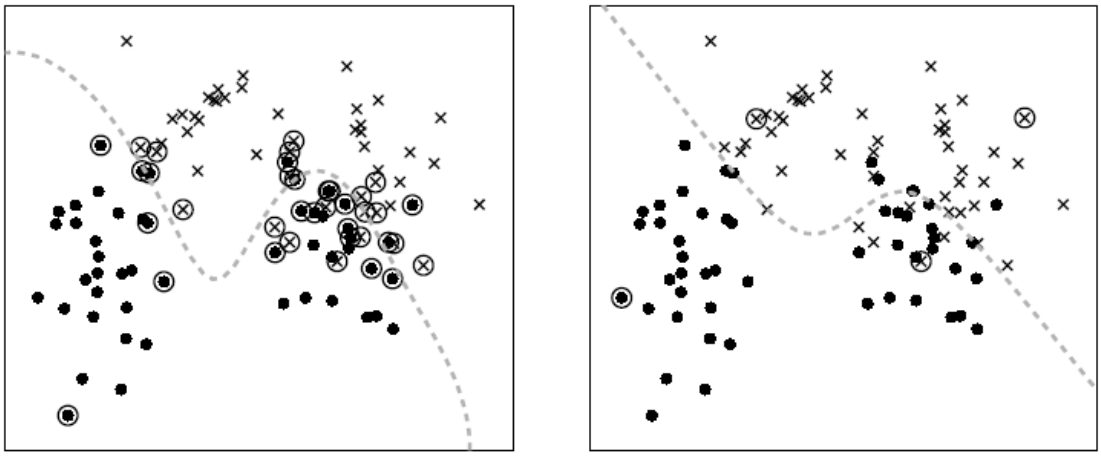


Figure 14: SVM (left) and RVM (right) classifiers on 100 examples from Ripley's Gaussian-mixture data set. The decision boundary is shown dashed, and relevance/ support vector are shown circled to emphasize the dramatic reduction in complexity of the RVM model.

RVM Model Specification

[Bish04, Tipp01] Like in supervised learning, a set of example input vectors $\{\mathbf{x}_n\}^N_{n=1}$ is given along with a corresponding set of targets $\mathbf{t} = \{t_n\}^N_{n=1}$. These targets will be real values in the case of regression and class labels in the case of classification. This set of input vectors and targets is called the 'training set' from which we wish to learn a model of dependency. The objective is to make accurate predictions of t for previously unseen values of $\mathbf{x}$. Assuming that the targets are some noisy realization of an underlying functional relationship $y(\mathbf{x}_n; \mathbf{w})$ that we want to estimate, the desired model of dependency can be described as $t_n = y(\mathbf{x}_n; \mathbf{w}) + \varepsilon_n$, with ε_n representing noise from a mean-zero Gaussian process with variance σ^2 and $\mathbf{w}$ a vector of adjustable parameters or 'weights'. Thus, $p(t_n \mid \mathbf{x}) = N(t_n \mid y(\mathbf{x}_n; \mathbf{w}), \sigma^2)$ where the notation specifies a Gaussian distribution over t_n with mean $y(\mathbf{x}_n; \mathbf{w})$ and variance σ^2.

A popular class of candidate functions for $y(\mathbf{x}_n; \mathbf{w})$ is that of the form

$$y(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \qquad \text{(Eq. 4.1)}$$

where the output is a linearly weighted sum of M generally non-linear and fixed basis functions denoted by $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_M(\mathbf{x}))^{\mathrm{T}}$. A basis function is defined as $\phi_i(\mathbf{x}) \equiv K(\mathbf{x}, \mathbf{x}_i)$ with the kernel parameterised by the training vectors, so that $y(\mathbf{x}_n; \mathbf{w})$ becomes

$$y(\mathbf{x_n};\mathbf{w}) = \sum_{n=1}^{N} \mathbf{w}_i K(x, x_i) + \mathbf{w}_0 \qquad \text{(Eq. 4.2)}$$

Due to the assumption of independence of the $\mathbf{t}_n$, the likelihood of the complete data set can be written as

$$p(\mathbf{t} \mid \mathbf{w}, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}\|\mathbf{t} - \mathbf{\Phi}\mathbf{w}\|^2\right\} \qquad \text{(Eq. 4.3)}$$

where the N x $(N+1)$ matrix $\mathbf{\Phi}$ (see Eq. 4.4) is called the *design* matrix, $\mathbf{t} = (t_1 \cdots t_n)^{\mathrm{T}}$ and $\mathbf{w} = (w_0 \cdots w_n)^{\mathrm{T}}$.

$$\mathbf{\Phi} = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), ..., \phi(\mathbf{x}_N)]^{\mathrm{T}} = \begin{bmatrix} 1 & K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ 1 & K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \qquad \text{(Eq. 4.4)}$$

The next step is to consider over-fitting. SVM avoided this problem by the inclusion of the 'margin term'. RVM approaches this problem by the introduction of an explicit prior probability distribution over the parameters. The authors choose for the smooth zero-mean Gaussian prior distribution over $\mathbf{w}$:

$$p(\mathbf{w} \mid \boldsymbol{\alpha}) = \prod_{i=0}^{N} N(w_i \mid 0, \alpha_i^{-1}), \qquad \text{(Eq. 4.5)}$$

with $\boldsymbol{\alpha}$ a vector of $N+1$ hyperparameters such that each α_i is associated independently with every weight.

To continue with the inference process, hyperpriors over $\boldsymbol{\alpha}$ must be defined, as well as over the noise variance σ^2. Suitable priors for these parameters are given by Gamma distributions:

$$p(\boldsymbol{\alpha}) = \prod_{i=0}^{N} \mathrm{Gamma}(\alpha_i \mid a, b), \qquad \text{(Eq. 4.6)}$$

$$p(\beta) = \text{Gamma}(\beta \mid c, d), \qquad \text{(Eq. 4.7)}$$

with $\beta \equiv \sigma^{-2}$ and where

$$\text{Gamma}(\alpha \mid a, b) = \Gamma(a)^{-1} b^{a} \alpha^{a-1} e^{-b\alpha}, \qquad \text{(Eq. 4.8)}$$

in which $\Gamma(a) = \int_0^\infty t^{a-1} e^{-t} dt$ is the gamma function. It is assumed that $a = b = c = d = 0$. This makes the hyperpriors uniform. As a result, predictions are independent of linear scaling of both $\mathbf{t}$ and the basis function outputs.

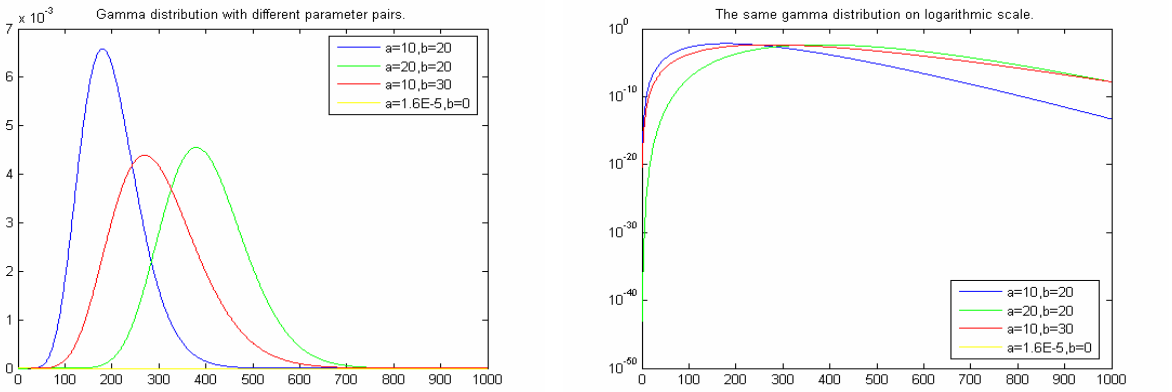


Figure 15: Gamma distribution. (Left) Gamma distribution with different parameters. (Right) Same distribution on logarithmic scale.

The choice of prior distributions is related to those used in Automatic Relevance Determination. The idea behind it is that if a basis function provides no information, because it is irrelevant to the problem, there is no value of the weight that will lead to a significant increase in the likelihood. At this point, the prior term can show its usefulness. By setting the α_i parameter to a large value, the prior distribution $p(w_i \mid \boldsymbol{\alpha})$ becomes sharply peaked around zero. By then setting w_i to zero, the posterior[1] probability of the model is maximized. In a word, when a basis function has sufficient high α, it can be marked as 'low relevance' and thus will be removed from the model.

[1] Given the prior distribution, data needs to be collected to obtain the observed distribution. Then calculate the likelihood of the observed distributions as a function of parameter values, multiply this likelihood function by the prior distribution, and normalize to obtain a unit probability over all possible values. This is called the posterior distribution. From: http://www.mathworld.com

Consequently, the RVM will learn simple models even when presented with a large starting set of basis functions.

To make a prediction for target t_*, given new input data $\mathbf{x}_*$, we have to find out $p(t_* \mid \mathbf{t})$ which can be expressed as:

$$p(t_* \mid \mathbf{t}) = \int p(t_* \mid \mathbf{w}, \boldsymbol{\alpha}, \sigma^2)\, p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2 \mid \mathbf{t})\; d\mathbf{w}\; d\boldsymbol{\alpha}\; d\sigma^2 . \qquad \text{(Eq. 4.9)}$$

Since these computations cannot be performed fully analytically, an effective approximation is needed. This is done by applying the Bayes rule. After some substitutions and decompositions (discussed in [Tipp01]), this posterior distribution can be computed analytically and the result becomes:

$$p(\boldsymbol{\alpha}, \sigma^2 \mid \mathbf{t}) \propto p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2)\, p(\boldsymbol{\alpha})\, p(\sigma^2) \qquad \text{(Eq. 4.10)}$$

The idea of relevance vector learning is actually the search for the hyperparameter posterior mode: the maximization of Eq. 4.10 with respect to $\boldsymbol{\alpha}$ and β. The prediction of a target t* can then be given by Eq. 4.11 in which $\boldsymbol{\alpha}_{MP}$ and σ^2_{MP} are the most probable values for $\boldsymbol{\alpha}$ and σ^2.

$$p(t_* \mid \mathbf{t}, \boldsymbol{\alpha}_{MP}, \sigma^2_{MP}) = \int p(t_* \mid \mathbf{w}, \sigma^2_{MP})\, p(\mathbf{w} \mid \mathbf{t}, \boldsymbol{\alpha}_{MP}, \sigma^2_{MP})\, d\mathbf{w} \qquad \text{(Eq. 4.11)}$$

Because both terms in the integrand are Gaussian, the result can be computed by:

$$p(t_* \mid \mathbf{t}, \boldsymbol{\alpha}_{MP}, \sigma^2_{MP}) = N(t_* \mid y_*, \sigma^2_*); \qquad \text{(Eq. 4.12)}$$

with

$$y_* = \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_*) \text{ and } \sigma^2_* = \sigma^2_{MP} + \boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{\phi}(\mathbf{x}_*) . \qquad \text{(Eq. 4.13, 4.14)}$$

where $\boldsymbol{\Sigma}$ the is the posterior covariance and $\boldsymbol{\mu}$ is the mean.

$$\boldsymbol{\Sigma} = \left(\sigma^{-2} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \mathbf{A}\right)^{-1}, \qquad \text{(Eq. 4.15)}$$

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \qquad \text{(Eq. 4.16)}$$

with $\mathbf{A} = diag(\alpha_1, \alpha_2, \ldots, \alpha_N)$.

RVM Classification

RVM is using an identical framework as detailed for regression in the case of classification. Only some modifications need to be made. To account for the changes in the target quantities, the authors use Bernoulli likelihood and a sigmoid link function $\sigma(y) = 1/(1+e^{-y})$ (see Figure 16). As a consequence, there is an additional approximation step in the algorithm.

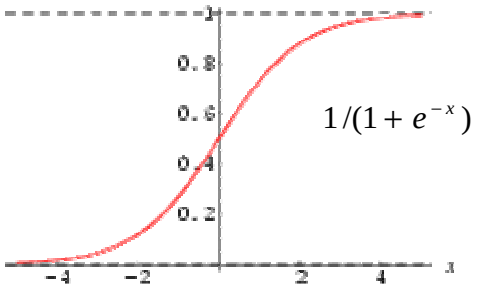


Figure 16: Sigmoid link function used in classification model, 1/(1+exp(-y)).

In the two-class case, applying the sigmoid link function to $y(\mathbf{x})$ and adapting the Bernoulli distribution for $P(\mathbf{t} \mid \mathbf{x})$, the likelihood can be written as:

$$P(\mathbf{t} \mid \mathbf{w}) = \prod_{n=1}^{N} \sigma\{y(\mathbf{x}_n; \mathbf{w})\}^{t_n} \left[1 - \sigma\{y(\mathbf{x}_n; \mathbf{w})\}\right]^{1-t_n}, \qquad \text{(Eq. 4.17)}$$

where, following from the probabilistic specification, the targets $t_n \in \{0, 1\}$. Note that there is no 'noise' variance here or we may assume that it is already included in the link function.

Unlike in the regression case, it is not possible to integrate out the weights analytically. Therefore, the authors use an approximation procedure based on a combination Laplace's method and Newton's 2nd order method. The outcome of this approximation procedure is a mapping of the classification problem to a regression problem with data-dependent noise.

In the multi-class classification case, where the number of classes K is greater than two, the likelihood (Eq. 4.3) is generalized to the standard multinomial form:

$$P(\mathbf{t} \mid \mathbf{w}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \sigma\{y_k(\mathbf{x}_n; \mathbf{w}_k)\}^{t_{nk}} \quad \text{(Eq. 4.18)}$$

where a conventional 'one-of-K' target coding for t is used and the classifier has multiple outputs $y_k(\mathbf{x}; \mathbf{w}_k)$, each with its own parameter vector $\mathbf{w}_k$ and associated hyperparameters.

4.2 Face detection – the initial idea

Most facial expression recognition systems assume the input to be an image containing only the face surrounded by a simple background. If the image does not contain a face will it still work properly? Such systems are also lacking the ability to extract multiple faces from an input image that contains more than one face. Since we do not want this restriction, the system should be able to detect multiple faces in the image. To achieve this we need a face detector to determine all faces from the input. This face detection module must precede the extraction of features for emotion recognition. In order to detect faces we need to train a classifier. In this case we want to differentiate faces from non-faces, which means that our training set should contain faces and non-faces. The classifier we are using is RVM. As a first attempt, we trained the RVM with intensity values. We used a subset from the MIT-CBCL database consisting of 1000 faces and 1500 non-faces of 19x19 (see Figure 17). So, each sample is arranged as a training vector of 1 by 361 when given to the RVM for training. Some sample images from MIT-CBCL are shown in Figure 17. This subset is used for both training and testing. There are a few ways to do that. One way is to use the whole set for training and also testing.

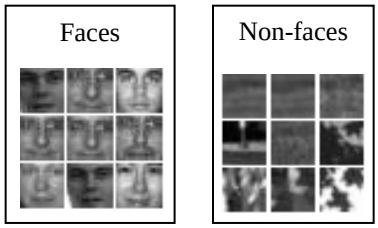


Figure 17: Samples from the MIT-CBCL face database.

However, this method is not very reliable because we cannot see how the trained classifier will behave for untrained samples. To improve this, testing should be done on a set different than the classifier is trained with. Another training method which is also more common is K-fold cross validation. Before passing the samples to the RVM, the data set is first divided into K sets of even size. Then RVM is trained on the K-1 sets of samples and testing is done on the remaining set.

This is done for all K sets. The choice for K depends on the size of the dataset. For large datasets, K should not be chosen too small since it is really ineffective. For cross validation the time to train RVM grows linearly with the size of the dataset. Usually K-fold cross validation is performed for parameter optimization. Since we first want to know if RVM can be trained with intensity values, we simply choose for the first method. We train and test RVM on the exact same set. The results are shown in Table 3.

Table 2: MIT CBCL subset specification.

Dataset parameters	Values and description
Database	MIT CBCL
Sample size	19x19 pixels
Number of classes	2
Class 0	Non-faces
Class 1	Faces
Number of samples (0/1)	1000/1500

Table 3: RVM test results: training and testing is done on the same set.

Kernel	Nr of test samples	Detection rate %	Nr of false negatives	Nr of true negatives	Nr of false positives	Nr of true positives
Gauss: 0.5, 1.0, 2.0, 3.0, 4.0, 5.0	2500	67.96	801	1000	0	699
Laplace: 0.5, 1.0, 2.0, 3.0, 4.0, 5.0						

Training RVM with only the intensity values gives unsatisfactory results. Not only does it take a long time to train, but the detection rate is also poor, only 67.96%. This is certainly not good enough for a face detection system. In addition, as Table 3 shows, the detection rate of 67.96% is achieved only due to the high true negative rate. True negative means that a non-face image is indeed recognized as a non-face by RVM. False positive is the opposite: a non-face is recognized as a face. In the test, only 699 of the 1500 faces are recognized as a face. So, the detection rate is calculated as (1000 + 699)/2500 = 0.6796. If the test is done on only faces, the detection rate would be 699/1500 = 0.466. This means that RVM is not able to be trained this way. By arranging each training vector of 19x19 in a 1x361 vector, all links between the pixels are totally lost. As a result, there are no consistencies in the input samples and thus, there is no difference between the positive and negative samples. This explains the 0 in the column of false positives. It also explains why the results are the same for all kernels. The raw data associated with each pixel

are insufficient to allow unambiguous identification of that pixel. We can conclude that the detection rate from this table does not have any meaning.

RVM is known for the relatively slower performance during training which means that it will take a long time to train on a large dataset with a dimension space of 361, which is relatively high. On a windows machine with AMD Athlon™ XP 2200+ 1.80 GHz processor with 512 MB RAM it was in the order of hours to train RVM on the chosen dataset.

Because of the unacceptable results, we need to consider alternatives. There are two things we must take into account: improvement of the detection rate and reduction of the training time. Apparently, using the intensity values as a feature vector for training the RVM does not work. This problem can be solved by training RVM with better features extracted from the images instead of only intensity values. Some well-known feature extraction methods are PCA [Jung01], DCT [Huan04], and HAAR [Papa98]. To reduce the training time, the dimension space of 361, which is extremely large, needs to be reduced.

We considered the PCA and DCT feature extraction method. Due to the sensitivity to illumination variance in the images these methods failed to capture the essential features of face images. The results were far from satisfactory as it is poorer than in the case of training RVM with intensity values. Next, we examined the HAAR features method and we came to the face detection method of [Viol01], on which our solution is based. This method has three key parts that make it fast and robust which are the main reasons for choosing it. The first part is a fairly simple feature extraction method that is based on a new image representation called integral image. The second is a learning algorithm, based on AdaBoost and the last one is a method for constructing complex classifiers into a cascade structure. In the following sections each of the three parts will be described in detail. Note that in the first place tests are done on 19x19 windowed pixels samples. This size is altered to 24x24 pixels in a later stage. The main reason for this is that the number of scanning windows will be reduced by using a larger sample size. As a result, the computational load will also be reduced.

4.3 Haar-like features and integral image representation

As concluded, in our findings using intensity values directly as input for the RVM classifier does not work adequate. Therefore, features need to be extracted from the images of the dataset. The features that are extracted are Haar-like features which have been used by [Papa98]. These features have a rectangular shape and are fairly simple. Compared with other filters, these features are somewhat primitive. For example, it is hard to use them for boundary analysis or

texture analysis. They are also sensitive to the presence of edges, bars, and other simple image structures. But on the other hand due to its simple construction, they have only horizontal and vertical orientation. It is computationally very efficient. This is the compensation they offer for their limited flexibility. As a result these features can be computed very fast. In our face detection algorithm, five types of rectangular features are used (see Figure 18). Type 1, 2 and 5 are calculated as the sum of all pixels in the dark area minus the sum of all pixels in the light area. Type 3 and 4 are calculated as half the sum of all pixels in both dark areas minus the sum of all the pixels in the light area in the middle.

type 1 and 2 and 5 : $$\sum pixels\_in\_dark\_area - \sum pixels\_in\_light\_area$$ (Eq. 4.19)

type 3 and 4 : $$\frac{1}{2}\sum pixels\_in\_dark\_area - \sum pixels\_in\_light\_area - \frac{1}{2}\sum pixels\_in\_dark\_area$$ (Eq. 4.20)

Here is where the integral image, an intermediate image representation, comes into play. It makes it possible to compute these features really fast. Instead of calculating the features from the original images, the features are calculated from the integral images. An integral image is actually a matrix containing the sum of pixel values from the original image. Location (x, y) of the integral image contains the sum of the pixels above and to the left of (x, y). It can be denoted as: $ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y')$ where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image (see Figure 19). Using the following pair of recurrences:

$$s(x, y) = s(x, y-1) + i(x, y) \quad \text{(Eq. 4.21)}$$

$$ii(x, y) = ii(x-1, y) + s(x, y), \quad \text{(Eq. 4.22)}$$

(where $s(x, y)$ is the cumulative row sum, $s(x, -1) = 0$ and $ii(-1, y) = 0$) the integral image can be computed in one pass over the original image.

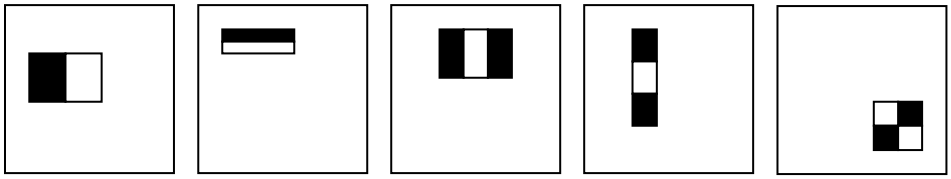


Figure 18: Example of rectangle features. The sums of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the dark rectangles.

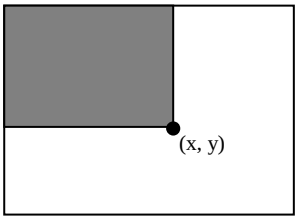


Figure 19: The value of the integral image at point (x, y) is the sum of all the pixels above and to the left.

Each of the five basic features is scanned on every possible scale and every possible position within a training sample. Given that the sample's dimension is 24x24, the complete set of features that can be constructed is tremendously large, namely 162336. Our training set consists of 3000 samples with 1500 faces and 1500 non faces (see Table 4). Each of the features is encoded as a tuple with five values: (*x_left*, *y_top*, *x_right*, *y_bottom*, *type*) in which:

- *x_left*: minimum x-value that defines the left boundary of the feature.
- *y_top*: minimum y-value that defines the upper boundary of the feature.
- *x_right*: maximum x-value that defines the right boundary of the feature.
- *y_bottom*: maximum y-value that defines the lower boundary of the feature.
- *type*: type of the feature: 1, 2, 3 or 4.

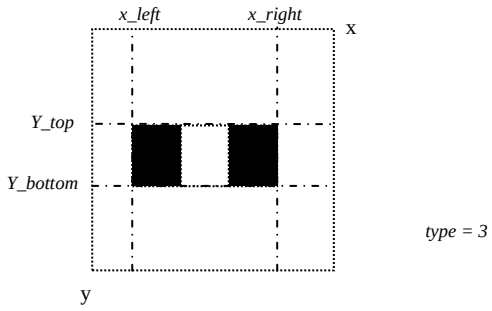


Figure 20: A Haar-like feature has five attributes: (x_left, x_right, y_top, y_bottom, type).

Table 4: Dataset used to learn the Haar-like features.

Dataset parameters	Values and description
Database	Viola Jones
Sample size	24x24 pixels
Number of classes	2
Class 0	Non-faces
Class 1	Faces
Number of samples (0/1)	1500/1500

Having defined the Haar-like features one might wonder how they can be applied for training the RVM. Assume that one Haar-feature is selected out of the total set of 162336. This feature will be applied to all images in our training set specified in Table 4. Each image produces a feature value according to Eq. 5.1 and 5.2. As a result, a labelled set of feature values is produced. Thus the size of this set is equal to the size of the training set. This set will be the input for our classifier which will be discussed more in details in the following sections. The selection of a Haar-like feature is done by AdaBoost algorithm explained in the next section.

4.4 The AdaBoost learning algorithm

In the previous section simple Haar-like features can be calculated and extracted really fast. Recall that there are more than 162336 features associated with each 24x24 sub-window. Combined together it will far exceed the number of pixels we have as feature vector in the beginning. The positive thing now is that a very small number of these features can be combined to form an effective classifier. But the challenge is to find these features. We need to train a classifier that consists of several discriminating features within a sub-window. AdaBoost is used both to select features and to train the classifier.

The AdaBoost algorithm [Freu95] is proven to boost the performance of the classifier. It is also proven that the training error of the strong classifier approaches zero exponentially in the number of rounds and the generalization performance is also very high. The AdaBoost algorithm can be interpreted as a greedy feature selection process. Consider a general boosting case where a large set of classification functions are combined using weights. The challenge is to associate a large weight with each good classification and a smaller weight with poor functions. AdaBoost is an aggressive and effective algorithm used to select a low number of good classification functions, so called ‘weak classifiers’, to form a stronger classifier. The classifier is called weak because we do not expect even the best classification function to classify the training data well. The final strong classifier is actually a linear combination of the weak classifiers.

In analogy of the AdaBoost algorithm, the weak classifier is restricted to the set of classification functions of single features. Thus, the weak learning algorithm will be designed to select a single rectangle feature which best separates the positive and negative samples. The algorithm to select a predefined number of features given a set of positive and negative samples is shown in Figure 21. As we can see in the figure the input is a predefined set of positive and negative training examples (x_i, y_i). In our case the positive examples are face images and the negative examples are non-face images.

1. Input: Given training examples (x_i, y_i), i =1..N where $y_i = 1$ for positive and $y_i = 0$ for negative examples
2. Initialization: initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for y_i = 0,1 respectively, where m and l are the number of positive and negative examples.
3. For t = 1, …, T:
   a) Normalize all weights
   b) For each feature j, train classifier h_j with error $\varepsilon_t = \sum_i w_i \, | h_j(x_i) - y_i |$.
   c) Choose h_t with lowest error ε_t
   d) Update the weights: $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$

$$e_i = \begin{cases} 0 & : \ x_i \text{ correctly classified} \\ 1 & : \ \text{otherwise} \end{cases}$$

and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

4. Final strong classifier:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

with

$$\alpha_t = \log \frac{1}{\beta_t}$$

Figure 21: The AdaBoost algorithm.

The AdaBoost algorithm iterates over a number of T rounds. In each iteration the space of all possible features is searched exhaustively to train weak classifiers that consist of one single feature. In [Viol01], to train a single weak classifier a threshold has to be found for the feature value to discriminate between positive and negative examples. In our approach, the latter is slightly different. Instead of using a threshold, the chosen weak classifier is the RVM for discriminating between the positive and negative examples. This means that for each feature, the

weak RVM classifier determines the optimal classification function such that a minimum number of examples is misclassified. The next step in the algorithm is to choose the best weak classifier, which means choosing a classifier with the lowest classification error ε_t. The error is evaluated with respect to w_t, which are the normalized weights of round *t*.

After choosing the best weak classifier concerning the weighted classification error on the training set, all training examples are re-weighted to focus in the next round on those examples that were not correctly classified. The weights are updated according to the following:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \qquad \text{(Eq. 4.23)}$$

In the equation (Eq. 4.23) β is equal to $\frac{\varepsilon_t}{1-\varepsilon_t}$, where ε_t is the lowest error of round *t*. The parameter e_i is defined to be zero if the example x_i is classified correctly by the classifier, which is the RVM in our case. Otherwise, parameter e_i is equal to one. Depending on how many features we want to select, *T* can be set to that number which will force the algorithm to iterate *T* times. In each iteration one feature will be selected with the lowest error.

It is important to note that for each iteration we need to train more than 162336 features and select the best feature out of the whole set. In the next rounds this procedure is continued again and again for all *T* rounds for all 162336 features. In our case with 3000 samples the training time for one feature differs from one to 1.5 minutes. Note that the implementation is done in Matlab 6.5 on a Windows machine with AMD Athlon™ XP 2200+ 1.80 GHz processor with 512 MB RAM. The total training time would be more than 20 weeks. This is only the case if we assume that the features only need to be trained once. This means in the next round *t* of the boosting algorithm we do not need to train the features anymore, but merely need to re-weight the training samples. But still this will take weeks before the first feature can be produced. Considering the time we have for finishing the thesis project, we either have to speed up the boosting process or find another solution for extracting and selecting features for face detection. Our solution lies in the application of genetic algorithms. The latter are in general suitable for searching on very large data sets in reasonable time. The exhaustive search of AdaBoost will be replaced by an evolutionary search algorithm. Instead of looking in the complete feature space, the evolutionary algorithm finds the optimal solutions in a subspace of all features. Further details will be given in section 4.5.

In the final step of Figure 21 a strong classifier can be constructed from the selected features. This is done using the following equation:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Eq. 4.24)}$$

Thus, the final classifier is a weighted linear combination of the *T* features, where the weights are inversely proportional to the training errors. The initial AdaBoost threshold ($1/2 \sum_{t=1}^{T} \alpha_t$), is designed to yield a low error rate on the training data. A lower threshold yields higher detection rates and higher false positive rates. This is extremely important for constructing a cascade of classifiers.

4.5 Genetic Algorithm for faster boosted feature selection

Genetic Algorithms (GA) is a term used to describe problem solving systems in which evolution is the key element. The emphasis in GA lies on natural selection and survival of the fittest. The evolution of individuals is simulated by probabilistic genetic processes of selection, mutation and reproduction. They are the driving forces that lead to 'well-adapted' individuals. GA can be used for different purposes on different areas. For example, it can be used as simulation tools for the evolution of biological populations (Roosenberg, 1967). It can also be used as a stochastic search technique to combinatorial optimization problems. In our case, it will be applied to speed up the AdaBoost algorithm.

The exhaustive search of AdaBoost is in fact a brute force search on the whole space of rectangular Haar-like features. As described in section 4.1 there are in total 162336 features to be trained and it will be in the order of weeks to train them all. To speed up the terrible long training time, it will be beneficial to use GA in combination with AdaBoost. Speeding up the boosting algorithm is performed by replacing the exhaustive search of AdaBoost by an genetic search algorithm called Evolutionary Search (ES). ES is an instance of GA and as the name already suggests its focus is on the field of searching. Figure 22 shows the steps of the ES algorithm.

There are two probabilistic genetic operators that drive the ES process: crossover and mutation. The crossover operator simulates the process of reproduction in the evolution theory. With reproduction the sexual or asexual process is meant by which organisms generate new individuals of the same kind. Sexual reproduction results in an offspring that contains a combination of information from each parent. Asexual reproduction typically results in an offspring that is genetically identical to the parent. The mutation operator brings in diversity in the individuals. This is essential because evolution, by definition, requires diversity in order to work.

```
Step 3b)  Evolutionary Search ()
Begin
        t := 0;
        initialize_feature_population (P(0));
        repeat
                P' := select (P(t));
                Crossover (P');
                Mutate (P');
                Train_classifiers (P');
                Evaluate_classification_error (P');
                P (t+1) := replace (P(t), P');
                t := t+1;
        until terminated;
end;
```

Figure 22: Evolutionary Search

With EABoost, (ES combined with AdaBoost) the space of all rectangular Haar-like features is searched for good features. Crossover differs slightly from what is described above in the implementation of EABoost. It is implemented as follows: given two parents A and B the resulting offspring C is calculated as:

$$C_i = \begin{cases} B_i : r < 0.5 \\ A_i : otherwise \end{cases} \quad , i = 1...n \qquad \text{(Eq. 4.25)}$$

where r is a uniform random number $\in [0,1]$, n describes the length of the individuals and the parents are selected randomly. The reason that crossover is implemented this way is the fact that we have complete knowledge of the whole feature set. We know exactly the number of possible features: no new features will be generated in the feature space and we also know the algorithm should not select any features outside this set. In our case crossover means randomly selecting another feature from the feature space. Mutation of an individual is done by the following scheme: 1) Choose a new type $t \in [1,5]$ with probability p_{mt}. 2) Mutate positions of feature corners by adding a random constant (x_{rm}, y_{rm}), with $x_{rm}, y_{rm} \in [-7,7]$. 3) Use a repair operator on individuals that are no longer feasible after the mutation. After mutation, corners of the rectangle features could be in wrong order. Also could the feature's lengths be negative or bigger than the maximum allowed length. Mutation in EABoost is in fact also a process to randomly select another feature from the feature set.

The third important operator in EABoost is the fitness operator that implements the natural selection process of evolution. It measures how good an individual is at competing in its

environment. In our case, with a population of 250, each feature f_i with $i=1,\ldots,250$, that is generated will be evaluated against the test set which results in an error ε_i. The strongest feature that will be selected will then be the one that satisfy the fitness function $min(\varepsilon_i)$. It is the same criterion AdaBoost was using to select a weak feature. In Figure 22 it is displayed as the evaluation of the classification error. We get EABoost when we replace step 3b, the exhaustive search of AdaBoost, with ES.

In the implementation of EABoost a generation consists of 150 Haar-like features. This size is the result of a trial and error process in which a trade-off is made between speed and error rate. The size of the feature set would affect the training time and the quality of the found solution. If the population is too small, the training time will be shorter, but there is a probability that no good features will be generated and as a result a bad feature will be selected. On the other hand if the population is too large, it will take longer to train and the purpose of using ES in the first place was to reduce the training time. A generation should be sufficiently large to create sufficient diversity covering the possible solution space. Other parameters of EABoost are the probabilities for crossover and mutation. In literature crossover usually happens with a probability of 75-95% and mutation 0.5-5%. In our situation, crossover and mutation are two processes for randomly selecting another feature from the total set. The difference is that crossover really selects a feature at random. Mutation also selects at random, but the feature's location will be near to the original feature which it is mutated from. Since we prefer the latter, in EABoost crossover takes place in 20% of the time and mutation takes place in 80% of the time. ES ends when it converges.

Note that ES as a simulation of a genetic process is a non-deterministic search. It is not sure whether a found solution is optimal or suboptimal. This could be seen as one of the disadvantages of ES. But on the other hand, ES can quickly scan a huge solution set. On top, bad solutions in the population set do not affect the end solution negatively as they are simply discarded. ES is also very useful for complex or loosely defined problems. Once the problem is translated successful into an ES problem it does not have to know any rules of the original problem. The evolution mechanism will do the work.

4.6 The differential cascade

This section describes an algorithm for constructing a cascade of classifiers which drastically reduces the computation time. The number of sub-windows to be classified by the detector is enormous and requires a lot of computation time. The main idea is that smaller and therefore more efficient, boosted classifiers can be built which reject many of the negative sub-windows while detecting almost all positive instances. Simpler classifiers are used to reject the majority of sub-windows before more complex classifiers are called upon to achieve low false positive rates.

The phases in the cascade are constructed by training classifiers using EABoost. In general the cascade has the form of a degenerate decision tree (See Figure 23). Input for the cascade is the collection of all sub-windows also called scanning windows. They are first passed through the first layer in which all sub-windows will be classified as faces or non faces. The negative results will be discarded. The remained positive sub-windows will trigger the evaluation of the next classifier. The same process is performed in every layer. The sub-windows that reach and pass the last layer are true faces.

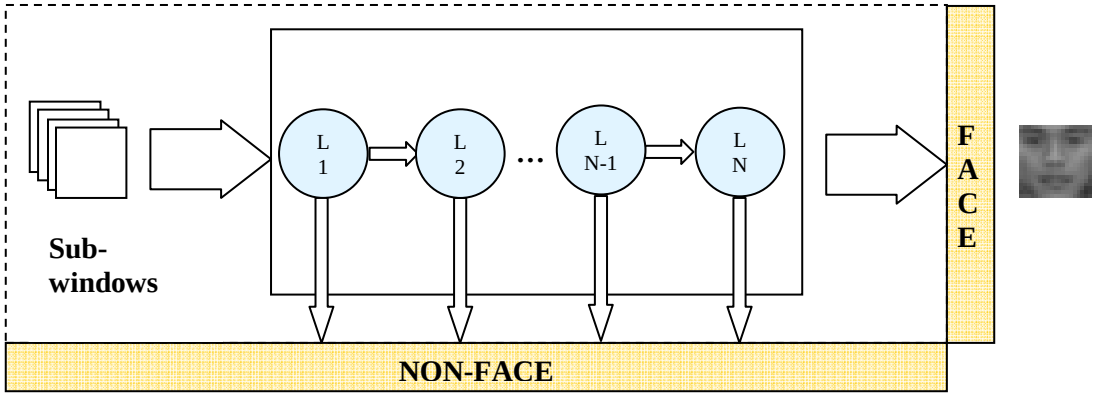


Figure 23: A cascaded classifier with N layers.

The structure of the cascade reflects the fact that within any single image on overwhelming majority of sub-windows are negative. As such, the cascade attempts to reject as many negatives as possible at the earliest stage possible. Every layer consists of only a small number of features. In the early stages, with only a couple of the best features it is possible to determine the existence of a non-face (negative sub-window). Determining the presence of a face usually needs more features. Therefore, the cascade has an increasing number of features in each consecutive layer. While a positive instance will trigger the evaluation of every classifier in the cascade, this is an exceedingly rare event.

During implementation the number of layers and the number of features per layer was driven through a trial and error process. In this process the number of features was increased until a significant reduction in the false positive rate could be achieved. More layers were added until the false positive rate on the validation set was nearly zero while still maintaining a high correct detection rate. Following is the specification of the cascaded classifier that we get after training.

Table 5: cascaded classifier specification.

Cascade layer nr.	Number of features	True positive rate	False positive rate
1	2	0.95	0.422
2	4	0.95	0.69
3	15	0.932	0.244
4	17	0.93	0.24
5	19	0.93	0.188

To conclude, this chapter described different algorithms that when combined together make face detection possible. To be more precise, the algorithms in this chapter are implemented as the WUXTRAP application for face detection. As stated in chapter 3, WUXTRAP's design goal was to train our classifier and produce the proper WUX values that are needed for our FLEX module that reads images, find faces and extract FCPs. WUXTRAP is implemented using Matlab 6.5. Thus, all training of the RVMs is done using Matlab and FLEX, the running detection system, is implemented in Java. Remind that Java was chosen as stated in our implementation objectives of chapter 2. Since the training is independent of the functionalities in FED, it was not necessary to do the training in Java. Working with images, training the RVM, testing and evaluating the selected features are all computationally intensive tasks. We can state that our choice for Matlab is mainly based on its powerful computation engine and its high-level interactive environment in which a lot of very useful mathematical functions are already implemented and made available.

5

Face Detection – Experimental Results

The experimental results related to face detection will be presented in this chapter. Section 5.1 discusses the result of RVM trained on intensity values. Section 5.2 gives the results of RVM trained on binary values. Section 5.3 presents the parameter tuning of RVM for Haar-like features. The optimal kernel will be discussed and the results of the training using Haar-like features, combined in EABoost, are given in section 5.4.

5.1 Relevance Vector Machine on intensity values

The results from applying RVM on intensity values are already presented in Table 3. The results show the performance of the RVM trained and tested with the same data set. The overall detection rate is 67.96%. To see how RVM performs on images it has not been trained with, we have tested the same RVM on a subset of the CMU face database (see Table 6). The results are shown in Table 7.

Table 6: Test set specification.

Dataset parameters	Values and description
Database	CMU
Sample size	19x19 pixels
Number of classes	2
Class 0	Non-faces
Class 1	Faces
Number of samples (0/1)	5036/472

Table 7: RVM test results on intensity values.

Kernel	**CMU dataset consisting of faces only**		**CMU dataset consisting of non-faces only**	
	Number of test samples	**Detection rate %**	**Number of test samples**	**Detection rate %**
Gauss: 0.5, 1.0, 2.0, 3.0, 4.0, 5.0	472	0	5036	100
Laplace: 0.5, 1.0, 2.0, 3.0, 4.0, 5.0		0		100

The results confirm what we concluded earlier. RVM can not be trained on pixel intensities. These are insufficient to allow unambiguous recognition of a pixel. In this case, the RVM trained on intensity values does not generalize at all to recognize any sample that is not present in the training set. This simply means that the RVM is not trainable in this way. For all kernel tested, the detection rate was zero, i.e. no single face in the test set is recognized as a face.

Before we come to this conclusion, RVM is also trained on binary information. This means that input images are first converted into binary images by a threshold function. This function converts all intensity values higher than the mean intensity value to 1 and 0 otherwise.

5.2 Relevance Vector Machine on binary values

The training results of RVM trained on binary information is shown in Table 8 and Table 9. Training is done on the same subset from MIT CBCL (see Table 2). The result of testing the trained RVM on this same set is shown in Table 8. For the chosen type of data representation it is clear that the detection rate is very good except for the Laplace kernel with a scale smaller than 2.0 and the Gaussian kernel with a scale smaller than 3.0. These lower width kernels are not able to fit the data properly. As a result, they can not achieve a detection rate higher than 68%. Increasing the width gives a stable detection rate of around 94-95%.

In Table 9, testing is done on the CMU subset with 472 faces and 5036 non faces (see Table 6). The results are very disappointing. The results show a detection rate of no more than 52 %. In this case overfitting occurred. The accuracy is very low and most samples are recognized as non-face. To conclude, the chosen approach to work with binary information of the image is apparently not working.

Table 8: RVM test results: training and testing are both performed on the same MIT CBCL subset.

Kernel	Nr of test samples	Detection rate %	Nr of false negatives	Nr of false positives
Gauss 0.5 – 3.0	2500	67.76	806-815	0
Gauss 5.0		93.92	103	49
Gauss 7.0		95.08	58	49
Laplace 0.5 – 1.0		67.80	805-815	0
Laplace 2.0		83.88	339	64
Laplace 3.0		92.76	117	64
Laplace 4.0		94.72	67	65
Laplace 5.0		95.04	60	64
Laplace 6.0		96.00	57	43
Laplace 7.0		94.96	72	54

Table 9: RVM test results: training is done on a MIT CBCL subset, testing is performed on a CMU subset.

Kernel	CMU database consisting of faces only		CMU database consisting of non-faces only	
	Number of test samples	Detection rate %	Number of test samples	Detection rate %
Laplace 0.5 – 1.0	472	0	5036	100
Laplace 2.0		22.03		100
Laplace 3.0		47.25		97.22
Laplace 4.0		50.85		96.64
Laplace 5.0		51.91		97.34
Laplace 6.0		43.86		97.93
Gauss 0.5 – 3.0		0		100
Gauss 4.0		9.53		93.59
Gauss 5.0		38.77		96.68
Gauss 6.0		50.42		96.82
Gauss 7.0		30.30		97.86

5.3 Parameter tuning for Relevance Vector Machine using 2-fold cross validation

In EABoost and in the cascade building process, RVM needs to classify Haar-like features. In order to choose the best parameters for RVM, 2-fold cross validation is used. During this tuning process we used the MIT CBCL subset (see Table 2) to let EABoost select three features. We then compare the performance of RVM of these features using different kernels. Table 10 shows the EABoost parameters that are used during this training process. The choice for the population size, crossover rate and mutation rate was already discussed in section 4.5. We have chosen to use a Laplace 2.0 kernel as it is as much as a random choice. At this point we do not know which kernel will perform better. It is only important to choose the same kernel for all features.

Table 10: EABoost specification: the selection of three features that will be used for 2-fold cross validation.

EABoost parameters	Values and description
Population size	250
Crossover rate	0.20
Mutation rate	0.80
Classifier/ kernel	RVM/ Laplace 2.0
Number of features to choose	3

Datasets are created of feature number 38978, 28893 and 45297, which are the features selected by EABoost. The datasets are filled with the feature values of the mentioned features. Feature 38978 has the best performance in the first pool of 250 features. Feature 28893 and 45297 are the best features in the second respectively third pool of 250 features. In Figure 24 all three features are plotted in a 19x19 window. The parameters of feature 38978 are (7, 8, 13, 12, 3). Those of feature 28893 and 45297 are (11, 5, 19, 11, 4) and (10, 10, 18, 16, 1).

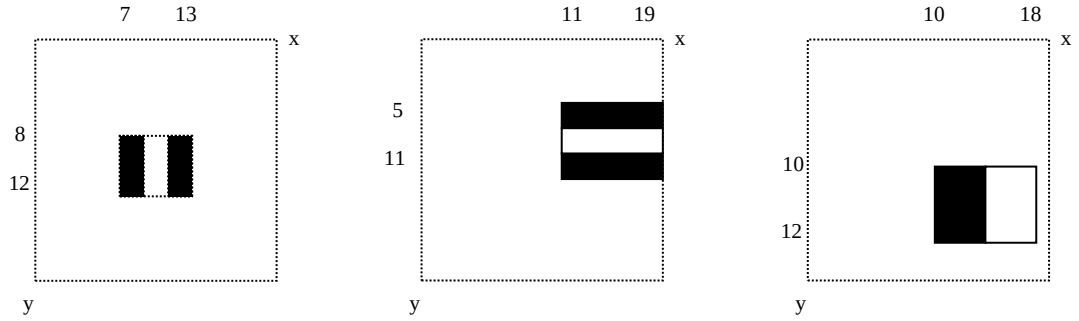


Figure 24: feature 38978 (left), feature 28893 (centre) and feature 45297 (right).

The results after using 2-cross validation are shown in Table 11. Tables with the complete cross validation results can be found in appendix A. A good classifier should have a low error rate with a small standard deviation. From these tables it is not easy to choose and conclude the best classifier. Therefore, Receiver Operating Characteristics (ROC) graphs are generated from these results, see Figure 25. ROC graph is a technique for visualizing, organizing and selecting classifiers based on their performance. In a ROC graph, the true positive rate is plotted on the y-axis and the false negative rate is plotted on the x-axis. The true positive rate is defined as the number of true positive samples divided by the total number of positive samples in the dataset. In the same way, the false negative rate is defined as the number of false negatives divided by the total number of positive samples in the dataset.

Table 11: RVM 2-fold cross validation results trained on Haar-like features.

Feature	38978		28893		45297	
Kernel	Error rate %	SD	Error rate %	SD	Error rate %	SD
Gauss 0.5	27.75	2.33	41.95	10.68	46.50	4.10
Gauss 1.0	29.10	2.83	39.60	7.64	44.80	6.93
Gauss 2.0	26.30	0.85	35.45	7.71	38.60	1.84
Gauss 4.0	26.60	6.08	31.70	7.35	35.50	3.68
Gauss 5.0	25.35	5.30	32.40	2.83	36.55	3.61
Laplace 0.5	35.20	14.42	26.70	0.99	42.70	9.62
Laplace 1.0	25.35	1.77	35.55	9.26	35.60	1.41
Laplace 2.0	29.25	9.83	32.00	7.50	41.60	7.78
Laplace 3.0	28.45	4.31	34.85	8.98	38.60	5.94
Laplace 4.0	24.85	1.77	26.10	2.97	42.55	1.77
Laplace 5.0	26.20	2.12	25.90	1.84	37.45	7.57

Informally, one point in ROC space is better than another if it is to the northwest of the first. It means that the true positive rate is higher and on the same time the false positive rate is lower, or even. In the first ROC graph the four classifiers (Gauss 4.0, Gauss 5.0, Laplace 1.0 and Laplace 4.0) are on a straight line and they are all northwest to the other classifiers. There is no single one that excels. The best classifier between these four should be the one that meets the requirements most. If it is desired that the true positive rate is high, no matter the false positive rate, then the classifier positioned most north in the ROC space should be chosen. If on the other hand it is preferred to keep the false positive rate low, then the classifier positioned most south in the ROC space should be chosen.

In the second graph it can be noticed that kernels Laplace 0.5, Laplace 4.0 and Laplace 5.0 performs slightly better than the others. In the last graph, all kernels are in fact performing not well. Keeping the true positive rate above 50% and the false positive rate low, Gauss 3.0, Gauss 4.0 and Laplace 1.0 would be the choice. From the three ROC graphs, three candidates can be chosen: Gauss 4.0, Laplace 1.0 and Laplace 4.0. To decide which of the candidates is the best we examine their performance in more details by adjusting their thresholds that is described in section 4.4. Lowering the threshold will give a better detection rate, but on the other hand it also increases the false positive rate. The threshold range is from 0.35 to 0.65. The result is plotted in a new ROC curve in Figure 26. It can be concluded that Laplace 4.0 has the best performance and therefore, Laplace 4.0 is used in EABoost and the cascade building process.

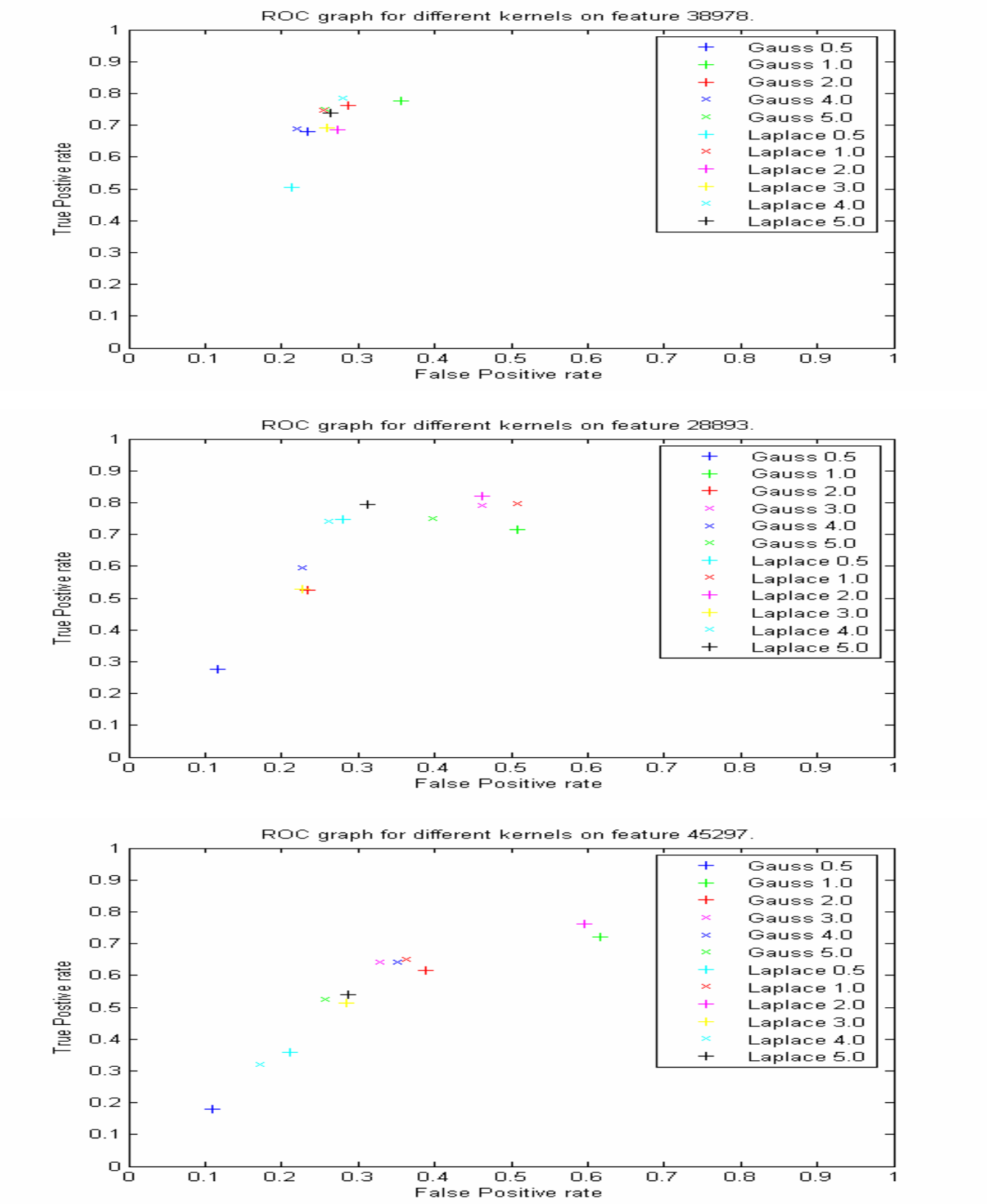


Figure 25: ROC graphs of three Haar-like features with different kernels.

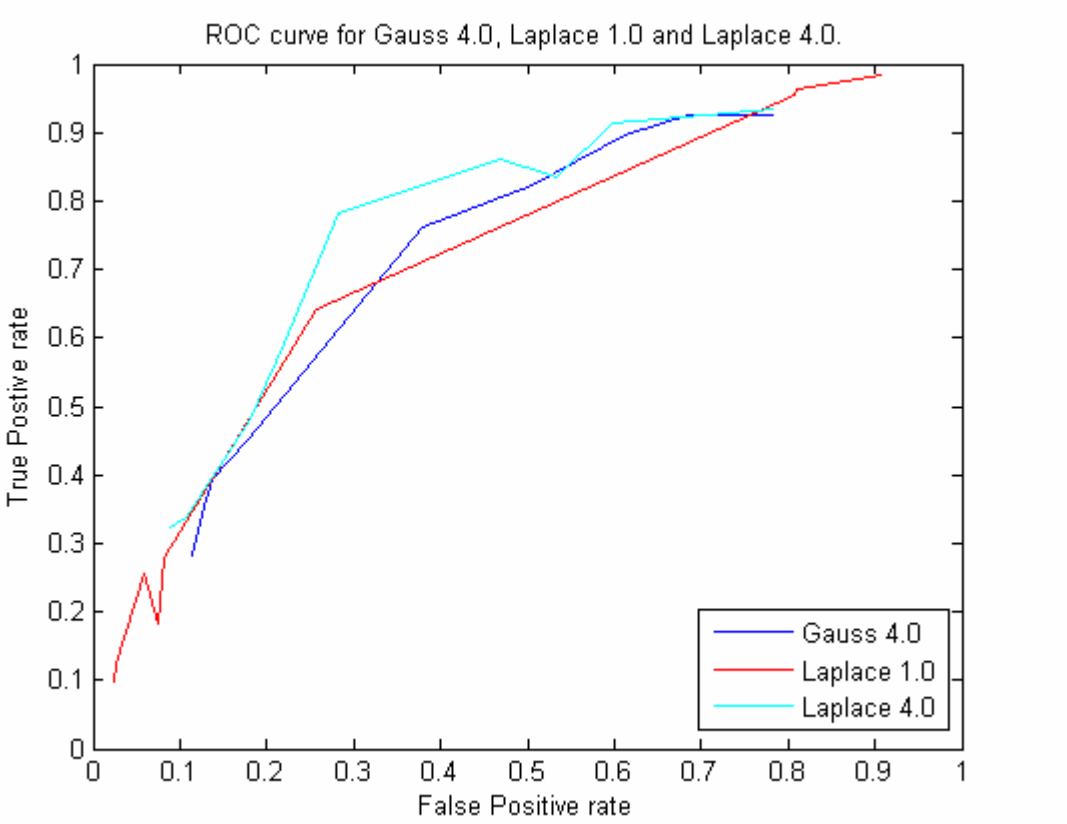


Figure 26: ROC curves of three kernels, obtained by adjusting each classifier's threshold.

5.4 Evolutionary-AdaBoost training results

Now that the RVM kernel is determined, the training procedure can start. To summarize, the training set is described in Table 4, the test set in Table 12 and the cascade test set in Table 13. Note that these subsets from the Viola Jones database are all distinct. Table 14 shows the parameters of the EABoost training.

Table 12: EABoost feature test set.

Dataset parameters	Values and description
Database	Viola Jones
Sample size	24x24 pixels
Number of classes	2
Class 0	Non-faces
Class 1	Faces
Number of samples (0/1)	500/500

Table 13: Cascade test set.

Dataset parameters	Values and description
Database	Viola Jones
Sample size	24x24 pixels
Number of classes	2
Class 0	Non-faces
Class 1	Faces
Number of samples (0/1)	1500/1500

Table 14: EABoost parameters.

EABoost parameters	Values and description
Population size	150
Crossover rate	0.20
Mutation rate	0.80
Classifier/ kernel	RVM/ Laplace 4.0
Feature false positive rate	0.3
Feature true positive rate	0.75
Target false positive rate	< 0.05
Target true positive rate	> 0.95
Number of features to choose	Depends on target false positive rate

The feature false positive rate and feature true positive rate are indications for EABoost whether or not a Haar feature should be selected for the cascade. In the training procedure, a selected feature will be trained on the training set and tested on the test set. So, each trained feature will produce a false positive rate and true positive rate. If the false positive rate is lower than 0.3 and true positive rate higher than 0.75, the feature will be selected for the cascade. Every time a qualified feature is added to a layer, the whole set of features in the layers will be evaluated on the cascade test set to see if the layer can achieve the target false positive rate and target true positive rate. If this is the case, the training procedure will be stopped and training for the next layer can begin. If it does not meet to the requirements of the maximal false positive rate and minimal true positive rate, EABoost searches for one extra feature and the above procedure is repeated. A qualified feature will only be added to the cascade if and only if it improves the cascades detection rate. The final result of the training procedure is already shown in Table 5 of the previous section.

Part III

Facial Characteristic Point Detection

6

Facial Characteristic Points Detection – Extraction

In the previous chapter a robust face detector has been discussed. This detector provides the necessary faces in the given image for further processing. We assume that the faces detected are full-frontal faces. Conform to the face model facial characteristic points needs to be extracted. A closer examination of the face model reveals that some of the facial characteristic points are actually corner points of the facial features i.e. eye, mouth, nose, eye brow. So, the idea is to use a corner detection algorithm to detect these corners. We conduct this by first comparing the most important corner detectors. By means of the performance and speed, the best corner detector is chosen.

Table 15 shows two corner detectors which satisfies the criterion of good performance, accuracy and speed. In the final implementation both corner detection algorithms are implemented which allow us to combine the corner results of both detectors. The results do not only include the corner points we are interested in but also lots of unimportant corners. The next logical step is thus finding the real FCP corners among the FCP corner candidates. To do this we will continue to use our classifier, the RVM, to decide whether the corner is what we need or else. As we already know, before we can use the RVM to classify corners it must be trained first. More of this will be discussed in detail in the next chapter after the discussion of the two corner detectors. In section 6.1, the different corner detectors will be compared. In section 6.2 and 6.3 the two corner detectors will be considered in detail.

6.1 Corner detection algorithms comparison

Algorithms for corner detection may be divided into direct corner detectors and colour distribution based corner detectors. The former is based on algorithms that work directly with the

brightness values of images. They often model an image as a surface considering the directional gradients or derivatives at each pixel point, if the value of its corner response function exceeds a defined threshold this pixel will be considered as a corner point. The detectors described by [Harr88], [Sojk03] both belong to the so called direct corner detectors. Each algorithm defines its own corner response function and so the performances are usually decided by the corner response function.

The second type detectors do not model an image as a surface. Instead, they consider the statistical colour distribution in a circular neighbourhood centring at each pixel rather than compute the directional gradients or derivatives. SUSAN corner detector [Smi97], Compass corner detector [Ruz01] and a proposed detector [Son03] belong to this type. The SUSAN detector classifies each pixel into edge, corner and flat area. Compass detectors utilize a group of colours, instead of a single colour, to represent the statistic colour distribution in a circular neighbourhood. It can handle both uniform-coloured region and textured regions. The detector proposed in [Son03] emphasizes both spatial and statistical colour distributions; it is much faster than compass detectors and more accurate than SUSAN detectors.

Table 15: Comparison of different corner detectors

Publication year	Index	Feature Detected	Accuracy	Speed	Real-time Processing	Grade
1978	[Bea78]	Corner	Low	Very fast	Yes	3
1982	[Kit82]	Corner	Low	Fast	Yes	3
1988	[Harr88]	Corner	High	Middle	Yes	4
1993	[Der93]	Corner	Very low	Very low	No	2
1997	[Smi97]	Edge & Corner	Low	Middle	Yes	3
2001	[Ruz01]	Corner, Edge, & Junction	Very high	Very slow	No	3
2003	[Son03]	Corner, Edge & Junction	Very high	Middle	Yes	4.5
2003	[Sojk03]	Corner	Very high	Middle	Yes	4.5

Table 15 [Sojk03b] shows the comparison of different corner detectors based on namely the accuracy and detection speed. Colour distribution based corner detection algorithms are known to have very accurate detection rate, but a drawback is the relative slower detection speed due to the detection of corners, edges and junctions in one round. The processing speed of early direct corner detectors is very fast, but they often do not have very high detection rate. We have also

included another table (Table 16) which shows the comparison of direct corner detectors tested on an image with 291 reference corners in total.

In the previous chapters it is stated that the face detection system detects faces from grey-level images. That is because the face detector is trained on a training set consisting of grey-level images. This gives the reason to adopt a direct corner detection algorithm instead of using a colour distribution based corner detection algorithm. From all direct corner detectors the Harris detector [Harr88] and the Sojka detector [Sojk03] are the better ones among them. In this part of the project it is only the intention to select a corner detector that is good enough; that means a detector that is able to detect all corners of the facial features (e.g. left-eye corners, right-eye corners etc.).

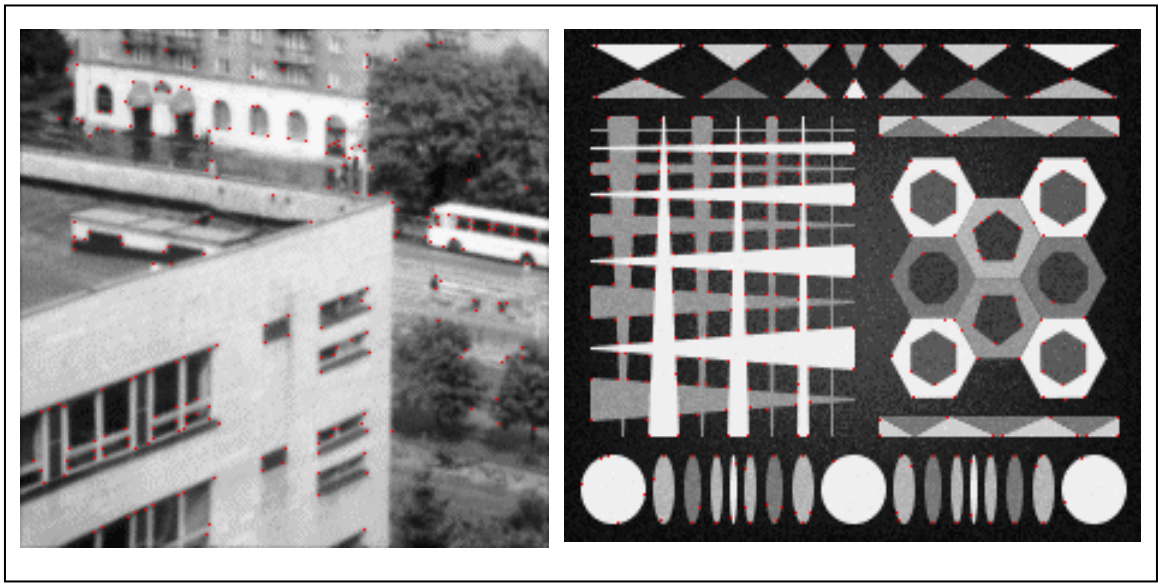


Figure 27: (Left) Test image with 291 reference corners.
(Right) Test image with 470 reference corners. The red dots are the corner points.

Table 16: Comparison of direct corner detectors on a test image of 291 reference corners.

Detector Name	Total Corners	Correct Det.	False Det.	Multiple Det.	Total Error	Localization Error	Grade (out of 5)
Beaudet	291*	155	21	10	167	1.85	2.0
Deriche		142	25	10	184	2.05	1.5
Harris		**187**	**10**	**6**	**120**	**0.98**	**4.0**
Kitchen		163	26	15	169	1.87	2.0
SUSAN		152	29	1	169	1.63	2.5
Sojka		**229**	**9**	**8**	**79**	**0.81**	**4.5**

* The total number of corners is the number of reference corners in the real image.

Table 17: Comparison of direct corner detectors on a test image of 470 reference corners.

Detector Name	Total Corners	Correct Det.	False Det.	Multiple Det.	Total Error	Localization Error	Grade (out of 5)
Beaudet	470*	363	54	24	185	1.62	2.0
Deriche		308	22	5	189	1.49	1.5
Harris		**431**	**14**	**9**	**62**	**0.73**	**4.0**
Kitchen		356	48	31	193	1.75	2.0
SUSAN		338	34	7	173	0.87	2.5
Sojka		**466**	**1**	**1**	**6**	**0.57**	**4.5**

* The total number of corners is the number of reference corners in the real image.

6.2 Harris and Stephens corner detection

6.2.1 Theoretical background

The corner detector that is known as the Harris corner detector should actually be called the Harris-Stephens corner detector. Since it is adopted as the Harris corner detection in general by most researchers, we will do that also. The main concern in [Harr88] is to use a computer vision system based upon motion analysis of a monocular image sequence. By extraction and tracking of image features 3D analogues can be constructed of the features. Conclusions are drawn that explicit 3D representation of curving edge may be unobtainable, the connectivity it provides may be sufficient for many purposes. Tracked edge connectivity, supplemented by 3D locations of corners and junctions can provide both a wire-frame structural representation and delimited image regions which can act as putative 3D surfaces. Consistency of image edge filtering is considered of prime importance for 3D interpretation of image sequences using feature tracking algorithms. The state-of-the-art edge filters are not designed to cope with junctions and corners, and are reluctant to provide any edge connectivity. The use of edges to describe some objects like the bush is very doubtful since a small change in edge strength or in the pixilation causes a large change in the edge topology. The solution to their problem is to detect both edges and corners.

6.2.2 Corner model

[Harr88] proposes a detector which is actually a combined corner and edge detector. Harris corner detector draws its origin in the corner detector proposed by Moravec. Moravec's corner detector works by considering a local window in the image, and determining the average changes of image intensity that result from shifting the window by a small amount in various directions. Three cases are actually considered by this detection algorithm:

- If the windowed image patch is flat, which means it is approximately constant in intensity, then all shifts will result in only a small change
- If the window straddles an edge, then a shift along the edge will result in a small change, but a shift perpendicular to the edge will result in large change
- If the windowed patch is a corner or isolated point, then all shifts will result in a large change. A corner can thus be detected by finding when the minimum change produced by any of the shifts is large.

Denoting the image intensities by *I*, the change *E* produced by a shift (x, y) is given by:

$E_{x,y} = \sum_{u,v} w_{u,v} \left| I_{x+u,y+v} - I_{u,v} \right|^2$, where *w* specifies the image window: it is unity within a specified window and zero elsewhere. The shifts, (x, y), which are considered comprise [(1, 0), (1, 1), (0, 1), (-1, 1)]. The Moravec's corner detector is to simply look for local maxima in min {E} above some threshold value.

The Moravec's corner detection algorithm suffers from a number of problems. Corrective measures are taken by [Harr88] and this lead to a new corner detection method: Harris corner detector.

Ø The response in Moravec's algorithm is anisotropic because only a discrete set of shifts at every 45 degrees is considered. In Harris corner detector all possible small shifts can be covered. This is done by performing an analytic expansion about the shifts origin:

$$E_{x,y} = \sum_{u,v} w_{u,v} \left| I_{x+u,y+v} - I_{u,v} \right|^2$$

$$= \sum_{u,v} w_{u,v} \left| xX - yY + O(x^2, y^2) \right|^2 \qquad \text{(Eq. 6.1)}$$

where the first gradients are approximated by:

$$\mathrm{X} = I \otimes (-1, 0, 1) = \partial I / \partial x \qquad \text{(Eq. 6.2)}$$

$$\mathrm{Y} = I \otimes (-1, 0, 1)^T = \partial I / \partial y \qquad \text{(Eq. 6.3)}$$

Hence, for small shifts, E can be written as:

$$E(x,y) = Ax^2 + 2Cxy + By^2 \quad \text{(Eq. 6.4)}$$

where

$$\begin{aligned} A &= X^2 \otimes w \\ B &= Y^2 \otimes w \\ C &= (XY) \otimes w \end{aligned} \quad \text{(Eq. 6.5)}$$

Ø The response in Moravec's algorithm is also noisy because the window is binary and rectangular. To cope with this problem Harris uses a smooth circular window, for example a Gaussian:

$$w_{u,v} = \exp -(u^2 + v^2)/2\sigma^2 \quad \text{(Eq. 6.6)}$$

Ø The operator in Moravec's detector responds too readily to edges because only the minimum of E is taken into account. [Harr88] reformulates the corner measure to make use of the variation of E with direction of shift.

The change, E, for the small shift (x, y) can be concisely written as:

$$E(x,y) = (x,y)M(x,y)^T \quad \text{(Eq. 6.7)}$$

Where the 2x2 symmetric matrix M is

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \quad \text{(Eq. 6.8)}$$

E is closely related to the local autocorrelation function, with M describing its shape at the origin (explicitly, the quadratic terms in the Taylor expansion). Let α, β be the eigenvalues of M. α and β will be proportional to the principal curvatures of the local autocorrelation function, and form a rotationally invariant description of M. As before, there are three cases to be considered:

- If both curvatures are small, so the local autocorrelation function is flat, then the windowed image region is of approximately constant intensity. This means arbitrary shifts of the image patch cause little change in E;
- If one curvature is high and the other low, so that the local autocorrelation is ridge shaped, then only shifts along the ridge/the edge cause little change in E: this indicates an edge;
- If both curvatures are high, so that the local autocorrelation function is sharply peaked, then shifts in any direction will increase E: this indicates a corner

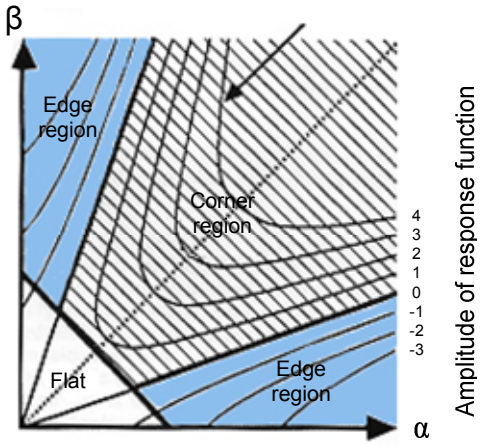


Figure 28: Auto-correlation principal curvature: bold lines give corner/flat classification, fine lines are equi-response contours.

6.2.3 Corner decision

Consider the graph of (α, β) space. An ideal edge will have α large and β zero (this will be a surface of translation), but in reality β will merely be small in comparison to α, due to noise, pixilation and intensity quantization. A corner will be indicated by both α and β being large, and a flat image region by both α and β being small. Since an increase of image contrast by a factor of p will increase and β proportionally by p^2, then if (α, β) is deemed to belong in an edge region, then so should $(\alpha p^2, \beta p^2)$, for positive values of p. Similar considerations apply to corners. Thus (α, β) space needs to be divided as shown by the heavy lines in Figure 28.

In summary, [Harr88] consider the minimum and maximum eigenvalues, α and β, of the image gradient covariance matrix in developing their corner detector. The gradient covariance matrix is given by:

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad \text{(Eq. 6.9)}$$

where I_x and I_y denote the image gradients in the x and y directions. This matrix characterizes the structure of the image gray level patterns. In fact, the geometric interpretation of the gray levels is encoded in the eigenvectors and eigenvalues of the matrix. A `corner' is said to occur when the two eigenvalues are large and similar in magnitude.

Corner/edge response function

Not only are corner and edge classification regions needed but also a measure of corner and edge quality or response. The size of the response will be used to select isolated corner pixels and to thin the edge pixels.

First consider the measure of corner response, R, which is required to be a function of α and β alone, on grounds of rotational invariance. Tr(M) and Det(M) will be used in the formulation to avoid the explicit eigenvalue decomposition of M, thus:

$$Tr(M) = \alpha + \beta = A + B \quad \text{(Eq. 6.10)}$$

$$Det(M) = \alpha\beta = AB - C^2 \quad \text{(Eq. 6.11)}$$

A corner region pixel (i.e. one with positive response) is selected as a nominated corner pixel if its response is an 8-way local maximum. Similarly, edge region pixels are deemed to be edges if their responses are both negative and local minima in either the x or y directions, according to whether the magnitude of the first gradient in the x or y direction respectively is the larger. This results in thin edges. By applying low and high thresholds, edge hysteresis can be carried out, and this can enhance the continuity of edges. These classifications results in a 5-level image comprising: background, two corner classes and two edge classes. Further processing (similar to junction completion) will delete edge spurs and short isolated edges, and bridge short breaks in edges. This results in continuous thin edges that generally terminate in the corner regions. The

edge terminators are then linked to the corner pixels residing within the corner regions, to form a connected edge-vertex graph.

To avoid an explicit eigenvalue-decomposition Harris and Stephens devise a measure using the determinant and trace of the gradient covariance matrix:

$$R = \mathrm{Det}(M) - k(\mathrm{Tr}(M))^2 \qquad \text{(Eq. 6.12)}$$

where $\mathrm{Det}(M)$ and $\mathrm{Tr}(M)$ are defined above, the parameter k is traditionally set to 0.04. This produces a measure that is large when both α and β are large. However there is a problem of determining what is large. Noting that elements of the image gradient covariance matrix have units of intensity gradient squared we can see that the determinant, and hence the measure R will have units of intensity gradient to the fourth. This explains why the Harris operator is highly sensitive to image contrast variations which, in turn, makes the setting of thresholds exceedingly difficult. Some kind of sensitivity to image contrast is common to all corner operators that are based on the local autocorrelation of image intensity values and/or image gradient values.

6.2.4 Parameter tuning

The Harris corner detector is implemented to have parameters, which can be used to control the algorithm. In total there are three parameters: sigma, threshold and radius. These parameters are defined as follows:

Sigma: this parameter is used to define the standard deviation of the Gaussian function. This Gaussian function is called the kernel and is actually used for smoothing.
Threshold: this parameter is used to decide which pixels are corners. Only those above the threshold are considered as corners.
Radius: the radius is used to define the neighbourhood that needs to be considered. Only in this neighbourhood a local maxima will be found.

Table 18: Specification of the type of corners.

Type	Position	Total number
Inner eyebrow corner	left + right	2
Outer eyebrow corner	left + right	2
Inner eye corner	left + right	2
Outer eye corner	left + right	2
Mouth corner	left + right	2

To find the optimal values for each parameter, the detection algorithm is tested on a set of 20 images of frontal faces from the Carnegie Mellon face database. Before the test could be done, we have to define what kind of corners we want the algorithm to detect. In total 12 corners are identified (see Table 18).

Table 19: Harris test parameters.

No.	Parameter name	Values
1	Sigma	05 – 4.0
2	Thresh	800 – 1200
3	Radius	0.5 – 4.0

After identifying the correct corners a reference database for each image is made by manually selecting the concerning points. Testing is done by varying one parameter and setting the other parameters to a constant. The results of this test are shown in Table 19, Table 20 and Table 21. The last column actually shows the rate of how many of the twelve identified type of points are detected.

Table 20: Test result by varying sigma.

Sigma	TDC	TCC	TNC	CDR
0.5	151	31,7	119,8	82,50
1.0	279	66,9	212,1	90,00
1.5	347	82,0	264,7	85,83
2.0	425	101,8	323,4	84,17
2.5	460	86,2	374,1	78,33
3.0	519	97,6	421,1	74,17
3.5	537	84,3	452,4	61,67
4.0	583	93,8	489,4	57,50

TDC: Total Number of Detected Corners
TCC: Total Number of Correct Corners
TNC: Total Number of Unimportant/False Corners
CDR: Corner Detection Rate in percentage

Table 21: Test result by varying threshold.

Threshold	TDC	TCC	TNC	CDR
800	279	68,0	212,1	90,00
900	252	61,4	190,2	89,17
1000	230	55,2	175,1	85,83
1100	211	50,5	160,7	82,50
1200	192	44,6	147,3	77,50

TDC: Total Number of Detected Corners
TCC: Total Number of Correct Corners
TNC: Total Number of Unimportant/False Corners
CDR: Corner Detection Rate in percentage

Table 22: Test result by varying radius.

Radius	TDC	TCC	TNC	CDR
0.5	279	68,0	212,1	90,00
1.0	51	11,0	40,1	80,00
1.5	46	10,6	35,5	80,00
2.0	41	9,3	31,8	75,00
2.5	36	8,5	27,1	70,00
3.0	33	8,1	24,5	67,50
3.5	30	7,6	22	63,33
4.0	26	6,6	19,6	55,00

TDC: Total Number of Detected Corners
TCC: Total Number of Correct Corners
TNC: Total Number of Unimportant/False Corners
CDR: Corner Detection Rate in percentage

The optimal values for the Harris detection algorithm applied on face images are chosen corresponding to the best detection rate. In the final system the values chosen for the Harris corner algorithm is 1.0 for the sigma, 1000 for the threshold and 0.5 for the radius.

6.3 Sojka corner detection

6.3.1 Theoretical background

[Sojk03] The main idea for a new algorithm arose from the imperfection of existing corner detection algorithms. The majority of the existing direct corner detectors determine the values of a corner response function like the Harris corner detector. The value of a given point in an image is computed by examining the function of brightness and/or its derivatives in a certain neighbourhood of this point. Although the neighbourhood is not mentioned in all the detection algorithms, it is somehow determined when the derivatives are computed using the mask of a certain size or when the convolution is computed with the Gaussian function of certain σ. The value of the corner response function usually reflects the angle and the contrast of the corner. Points at which the value of the corner response function is greater than a chosen threshold, and also the function exhibits its maxima, are detected as corners. There are two drawbacks in this approach:

- Consider the situation depicted in Figure 29. In the areas, A, B and C, the magnitude of the gradient of brightness is non-zero. To determine the angle of the corner at point *Q*, only a part of the information contained in the neighbourhood $\Omega(Q)$ is relevant. Area A is the most important part, where area B is less substantial and area C is even almost irrelevant. Existing corner detectors do not take into account such situations and

unselectively examine the whole neighbourhood of *Q*. This may lead to incorrect decision of whether or not *Q* is a corner. This drawback cannot be avoided by using small neighbourhoods. The neighbourhood has to be big enough to make proper decision whether a candidate is a corner or not. In a small neighbourhood, noise affects the precision and the reliability of the computations.

- A considerable drawback is caused by thresholding the values of the corner response function. Suppose we can measure the angle α (see Figure 29). In digital images, we can do this with a limited precision. It is clear that if the difference $|\pi - \alpha|$ is less than the precision that can be achieved under the conditions of the measurement, the point should not be detected as a corner. Known corner algorithms do not check the angle value and use only the corner response function for thresholding. This function combines the angle and the contrast of the corner; it may happen that a small value of the difference is compensated by a high value of contrast, which leads to incorrect detections of corners on contrast edges. This problem cannot be avoided by increasing the threshold, which in turn will lead to missing the corners with a lower contrast.

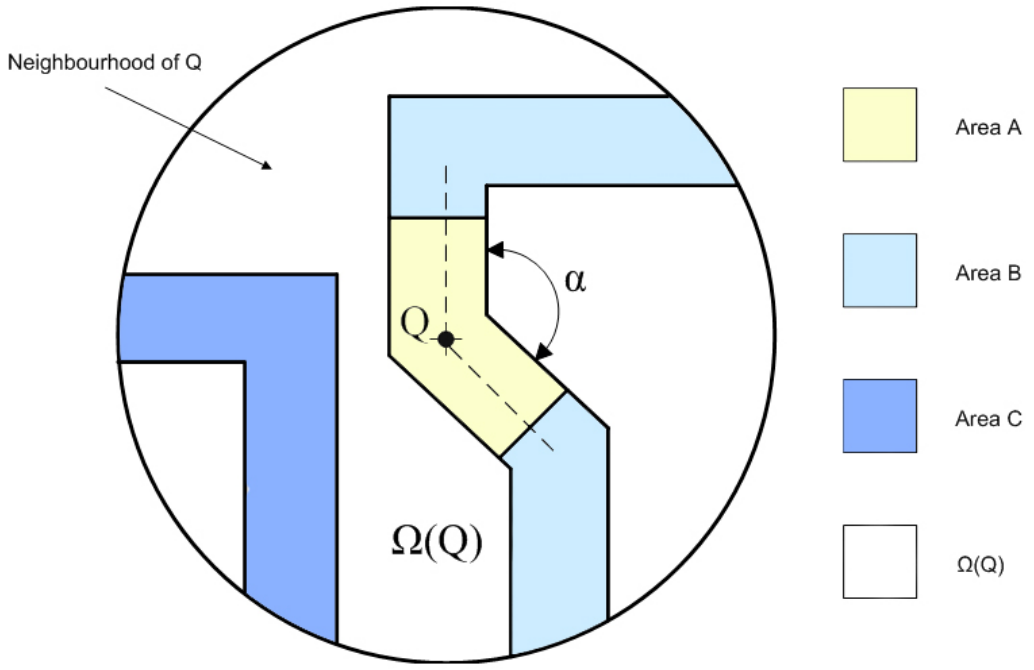


Figure 29: Neighbourhood of point *Q* on verifying the existence of corner at point *Q*. In the colored area A, B and C, the magnitude of the brightness gradient exhibits non-zero values. Only area A however is relevant for determining the angle of corner (denoted by α) at *Q*.

Taking into account the drawbacks mentioned above, a new algorithm we call the Sojka corner detection algorithm is proposed for direct corner detection. This algorithm also determines the

corner response function that combines the angle and the contrast of the corner similar to existing algorithms. The function is designed in such a way that it exhibits its local maxima at corner points. The main new features of the algorithm are the following:

- Information contained in the neighbourhood $\Omega(Q)$ is exploited selectively. It determines which areas are relevant for determining whether or not *Q* is a corner. It is done by introducing the probability $\mathrm{P}_{SG}(X)$ of the event that a point $X \in \Omega(Q)$ belongs to the approximation of the straight segment containing *Q* of the isoline of brightness. In Figure 29, for example, the values of $\mathrm{P}_{SG}(X)$ are high at the points in area A; at all other points the values of $\mathrm{P}_{SG}(X)$ are low. The values of $\mathrm{P}_{SG}(X)$ can be computed from the values of brightness and its gradient by making use of the Bayesian estimations.
- Explicit computations of the corner angle are included in this Sojka corner detection algorithm. The expected precision of the angle measurement is estimated. A point *Q* can only be accepted as a corner if the difference is significantly greater than the estimated precision of the angle measurement.
- Computation of a quantity expressing the obviousness of the corner is done. This quantity value, in essence, characterizes the size of the area that is relevant for deciding whether or not *Q* is a corner and the magnitude of the gradient of brightness in this area. A point can only be accepted as a corner if its 'appearance' is greater than a predefined threshold.

6.3.2 Corner model

A function $\psi(\xi)$ is defined to describe the values of brightness in the direction across the edges and its derivative is defined positive with a single maximal extremum at $\xi = 0$, which is regarded as an edge point. The edge is usually oriented by the rule that the higher brightness lies to the left. The corner is an intersection of two non-collinear straight edges. In the corner model of Sojka, the gradient of brightness along the edge that comes into the corner and the edge that comes out from the corner are defined as angle φ_1 and φ_2 respectively, where $\varphi_1, \varphi_2 \in \langle 0, 2\pi)$. Let the gradient vectors of the edges' gradient be represented by $\mathbf{n}_i = (\cos\varphi_i, \sin\varphi_i),\ i = 1,2$. The axis of a corner is a line through a corner point, which halves the angle between both edges, therefore also the angle between φ_1 and φ_2. The corner axis is oriented in such a way that it runs in the direction of increasing brightness. This means that the angle between the corner axis and φ_i is not greater than $\pi/2$. The directions of the gradient are different at the points lying to the left and to

the right of the corner axis. Consider the image containing a single corner point C. The brightness function is described by the following:

$$b(X) = \begin{cases} \min\{\psi(\mathbf{n}_1 \cdot (X - C)), \psi(\mathbf{n}_2 \cdot (X - C))\} & \text{if } \mathbf{n}_1 \text{ x } \mathbf{n}_2 \geq 0 \\ \max\{\psi(\mathbf{n}_1 \cdot (X - C)), \psi(\mathbf{n}_2 \cdot (X - C))\} & \text{otherwise} \end{cases}$$

(Eq. 6.13)

The term (X-C) expresses the distance of *X* from the edge. The corner is either convex or concave; concave if $\mathbf{n}_1 \text{ x } \mathbf{n}_2 > 0$ and convex if $\mathbf{n}_1 \text{ x } \mathbf{n}_2 < 0$.

6.3.3 Corner decision

Discussions are simply considered for two cases: continuous and discrete. In the case of a continuous and error-free representation of the image, *X* belongs to a straight isoline segment containing *Q* if the following conditions are satisfied:

- The brightness at *X* is equal to the brightness at *Q*, i.e. $\Delta b(X) = 0$.
- The line p_X passes through *Q*, i.e. $h(X) = 0$.
- For the angle difference $\Delta\varphi(X)$, the inequality $0 \leq \Delta\varphi(X) \leq \pi/2$ holds.
- The conditions above are satisfied not only at X, but also at all other points of the line segment $\overline{QX}$.

In the case of a discrete representation of the image, first a probability, $\mathrm{P}_{SG}(X)$, is introduced to denote the event that *X* belongs to the approximation of a straight line segment containing *Q*. We define:

$$\mathrm{P}_{SG}(X) = \min_{Y \in \overline{QX}} \{\mathrm{P}(BrQ \mid \Delta b(Y))\mathrm{P}(DirQ \mid h(Y))\mathrm{P}(AngQ \mid \Delta\varphi(Y))\}$$

(Eq. 6.14)

Explanation of the probabilities introduced in Eq. 6.31 can be found in the table (see Table 23) below.

Table 23: Explanations and relations among the expressions in Eq. 6.14.

Probabilities		**Description**	**Dependency**
$\mathrm{P}(BrQ \mid \Delta b(Y))$	$Y \in \overline{QX}$	Denotes the probability of the event that *Y* is a point of the approximation of the isoline with brightness $b(Q)$ thus *Y* is point of $\overline{QX}$.	Independent event. Since the neighbourhood area may contain one or more corners, the isoline segment with brightness $b(Q)$ need not generally aim at *Q*.
$\mathrm{P}(DirQ \mid h(Y))$		Denotes the probability of the event that *Y* is a point of the approximation of the isoline segment (not necessarily with brightness $b(Q)$) that aims at point *Q*.	Independent event. An isoline segment aiming at *Q* may generally have an arbitrary brightness.
$\mathrm{P}(AngQ \mid \Delta\varphi(Y))$		Denotes the probability of the event that *Y* belongs to the area of possible corner at *Q*.	Independent event. Points may exist that do not belong to the corner area despite the fact that the conditions, that the brightness difference and deviation are zero, are both satisfied (see Figure 30).

Table 23 shows that the three probabilities introduced in Eq. 6.14 are of independent events. The last probability is introduced due to the fact that points that satisfy both the conditions $\Delta b(Y) = 0, h(Y) = 0$ do not always belong to the corner area (see Figure 30). This fact is detected by the condition $0 \le \Delta\varphi(Y) \le \pi/2$, which is not satisfied for *Y*.

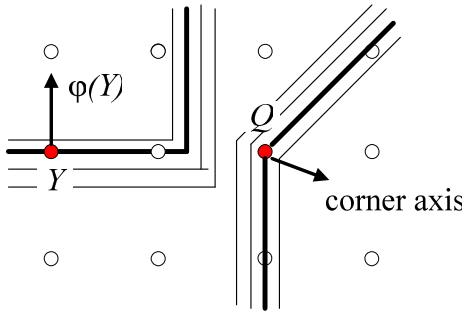


Figure 30: Although the conditions $\Delta b(Y) = 0, h(Y) = 0$ are both satisfied at *Y*, *Y* does not belong to the area of the corner at *Q*. This is detected by the fact that the condition $0 \le \Delta\varphi(Y) \le \pi/2$ is not satisfied for *Y*. Small circles show the position of pixel points.

Suppose that *X* is a point lying on a straight isoline segment containing *Q*, which means that all the points of the line segment $\overline{QX}$ also lie on the isoline segment. Figure 31 illustrates the fact the conditions do not suffice to decide whether or not X belongs to an isoline segment containing *Q*. Thus, the probability of *Q* and *X* connected by a segment isoline cannot be greater than the probability than any arbitrary point of the isoline segment $\overline{QX}$ connected to *Q*. The operator 'min' in Eq. 6.14 corresponds to the idea that if the event (event that a point belongs to the approximation of the straight line segment containing *Q*) occurs at the point with lowest probability, it will also occur at the remaining points of the line segment $\overline{QX}$.

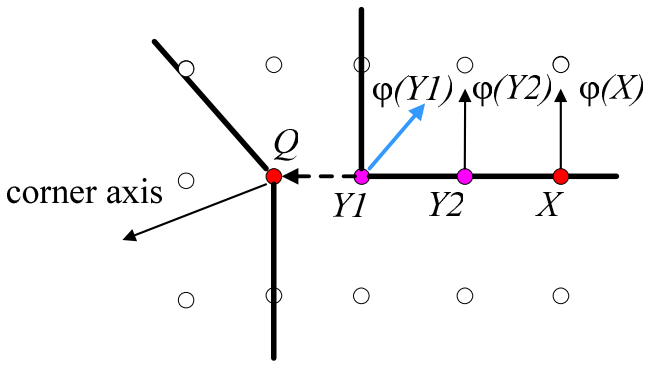


Figure 31: Isoline segment XY1 aiming at *Q*. Although all three conditions are satisfied by X and Y2, Y1 shows the contrary (the angle of difference must be less than π/2).

Substituting the estimations from $\mathrm{P}(BrQ \mid \Delta b(Y))$ and $\mathrm{P}(DirQ \mid h(Y))$ (discussed in [Sojk03]) into Eq. 6.14, we find:

$$\mathrm{P}_{SG}(X) = A_0 \min_{Y \in QX} \left\{ p_d(\beta^{-1}(\Delta b(Y)))\, p_d(h(Y)) \mathrm{P}(AngQ \mid \Delta\varphi(Y)) \right\} \qquad \text{(Eq. 6.15)}$$

Let us determine at *Q* the 'angle of break' of the isoline that passes through *Q*. We thereby need to examine the values of $\varphi(X)$ in Ω. Since the relevance of the value of $\varphi(X)$ at *X* not only depends on the probability $\mathrm{P}_{SG}(X)$, but also on the distance between the points Q and X, a positive weight function $w_r(r(X))$, where $r(X)$ stands for the distance, is introduced. The relevance is expressed by the following:

$$w(X) = \mathrm{P}_{SG}(X) w_r(r(X)) \qquad \text{(Eq. 6.16)}$$

To determine the angle of break at *Q*, first compute the quantities μ_φ and σ_φ^2, which determines the direction of the corner axis and the weighted square value of the difference between the direction of the gradient brightness $\varphi(X_i)$ at X_i and μ_φ. We define:

$$\mu_\varphi = \frac{\sum_{X_i \in \Omega} w(X_i)\varphi(X_i)}{\sum_{X_i \in \Omega} w(X_i)} \qquad \text{(Eq. 6.17)}$$

$$\sigma_\varphi^2 = \frac{\sum_{X_i \in \Omega} w(X_i)\left[\varphi(X_i) - \mu_\varphi\right]^2}{\sum_{X_i \in \Omega} w(X_i)} \qquad \text{(Eq. 6.18)}$$

Now use $\mu_\varphi(Q), \sigma_\varphi^2(Q)$ to express explicitly the values that were computed for a particular corner candidate *Q*. We define the functions:

$$Corr(Q) = g(Q)\sigma_\varphi^2(Q) \qquad \text{(Eq. 6.19)}$$

$$Appar(Q) = \sum_{X_i \in \Omega} \mathrm{P}_{SG}(X_i) g(X_i) \mid \varphi(X_i) - \mu_\varphi \mid \qquad \text{(Eq. 6.20)}$$

6.3.6 The practical application of the algorithm

In practical computation the Sojka corner algorithm is realized as follows. First, the magnitude and the direction of the gradient of brightness $g(X)$ and $\varphi(X)$, respectively are computed. The derivatives $\partial b(X)/\partial x, \partial b(X)/\partial y$ are replaced by the differences, which are computed using the following masks:

$$\begin{bmatrix} -0.1 & 0 & 0.1 \\ -0.3 & 0 & 0.3 \\ -0.1 & 0 & 0.1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0.1 & 0.3 & 0.1 \\ 0 & 0 & 0 \\ -0.1 & -0.3 & -0.1 \end{bmatrix}$$

Only those pixel points at which the magnitude of the gradient of brightness is greater than a predefined threshold are considered as candidates for corners. The candidate corners are then examined by determining the values $\mu_\varphi(Q), \sigma_\varphi(Q), Corr(Q), Appar(Q)$. The candidate at which the value $Corr(Q)$ exhibits its local maximum and at which the values of $\sigma_\varphi(Q), Appar(Q)$ are greater than chosen thresholds is determined as a corner. The probability density p_d and the weight function w_r are fixed estimated functions, where the probability density p_d is a normal distributed function with zero-mean and w_r is a Gaussian function.

Sojka corner detection algorithm:

1. Compute magnitude of the gradient of brightness *g(X)* and direction of the gradient of brightness *φ(X)*.

2. For all pixel points *Q*:
   if *g(X) > predefined threshold* then *Q* is a candidate corner.

3. For all candidate corners *Q*:
   - Compute the direction of the corner axis: $\mu_\varphi(Q)$.
   - Compute the weighted square value of the difference between $\varphi(X_i)$ at X_i and μ_φ as: σ^2_φ.
   - Using $\mu_\varphi(Q)$ and σ^2_φ, compute *Corr(Q)* and *Appar(Q)* according to equations Eq. 6.19 and Eq. 6.20.

4. Candidate corner *Q* is determined as a corner if:
   Corr(Q) exhibits its local maximum and
   Appar(Q) > predefined appearanceThreshold

Figure 32: Sojka corner detection algorithm.

The neighbourhood $\Omega(Q)$ is declared square-shaped with *Q* in the centre. The declared size of the neighbourhood is not so crucial since the effective size is always determined adaptively by the values of $\mathrm{P}_{SG}(X)$. In practice, ψ is an unknown function so the value of $\beta^{-1}(\Delta b(Y)) = d(Y)$ cannot be determined exactly, but can be estimated instead. With respect to the corner model, we yield:

$$\beta^{-1}(\Delta b(Y)) \approx \frac{2\Delta b(Y)}{g(Y) + g(Q)} \qquad \text{(Eq. 6.21)}$$

For computing the probability $\mathrm{P}(AngQ \mid \Delta\varphi(X))$ the angle difference $\Delta\varphi(X)$ is needed. But the direction $\mu_{\varphi}(Q)$ of the corner axis is not known, so it will be approximated by $\varphi(Q)$. Taking into account that the probability $\mathrm{P}(AngQ \mid \Delta\varphi(X))$ is either one or zero and substitute Eq. (6.21) into Eq. (6.15), we easily obtain:

$$\mathrm{P}_{SG}(X) = \begin{cases} \approx \min\limits_{Y \in \overline{QX}} \; p_d \; \dfrac{2\Delta b(Y)}{g(Y) + g(Q)} \; p_d(h(Y)) & 0 \le \Delta\varphi(X) \le \pi/2 \\ 0 & \textit{otherwise} \end{cases}$$

(Eq. 6.22)

The computation of $\mathrm{P}_{SG}(X)$ may be carried out effectively by proceeding from *Q*, which is the centre, to the border of $\Omega(Q)$. Once the values of $\mathrm{P}_{SG}(X)$ is known for all points in $\Omega(Q)$, the values of $\mu_{\varphi}(Q), \sigma_{\varphi}(Q), \; Corr(Q), \text{and } Appar(Q)$ can be easily computed using Eq. 6.17 – 6.20.

Finally, if we assume that the size of $\Omega(Q)$ is *MxM* pixels, it can be noted that values of $\mathrm{P}_{SG}(X)$ for all points in $\Omega(Q)$ can be computed in $\theta(M^2)$ time. The same time is also needed for the computation of other values, which means that a corner candidate is processed in $\theta(M^2)$ time. To speed up the computations one can neglect the small values of $\mathrm{P}_{SG}(X)$.

6.3.7 Parameter tuning

Before this method can be applied to faces for extracting FCPs, the parameters of the Sojka algorithm must be optimized for the best result. Following we will show the definition of the seven parameters of this algorithm described at [Sojk03b].

HalfPsgMaskSize: *the half of the mask size.*
The overall mask size, i.e., the size of the neighbourhood (2*halfPsgMaskSize + 1) is used to decide whether or not a candidate is a corner. Generally, the bigger masks give better results but the computation may be longer. The usual values of halfPsgMaskSize vary from 4 to 7, which give the overall size of the mask between 9x9 and 15x15 pixels.

CorrAngleThresh*: the threshold for the "angle of break" of the boundary at the corner point.*
Only a point at which the boundary is broken more than is required by this threshold may be accepted as a corner. Usual value is approximately 0.5, which is an angle size in radians. The value of this threshold is more or less stable for all images. Therefore, there is no need to experiment too much unless the detected corners must have a specific angle. As an indication, higher value of this threshold must be chosen if small masks are used since in small masks, the precision measuring the angles is generally lower than in greater masks.

NoiseGradSizeThresh*: the threshold for the size of the gradient of brightness.*
All the values of the gradient size less than this threshold are considered to be a noise and the corresponding points are therefore excluded from processing. Higher values of this threshold contribute to the reduction of influence of noise. At the same time, however, the less obvious corners may be missed. Remember that even in the synthetic images you have a noise that is caused by dividing the image into pixels. Typically, the value of this parameter varies between 0.04 and 0.08 of the image range (the difference between the maximal and minimal value of brightness in the image). If the less contrast corners are also wanted, this value should have a lower value. If corners caused by noise are detected in case it is not desired, this value should be increased. Note that higher values lead to faster computation since the algorithm examines a lower number of candidates.

AppearanceThreshold*: the threshold for the "appearance" (obviousness) of corner.*
The appearance combines the contrast, size and the shape of the possible corner area. Only the points whose appearance is greater than this threshold can be accepted as corners. The value of this threshold must again be properly adjusted. The typical value of this threshold varies usually

between 0.0 and 5.0. If the algorithm detects too many corners, increase the value of this threshold and vice versa. Remark that the value of this threshold may be negative.

SigmaD*: sigma for the normal distribution of the probability density* P_d.
From the theoretical point of view, it follows that the value should be less than 1.0. Values between 0.7 and 1.0 were confirmed as optimal also experimentally. The value of 0.95 proved to be suitable for most images. There is no need to experiment with this value too much unless absolutely best possible detection is wished.

SigmaR*: Sigma for the Gaussian expressing the function* w_r, *i.e., the weights depending on the distances from the candidate that is just being processed.*
For the above mentioned sizes of the mask, use the values between 2.5 and 3.5, typically 3.0. Usually, the value is not critical.

HalfExtMaskSize*: this parameter is the size of the area in which it is checked whether the corner response function Corr() has its maximum at the point just being tested.*
Typical value of this parameter is 1, which means that the maximum is checked in the area of 3x3 pixels with the point being tested in its center. Tests with this parameter should start with value 1. If problems with multiple detections of the same corner arise, this value should be increased to 2 or 3. The value of this parameter must always be less than or equal to the value of the parameter *HalfPsgMaskSize*.

Table 24: Sojka test parameters.

Parameter name	Range
HalfPsgMaskSize	4 (fixed)
CorrAngleThresh	0.5 (fixed)
NoiseGradSizeThresh	3 ~ 23 (variable)
AppearenceThreshold	0 ~ 5 (variable)
SigmaD	5 (fixed)
SigmaR	2 (fixed)
HalfExtMaskSize	5 (fixed)

Having defined the parameters, we can apply the algorithm on faces to see which parameters give the best results. From the definition, it can be seen that not all parameters have equal influence on the outcome of the detection. Most of them can be set to a default value. The two parameters with most influence are NoiseGradSizeThresh and AppearenceThreshold. In the test, we varied the

values these two. Following table shows the test parameters. The test set is the same as the one used for tuning Harris' parameters.

Table 25: Sojka test result.

noiseGradSizeThr.	appearanceThr.	TDC	TCC	TNC	CDR
3	0	60.07	17	42.7	79.17
3	5	16.47	5.5	9.3	33.33
4	2	33.03	10.6	21.9	60.00
6	5	8.27	2.3	5.2	15.83
13	5	1.00	0.4	0.8	2.50
14	2	9.50	2.8	5.1	19.17
18	5	65.40	18.7	52.3	77.50
19	3	73.10	20.8	59.2	81.67
19	4	58.77	18	46.7	77.50
20	0	98.37	24.2	83.8	88.33
20	2	72.93	20.2	58.3	83.33
20	4	47.33	14.8	36.5	69.17
21	0	89.50	22.9	73.1	89.17
21	1	76.33	21.2	61.3	85.83
21	2	60.87	18.2	47.8	79.17
21	3	46.93	15.2	35.3	74.17

TDC: Total Number of Detected Corners
TCC: Total Number of Correct Corners
TNC: Total Number of Unimportant/False Corners
CDR: Corner Detection Rate in percentage

The test results are shown in Table 25. It can be concluded that the detection rate decreases when the noise threshold is kept fixed and the appearance threshold is increasing. From the test results, we can see that a higher noiseGradSizeThresh (without looking at the appearanceThreshold) leads to a lower number of total detected corners. A lower value of appearanceThreshold (without looking at noiseGradSizeThresh) leads to a higher number of total detected corners and total correct detected corners. So, appearanceThreshold should get the value 0. The results show that the highest correct detection rate is achieved where noiseGradSizeThresh = 21 and AppearanceThreshold = 0 (highlighted in blue). These values are used in the final implementation.

7

Facial Characteristic Points Detection – Classification

As mentioned in the previous chapter, after detecting the corner points it is still needed to extract the positive corners from the total number of candidate corners. To achieve this we use the RVM classifier discussed in the previous chapters. This classification model needs to be trained with samples of the corners we want the classifier to extract. Analogous to the steps of training RVM for face detection, we will discuss the training for corners. In section 7.1 the preparatory steps preceding the training will be discussed. In section 7.2 the training process with the boosting algorithm will be considered and section 7.3 copes with the training and test results.

7.1 Feature vector extraction

The training samples needed to train the RVM corner classifier will be taken from images with full frontal faces. The resolution of these face images are actually determined by the face detector, which is the size of 64x64. It is discussed [Chan04] that this size is a trade-off between computational cost and minimum resolution in order to guarantee that information about eyes, nose and mouth is not lost in too small image versions. That means with this size it is still possible to see enough details around the facial feature corners, which is strongly necessary for extracting feature vectors to train RVM.

Before we can create the database of corners, we need to know which size of the region around the corners needs to be obtained. The solution to this is based on experimental results and testing. Thus, we rely on the results of our application. If the region around the corner is too small, we cannot extract proper features from it. This is because there is a lot of fuzziness around the facial feature corner points. A small window size does not contain enough detail to differentiate well

from other non-corner regions. We do not want the size to be too big because that would be computationally unattractive. It is known that the size of 13x13 image pixels has been successfully applied by other applications. Thus, we will use this size for the samples to be extracted from the face database. We manually extracted points for the sample database. For each corner point, from which there are 12, we made a database for training and another one for testing. The databases we extracted the corner points from are the BioID dataset and the Carnegie Melon dataset. More details about the training and testing databases used for RVM can be found in Table 26.

Table 26: Dataset description of different corners.

Database description:		
Database source:	BioID/Carnegie Melon	
Database sample size:	64x64 pixels	
Extracted sample size:	13x13 pixels	
Training:	2-class (0/1)	
Type	**Nr. of samples (0/1)**	
	Training set	**Testing set**
Left eye inner corner	500/500	500/500
Right eye inner corner	500/500	500/500
Left eye outer corner	500/500	500/500
Right eye outer corner	500/500	500/500
Nose left corner	250/250	259/250
Nose right corner	250/250	326/350
Mouth left corner	500/500	500/495
Mouth right corner	500/500	500/495

Having the databases with corner samples, we cannot train the RVM classifier with the intensity values of these samples. As we have seen before it does not work. We need a method that extracts essential features to provide for the training of the RVM. Even though the DCT feature extraction did not work for face detection, we apply it to the samples of the corners. DCT is short for the Discrete Cosine Transform, which is a technique used for compressing images. The DCT coefficient can be seen as feature vectors representing the whole image, which was a perfect property to consider in our application. It works better than in the case of face detection. However, the results were not good enough and thus unsatisfactory. We found that the DCT feature extraction method is too sensitive to illumination effects in the images. Disappointingly, it has to be concluded that DCT cannot be applied for this purpose.

At this point, we still need a method to extract the significant features from the images. We wonder if it is possible to employ the feature extraction method as used for face detection. This

method extracts simple rectangle features and with the boosting algorithm weak classifiers are combined to form a strong classifier. The problem of needing a method to extract the significant features from the training/testing database will be solved by this approach. So, we decided to utilize this method for our facial corner detection.

7.2 Training the Relevance Vector Machine with Evolutionary-AdaBoost

The steps taken for utilizing the boosting method with Haar-like features is analogous to that explained in Chapter 5. A general scheme for training the RVM for corner classification is shown in Figure 33.

The boosting algorithm selects a Haar-feature and this feature will be calculated for all samples in the training database. Note that the complete set of possible features (14140 features) is different and smaller than in the case of 24x24 sample dimension. A weak RVM classifier will be trained for this single feature and verified according the boosting principle. If this feature may not satisfy the given conditions another feature will be selected and trained. In case it does satisfy the given specification like having a positive detection rate greater than the given value, the feature will be added to the final list of weak classifiers of which the corresponding parameter values will be saved. These parameter values are the WUX values of the RVM classifier

Every time a candidate feature is found, the obtained set of final weak classifiers will be evaluated on a test set that is distinct to the training set. If the overall true positive rate and overall false positive rate meet the requirements the training ends. Otherwise, the algorithm will verify whether the added feature contributes to the existing set. This is done by assessing the overall false positive rate. This should be lower than the previous run of the selected final weak classifiers. If this is not the case then the feature will be discarded. The training algorithm will continue with searching for another weak feature to add to the final set of weak classifiers to form a strong classifier. The training is said to converge if the final strong classifier satisfies the given overall true detection rate and overall false detection rate.

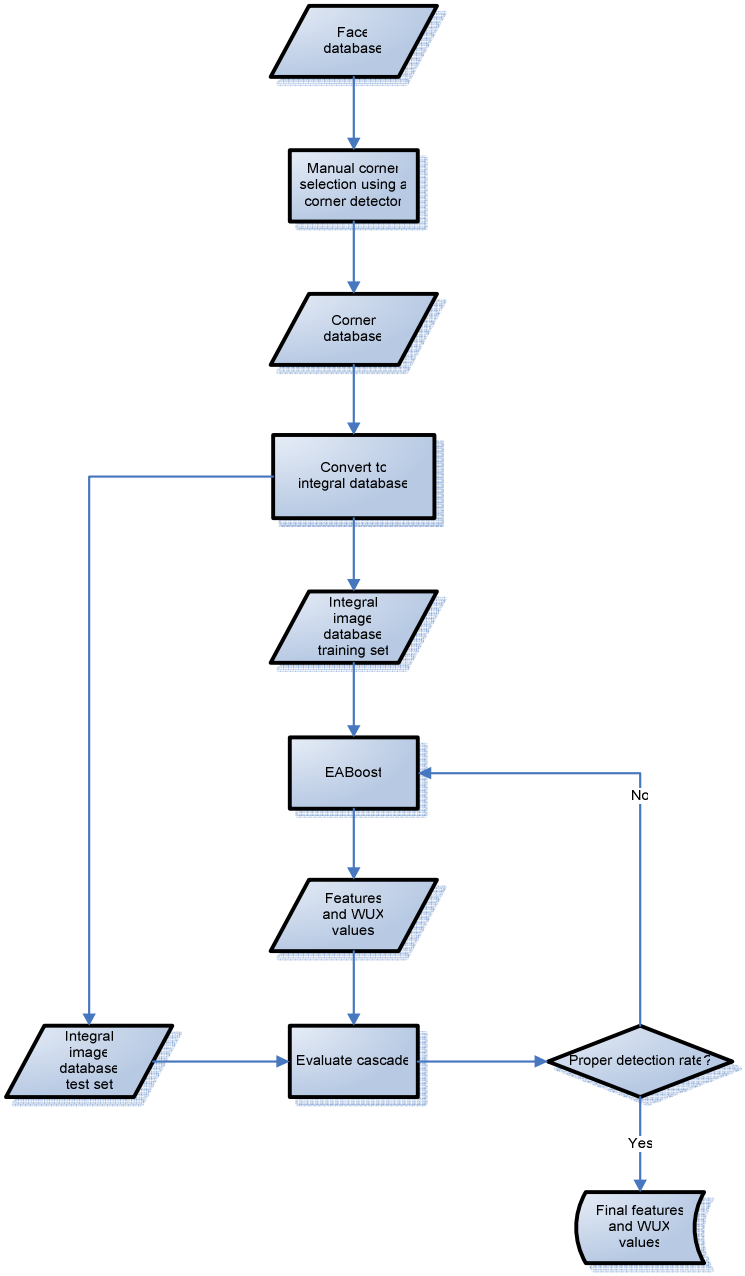


Figure 33: Scheme representing the training of weak classifiers with the WUXTRAP for FCP. extraction.

The chosen parameters for boosting the weak features are depicted in Table 27. The chosen RVM model is exactly the same as the one used for face detection with the same kernel.

Table 27: EABoost training parameters for all type of corners.

EABoost parameters	Values and description
Population size	250
Crossover/Mutation rate	0.20/0.80
Classifier/ kernel	RVM/ Laplace 4.0
Feature false positive rate	0.3
Feature true positive rate	0.75
Target false positive rate	< 0.05
Target true positive rate	> 0.90
Number of features needed	variable

7.3 Training and test results

Now it is time to look at some results acquired after the training algorithm has converged. In the implementation of the boosting algorithm it is already known that an Evolutionary Search algorithm is applied to speed up the selection process of the training algorithm. The training application is implemented such that it only stops when the required rate is gained. The single weak features are selected from a training set, which they have to satisfy the given rate for the training set. Then the evaluation is done on a testing set to assure that the same rate can be achieved from unknown samples. The evaluation results on the testing set are depicted in the following (Table 28).

Table 28: The evaluation results from the EABoost training.

Type	Acronym	TPR	FPR
Left eye inner corner	LEIC	0.922	0.02
Right eye inner corner	REIC	0.89	0.106
Left eye outer corner	LEOC	0.916	0.064
Right eye outer corner	REOC	0.892	0.034
Left nostril corner	NLC	0.935	0.06
Right nostril corner	NRC	0.893	0.049
Mouth left corner	MLC	0.955	0.049
Mouth right corner	MRC	0.916	0.049

TPR = True Positive Rate

FPR = False Positive Rate

Table 28 lists the training results after boosting the RVM classifier using weak classifiers. This should not be confused with the results of chapter 6. In chapter 6 only the performance of the

corner detectors is evaluated. The corner detectors are implemented to select candidate corners, which also includes the corners listed in the table above. The RVMs are trained to subtract the true corners from the false corners (corners that are not listed in the table above.). In other words RVM has to classify the candidate corners.

At this point we have extracted a part of the total number of points that needs to be extracted for the face model (see Chapter 3). The rest of the points cannot be extracted using this approach which detects candidate corners with the Harris and Sojka corner detector and then classify them with RVMs. The problem of the remaining landmarks (corner points) is that they cannot really be considered as a corner point. In most cases they cannot be detected by the corner detectors. Applying the technique of corner detectors with RVM would certainly fail to work.

Another problem would also be training the RVM. Features take on extreme shapes at these remaining non-stable points and are way fuzzier than the defined points in the table. This would make the extraction of a good database for training the RVM very difficult. To extract the remaining the points we have employed another technique called the integral projection method. This technique projects the image into the vertical and horizontal axes, from which we are able to obtain the boundaries (in this case the facial features' boundaries, which happens to contain the points we are interested in). The boundaries of the features have relatively high contrast compared to its near environment. This property can be perfectly explored for extracting the remaining points. Further details can be found in the next chapter.

8

Hybrid Projection for FCP Detection

This chapter describes a method for extracting other remaining FCPs, namely the projection method. First, section 8.1 gives a theoretical introduction to this method. In section 8.2 to 8.4 different types of the projection method is discussed. In the final solution the hybrid projection method is chosen to extract the remaining characteristic points. The method requires the corresponding feature as input because a larger region might influence the final detection results negatively. That is why section 8.5 discusses the algorithm and the method for extracting the facial features like eyes, etc. The results and findings are also contained in this section.

8.1 Theoretical foundation

Many applications exist which use the technique of projection functions to detect the boundary of different image regions. In our case we applied this technique in extend to the other approach to extract the FCPs. This is needed because as the test results of the previous chapter showed, not all FCPs could be extracted with the corner detection algorithms. Thus, we will use hybrid projection with the purpose of finding the missing FCPs. It can also be seen as an extra verification for the corner detection algorithm since it can also find FCPs that we have already extracted as described in the last chapter. In a face, it is observed that some of the landmarks (including corner points) have relatively high contrast and that is what makes the method suitable for FCP extraction. Using projection functions the image is actually presented by two 1D orthogonal projection functions. The dimension reduction from 2D to 1D reduces the computational load, which is very interesting property. Consider Figure 34, which sketches the model; in this case for the eye.

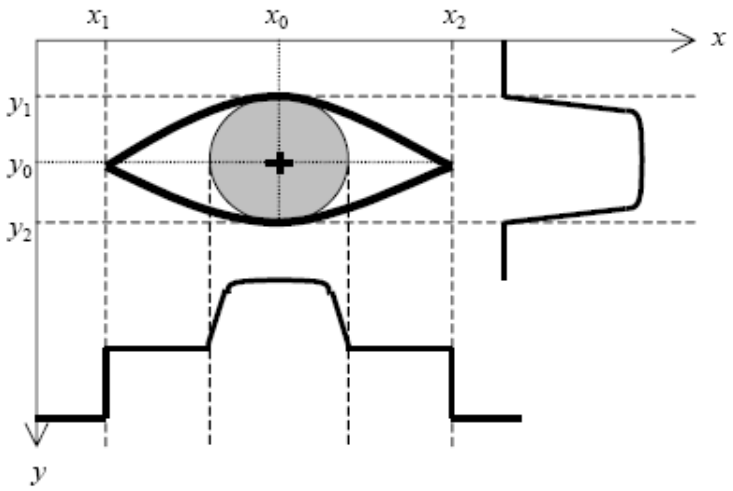


Figure 34: Model of an eye image. The bold lines on the side of the eye represent the integral projection of the image.

Suppose that *PF* is a projection function and ξ is a small constant. If the value of *PF* rapidly changes from v_0 to $(v_0 + \xi)$, then v_0 may lie at the boundary between two homogeneous regions. This property of *PF* can be well exploited for FCP detection. There exist a few projection functions and in the following sections we will discuss them more in detail. Among all the image projection functions used, the integral projection function (IPF) is the most popular one. Another one is the variance projection function (VPF) and we will also define a more generalized projection function, which combines IPF and VPF. For the optimal parameters of the generalized projection function we call it the hybrid projection function (HPF).

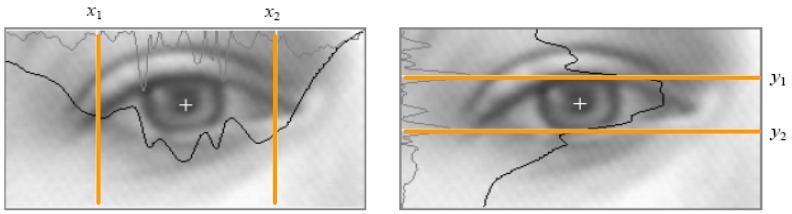


Figure 35: Use projection function to locate the boundaries of the facial feature.

8.2 Integral projection function

This type of projection function is the most popular one. However, in some cases like the one in Figure 36 it cannot well reflect the variation in the image. In this case it fails to capture the vertical variation of the image.

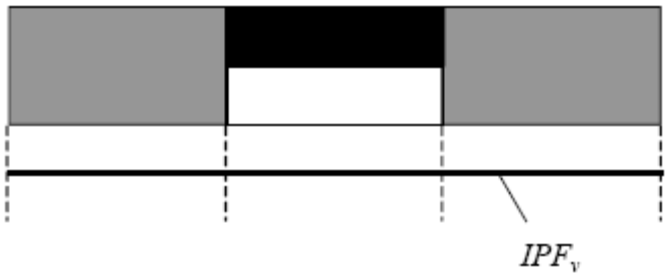


Figure 36: Case where IPF cannot retrieve the vertical variation.

[Feng98] Suppose *I(x, y)* is the intensity of an image pixel at location *(x, y)*. We denote $IPF_v(x)$ to be the vertical projection function projected on the vertical axis, and $IPF_h(x)$ to be the horizontal projection function projected on the horizontal axis. Both functions of the image pixel *I(x, y)* in intervals $[y_1, y_2]$ and $[x_1, x_2]$ can be defined respectively as:

$$IPF_v(x) = \int_{y_1}^{y_2} I(x, y)dy \quad \text{(Eq. 7.1)}$$

$$IPF_h(y) = \int_{x_1}^{x_2} I(x, y)dx \quad \text{(Eq. 7.2)}$$

More often, the mean vertical and horizontal projections are used, which are defined as:

$$MIPF_v(x) = \frac{1}{y_2 - y_1} \int_{y_1}^{y_2} I(x, y)dy \quad \text{(Eq. 7.3)}$$

$$MIPF_h(y) = \frac{1}{x_2 - x_1} \int_{x_1}^{x_2} I(x, y)dx \quad \text{(Eq. 7.4)}$$

The vertical and horizontal boundaries in the image can be identified according to the following:

$$B_v = \max \left| \frac{\partial PF_v(x)}{\partial x} \right| > T_v \quad \text{(Eq. 7.5)}$$

$$B_h = \max \left| \frac{\partial PF_h(x)}{\partial x} \right| > T_h \quad \text{(Eq. 7.6)}$$

PF can be any projection function for instance *IPF, MIPF, VIPF, HPF*. The parameter *T* in the equations is the given threshold. A set of B_v vertically divides the image into different regions and a set of B_h divides the image into different horizontal regions.

8.3 Variance projection function

To solve the problem mentioned in the previous section the variance projection function is introduced. The variance projection function (VPF) [Feng99] is more sensitive to the variation in the image than IPF does. VPF is also proved to be orientation and scale invariant. Another attractive property of VPF is that it is insensitive to random noise in the image. So, suppose that the facial feature, for instance the eye, is bounded by $[x_1, x_2]$ and $[y_1, y_2]$. Let us denote $VPF_v(x)$ and $VPF_h(x)$ to be the average of vertical projection and horizontal integral projection of image pixel I(x, y) in intervals $[x_1, x_2]$ and $[y_1, y_2]$. The projection functions in the vertical and horizontal direction are defined as:

$$VPF_v(x) = \frac{1}{y_2 - y_1} \sum_{y_{i=}y_1}^{y_2} [I(x,y) - MIPF_v(x)]^2 \qquad \text{(Eq. 7.7)}$$

$$VPF_h(y) = \frac{1}{x_2 - x_1} \sum_{x_{i=}x_1}^{x_2} [I(x,y) - MIPF_h(y)]^2 \qquad \text{(Eq. 7.8)}$$

Although VPF is more sensitive to the variation in the image than IPF, it still does not mean that VPF always works well. As a matter of fact, if we consider Figure 37; it is obvious that the vertical projection fails to expose the vertical variation of the image. The variation is presented as a flat line after the projection, which totally masks the true variation in the image.

Figure 37: Case where VPF fails to capture the vertical variation.

Similarly to IPF the vertical and horizontal boundaries can be computed along with Eq. 7.5 and 7.6. An illustration of VPF using synthetic eye image is shown below. Also random noise is added to the image to show the influence of random noise in the image to VPF.

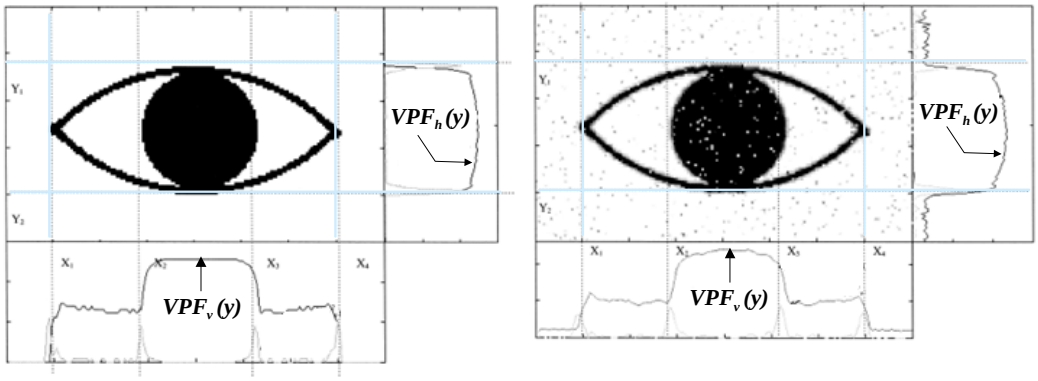


Figure 38: (Left) Synthetic eye image and its vertical and horizontal projection. (Right) Eye image is added with random noise and its vertical and horizontal projection.

8.4 Hybrid projection function

In the previous sections it is shown that both IPF and VPF have weaknesses. But it is also easy to find that they are complementary. The reason is that IPF considers the mean of the intensity values while VPF considers the variance of the intensity. This is shown in Figure 39, where VPF works better in one case and IPF on the other.

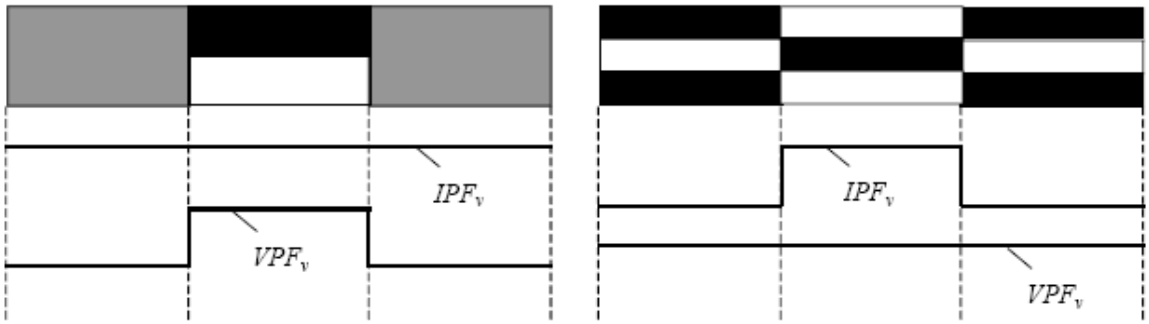


Figure 39: IPF and VPF complement each other in retrieving the vertical variation in some cases.

Now if we combine the results of IPF and VPF we yield a new projection function, which will be the generalized projection function (GPF) [Zhou02]. Denote $I(x, y)$ to be the intensity pixel value at location *(x, y)*. Also denote $GFP_v(x)$ and $GFP_h(x)$ to be the vertical and horizontal

generalized projection function of $I(x,y)$ in the intervals $[y_1, y_2]$ and $[x_1, x_2]$ respectively. We define:

$$GPF_v(x) = \alpha \cdot MIPF_v(x) + \beta \cdot VPF_v(x) \ , \qquad 0 \le \alpha, \beta \le 1 \qquad \text{(Eq. 7.9)}$$
$$GPF_h(y) = \alpha \cdot MIPF_h(y) + \beta \cdot VPF_h(y) \ ,$$

where α, β is used to control the contribution of IPF and VPF. To understand this relation we have Figure 40 and Figure 41 to demonstrate this. The first graphs (Figure 40) actually show the GPF in relation to IPF and VPF of an eye feature. This is also done for the mouth in Figure 41. In both cases α is set to the values [0, 0.25, 0.5, and 1]; $0 \le \beta \le 1$.

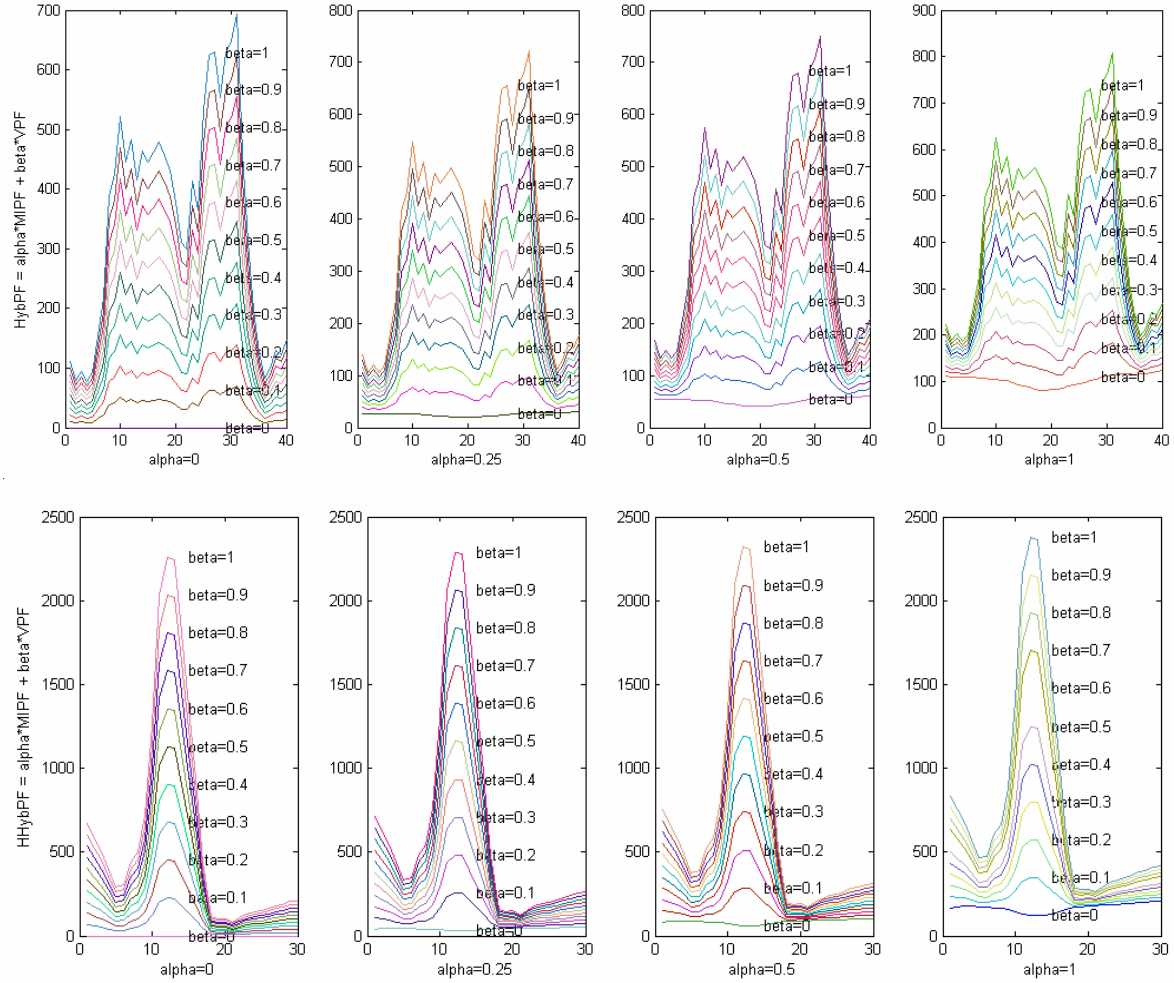


Figure 40: Eye image: (Top) Vertical projection (Bottom) Horizontal projection. For beta=0 the variational information cannot be captured. Clear variational differences can be noticed for alpha>0.3.

It is clear that IPF alone $(\alpha = 1, \beta = 0)$ is not enough for representing the variational information contained in the image. VPF, on the other hand, works pretty good even without IPF $(\alpha = 0, \beta = 1)$. But IPF does contribute to the overall GPF in that it still contains the mean of the intensity values which can be complementary to VPF.

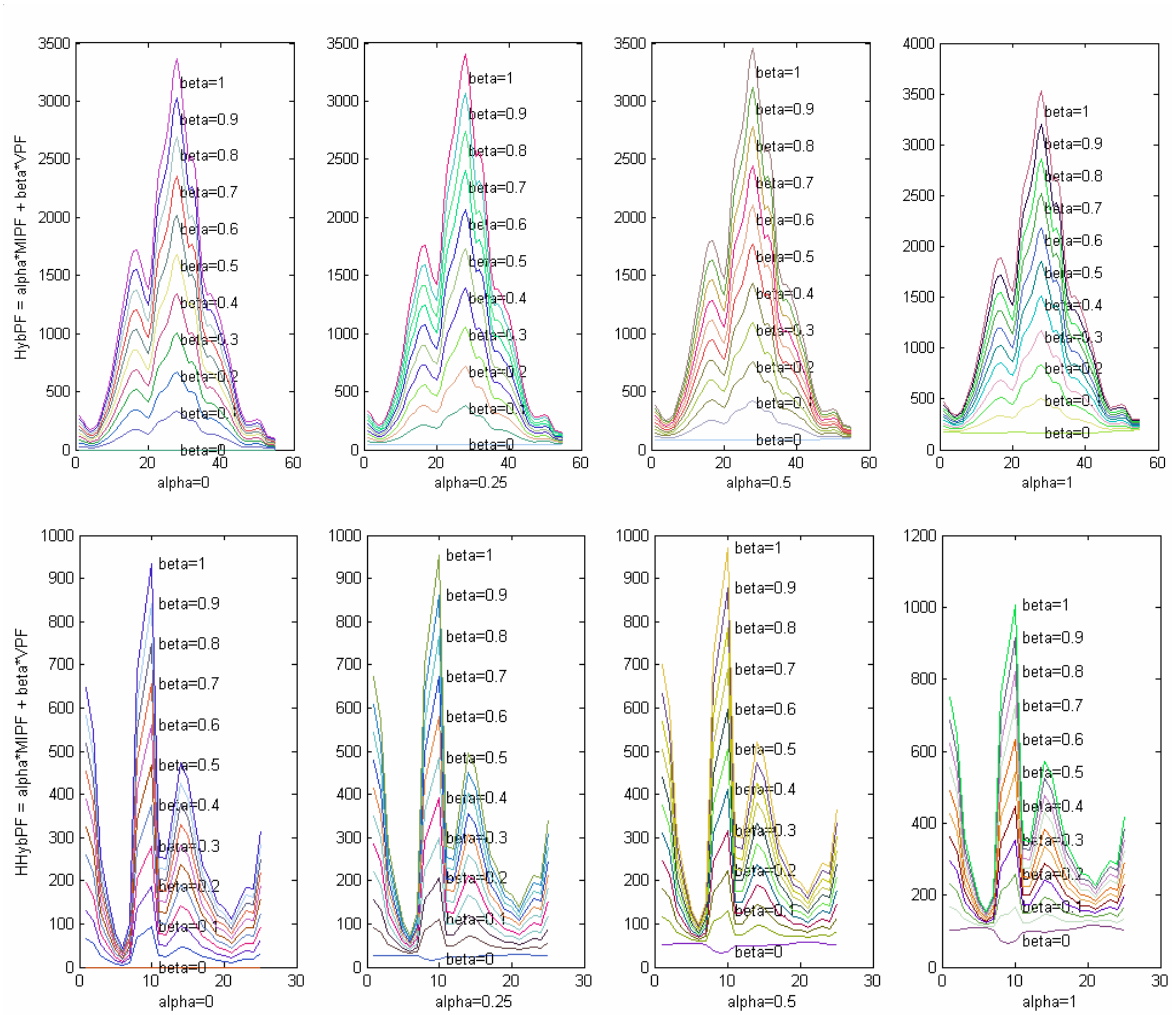


Figure 41: Mouth image: (Top) Vertical projection (Bottom) Horizontal projection. For beta=0 the variational information cannot be captured. Clear variational differences can be noticed for alpha>0.3.

To actually see the relative contribution of IPF and VPF in the general function we define GFP as follows:

$$GPF_v(x) = (1-\alpha)\cdot MIPF_v(x) + \alpha \cdot VPF_v(x)\ ,$$
$$0 \le \alpha \le 1 \qquad \text{(Eq. 7.10)}$$
$$GPF_h(y) = (1-\alpha)\cdot MIPF_h(y) + \alpha \cdot VPF_h(y)\ ,$$

By this definition IPF and VPF are actually special cases of GPF, where α is 0 or 1, respectively. The following figures (Figure 42, Figure 43, Figure 44 and Figure 45) show this relative relation of IPF and VPF. The results are obtained from an eye and mouth image as the variational information of both features differs. Same things as before can be noticed, as in this case α is replaced by $(1-\alpha)$ and $\beta = \alpha$.

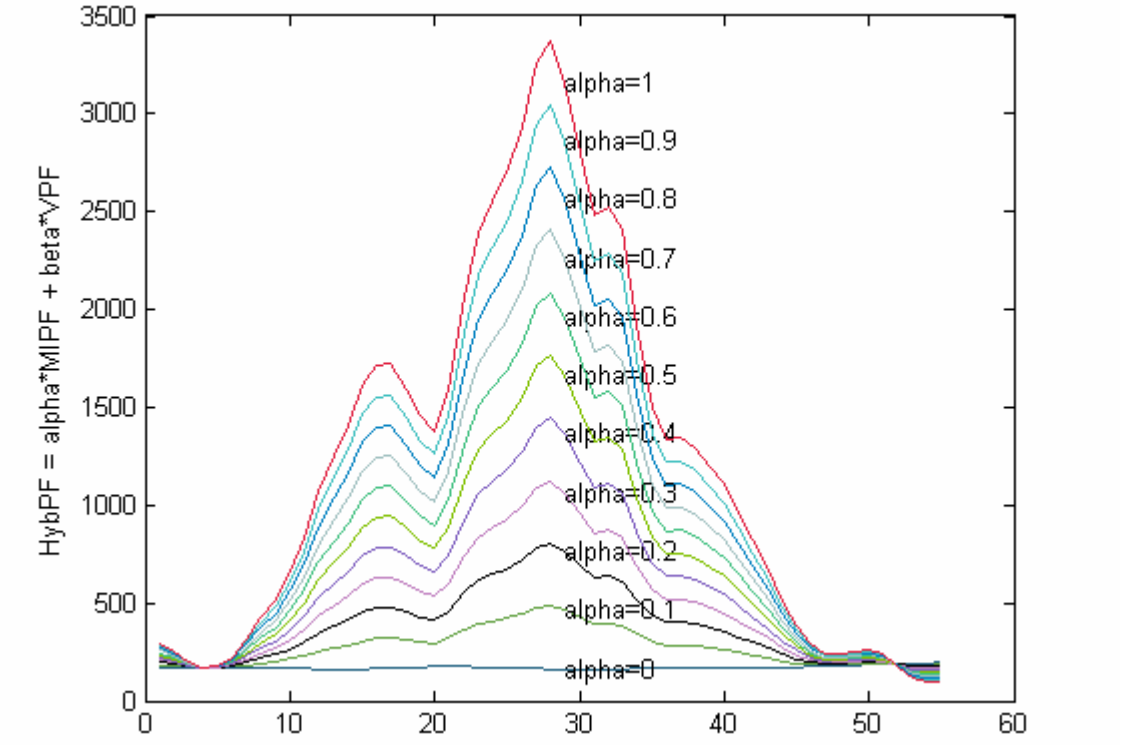


Figure 42: Vertical projection of an eye image using – +

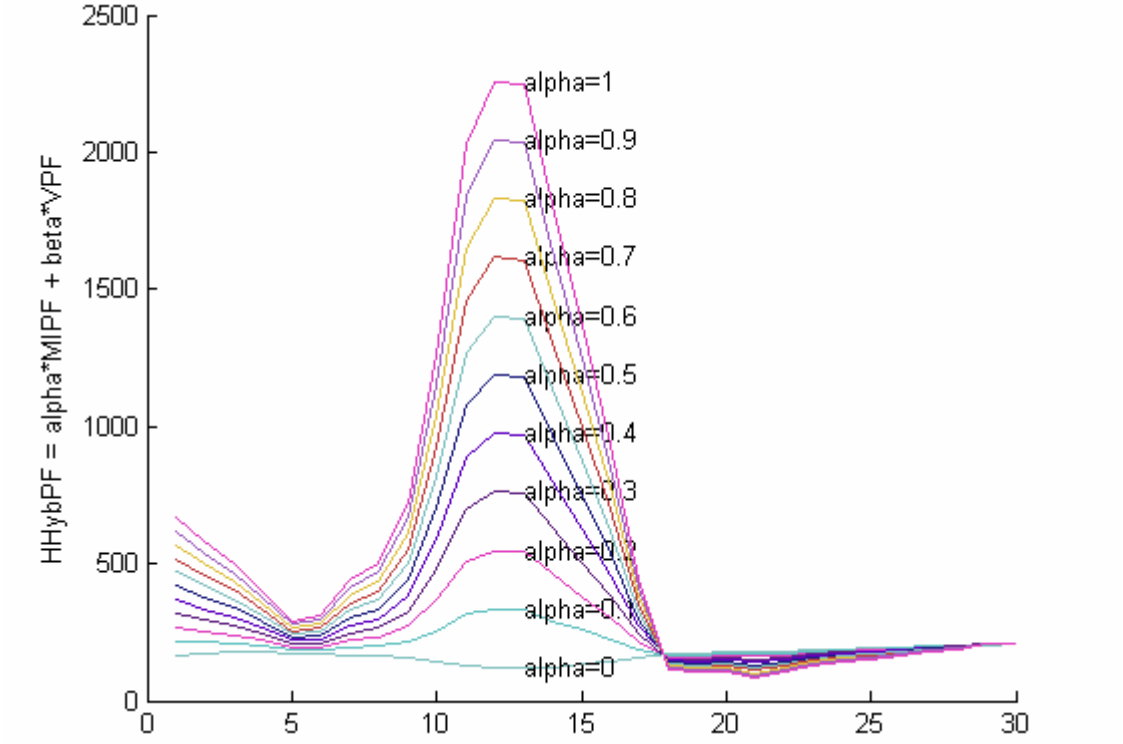


Figure 43: Horizontal projection of an eye image using – +

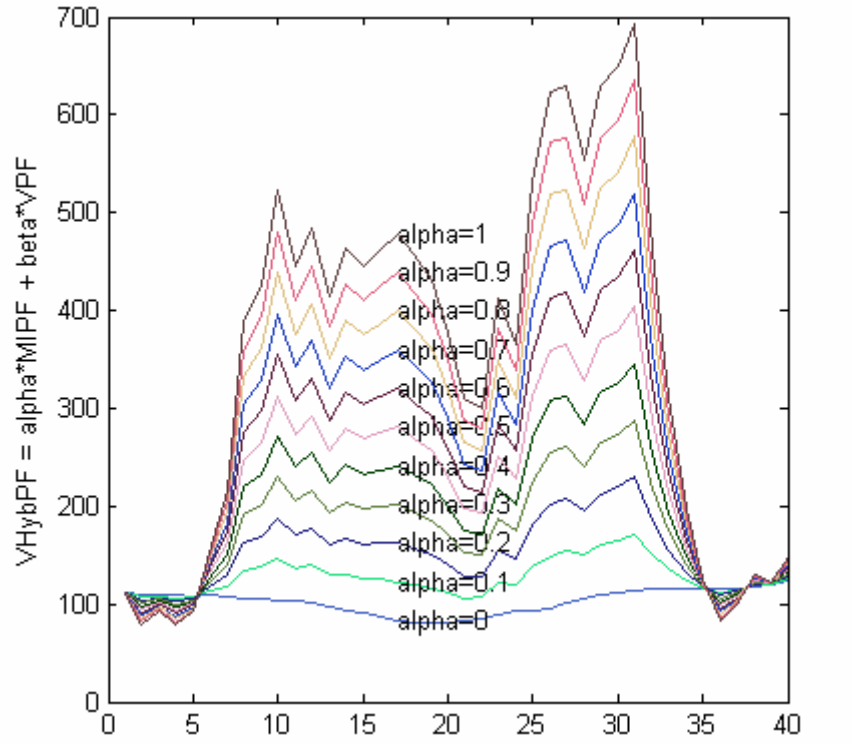


Figure 44: Vertical projection of a mouth image using – +

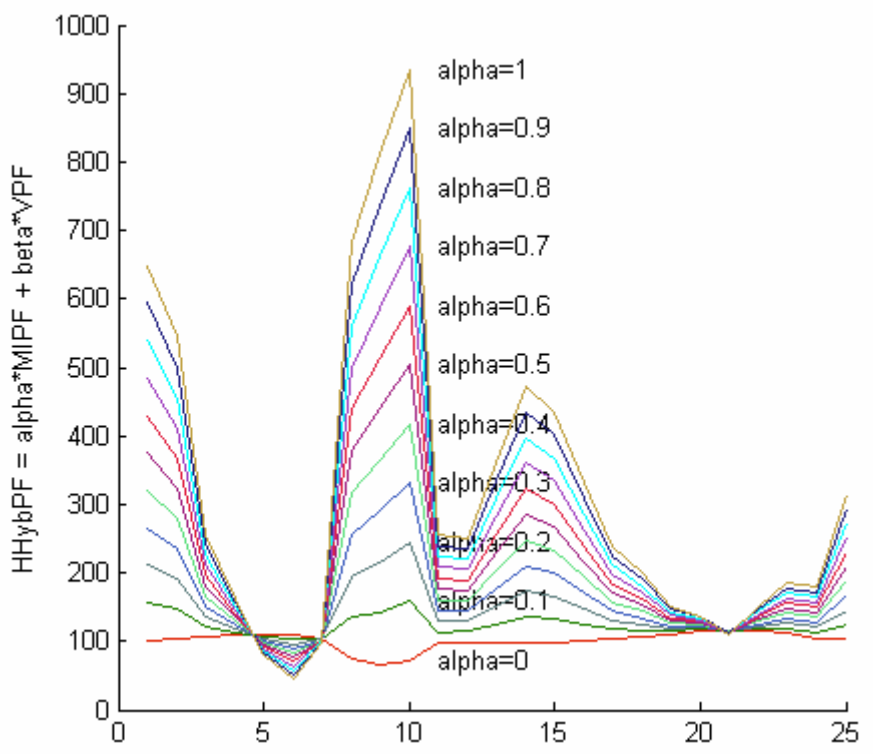


Figure 45: Horizontal projection of a mouth image using – +

The optimal value of α for mouth and eyes are derived from some test results on mouth and eye images (see Table 29 and Table 30). The optimal values are highlighted in the table. The values are selected based on their accuracy and stability. The threshold set for the images at these values are derived from the image histograms and is about the same for all test images unlike for other values of α. In our final implementation we choose $\alpha = 0.6$ as the optimal value for the hybrid projection.

Table 29: Test results at finding the optimal value for the hybrid projection for mouth.

MOUTH					
Alpha (α)	LB	RB	UB	BB	ODR
0.2	100 %	100 %	75 %	50 %	50-75 %
0.3	100 %	100 %	75 %	50 %	50-75 %
0.4	100 %	100 %	75 %	75 %	50-75 %
0.5	100 %	100 %	90 %	90 %	75-100 %
0.6	100 %	100 %	95 %	95 %	75-100 %
0.7	100 %	100 %	95 %	95 %	75-100 %
0.8	100 %	100 %	90 %	90 %	75-100 %
0.9	100 %	100 %	90 %	75 %	50-100 %
1.0	100 %	100 %	90 %	75 %	50-100 %

LB = Left Boundary
RB = Right Boundary

UB = Upper Boundary
BB = Bottom Boundary
ODR = Overall Detection Rate

Table 30: Test results at finding the optimal value for the hybrid projection for eyes.

EYES					
Alpha (α)	LB	RB	UB	BB	ODR
0.3	100 %	90 %	75 %	50 %	50-75 %
0.4	100 %	90 %	75 %	75 %	50-75 %
0.5	100 %	90 %	90 %	90 %	75-100 %
0.6	100 %	90 %	100 %	100 %	90-100 %
0.7	100 %	90 %	100 %	100 %	90-100 %
0.8	90 %	90 %	100 %	100 %	90-100 %
0.9	90 %	90 %	100 %	100 %	90-100 %
1.0	90 %	90 %	100 %	100 %	90-100 %

LB = Left Boundary
RB = Right Boundary
UB = Upper Boundary
BB = Bottom Boundary
ODR = Overall Detection Rate

8.5 Facial feature extraction

Since our purpose of using hybrid projection function is to extract FCPs which are located on the facial features, our system need to know where these facial features are positioned. Therefore, to apply hybrid projection successfully, proper images of the facial features need to be provided. In order to extract the exact position of a facial feature, we need to apply our WUXTRAP algorithm with proper images of the eyes, eye brows, nose and mouth to train our classifier to recognize them. Once detected, they can be passed to the hybrid projection module for further processing. Following are the specification and result for the training procedure of the left eye, right eye and mouth.

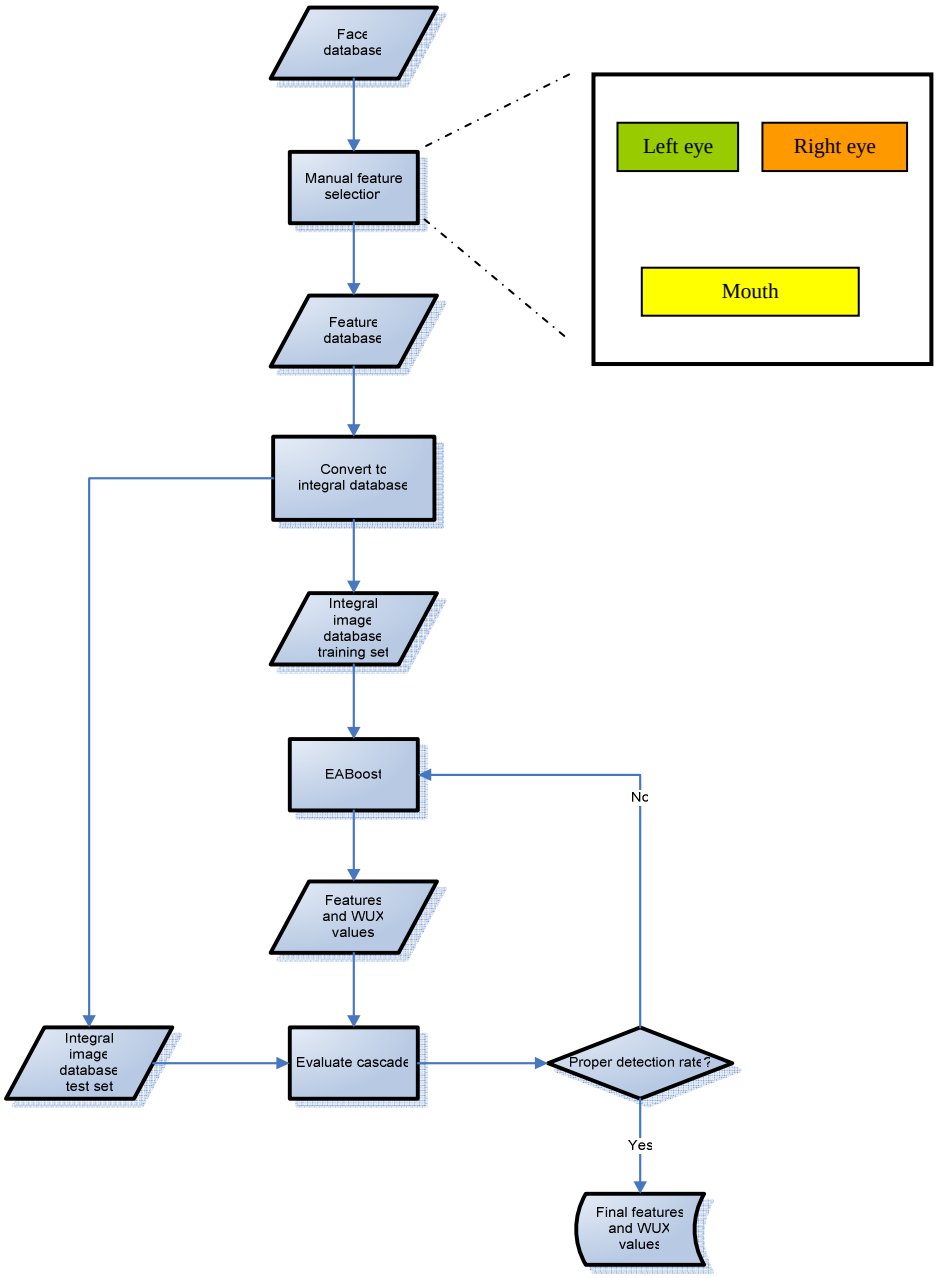


Figure 46: General scheme of WUXTRAP for feature training.

Left eye and right eye

The two databases, containing respectively images of the left eye and right eye are created from the BioID face database. In total, we used a set of 1000 positive samples of the left eye and 1000 negative samples of the left eye. This set is split into two set of equal sizes: one for training and one for testing. The same yields for the right eye dataset. For more details, see Table 31 and Table 32. The training parameters for EABoost are the same as for faces (see Table 14). Note that the set of Haar-like features to be searched and trained is not the same as that of faces. Since an

eye image is 15x25 pixels, we need to apply the Haar-like features on every possible scale and in every possible position in every eye sample. That results in a set of 69031 Haar-like features to be searched. The result that we get after training is shown in Table 33.

Table 31: Dataset specification for left eye.

Dataset parameters	Values and description
Database source	BioID
Sample size (h x w)	15x25 pixels
Number of classes	2
Class 0	Non-left eye
Class 1	Left eye
Number of samples (0/1)	500/500

Table 32: Dataset specification for the right eye.

Dataset parameters	Values and description
Database source	BioID
Sample size (h x w)	15x25 pixels
Number of classes	2
Class 0	Non-right eye
Class 1	Right eye
Number of samples (0/1)	500/500

Table 33: Evaluation test result for detecting the whole eye.

Type	TPR	FPR
Left eye	0.906	0.05
Right eye	0.912	0.049

TPR = True Positive Rate
FPR = False Positive Rate

Mouth

The mouth database we used to train RVM with is also extracted from the BioID face database. We used a subset of 500 positive and 500 negative samples for training and a distinct subset of the same size for testing. The set is specified in the following table. The training parameters for EABoost are also the same as for faces. The Haar-like features' set in a 20x40 pixels image is 312260. The training results are shown in Table 34.

Table 34: Dataset specification for the mouth.

Dataset parameters	Values and description
Database source	BioID
Sample size	20x40 pixels
Number of classes	2
Class 0	Non-mouth
Class 1	Mouth
Number of samples (0/1)	500/500

Table 35: Evaluation test result for detecting the mouth.

Type	TPR	FPR
Mouth	0.852	0.024

Part IV

System Implementation

9

Analysis and System Design

In this chapter, we will describe the implemented system FLEX that is able to perform the task of face detection and FCPs extraction. All models and algorithms that are used in FLEX are already described and discussed in the previous chapters. FLEX makes use of the WUX values retrieved with WUXTRAP to carry out its tasks. The result of the requirement analysis definition and the functional design will be given in section 9.1. Section 9.2 will be devoted to the system models and implementation details. The user interface and some screen mock-ups will be presented in section 9.3.

9.1 Requirement analysis

FLEX consists of two modules: a face detection module and a FCP extraction module. The tasks of the two modules are as their name suggests.

9.1.1 Purpose of the system

The main purpose of FLEX is the extraction of FCPs from digital images. The input for FLEX is a digital image selected by the user. First, FLEX will automatically scan the image for faces. If present and their resolution are bigger than 64 x 64 pixels, it will extract for each face the FCPs. Within the framework of FED, these FCPs can be directly passed to the QPM (see section 2.1) which will query the FED database for a facial expression. The result of FLEX will be a vector containing FCP information.

9.1.2 Scope of the system

FLEX can handle one input image at a time. The input images can either be colour or gray-scale images. From the input image, only faces with a resolution equal to or greater than 64 x 64 will be passed to the FCP extraction module. As mentioned before, this final image size has been chosen because of the trade-off between computational cost and minimum resolution in order to

guarantee that information about eyes, nose and mouth is not lost in too small image versions. The input images should be one of the following formats: BMP, JPEG, JPEG2000, PNG, PNM, Raw, TIFF and WBMP. FLEX can not yet handle other multimedia formats like video input and is also not required within the FED framework at the moment.

9.1.3 Functional requirements

The main purpose of FLEX is extracting FCPs from the input image. The user is able to select a digital image from the file system of the computer FLEX is running on. FLEX determines the FCPs of the faces that are present in the image. FLEX complete its task with showing the FCPs that it has found.

9.1.4 Non-functional requirements

Adaptability: We attempt to build the module as compatible as possible to the FED so that minimal adjustment to the existing FED code is needed for the integration of FLEX into FED.

Documentation: FLEX is well documented for future studies.

Error handling: All errors are handled correctly by the program. In case of errors, comprehensible messages will be displayed.

Extendibility: It is possible to make modifications to the FLEX system in the future. Additional functionalities such as processing video input can be easily added since we are using Object Oriented Design.

Performance characteristics: FLEX should be able to detect 80% of the characteristic points in a face.

Real-time: Speed is of critical importance. Especially when FLEX is going to be integrated into the FED website. At the moment, the speed of FLEX depends on factors like: size of the input image, the processor on which FLEX is running (compared to a Celeron 1.5 GHZ, 512 MB machine, FLEX is running about twice as fast on an AMD Athlon XP, 1.8 GHZ, 512 MB machine), and so on.

9.1.5 Pseudo requirements

FLEX is initially designed as a module to be integrated into the FED framework. FED is a web application written in Java/Java2, therefore FLEX is also written in Java/Java2. FLEX is developed in a Windows environment and does not make any use of a database.

9.2 System models

9.2.1 Use cases

Figure 47 shows the use case diagram of FLEX. It describes the behaviour of FLEX as seen from the user's point of view. The only actor in this use case diagram is the User of FLEX. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Following the figure is the use case description of the use cases.

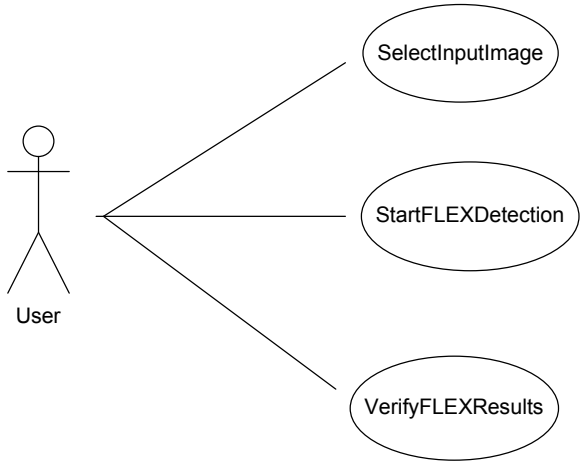


Figure 47: Use case diagram for FLEX.

Use case name	*SelectInputImage*
Participating actor	*User*
Entry condition	1. FLEX is started up
Flow of events	2. The *user* opens a getFileDialog to select the input image.
Exit condition	3. FLEX confirms the *user* by a message and by showing the selected image.

Figure 48: The SelectInputImage use case.

Use case name	*StartFLEXDetection*
Participating actor	*User*
Entry condition	1. The *user* has selected an input image.
Flow of events	2. The *user* initiates the detection process by clicking on the Detect button. 3. The face detection module of FLEX reads in the input image and scans for faces at different resolutions. 4. For each detected face with a resolution of at least 64 x 64, the FCP extraction module scans for all the predefined FCPs.
Exit condition	5. FLEX shows its results by displaying the detected faces with the FCPs marked.

Figure 49: The StartFLEXDetection use case.

Use case name	*VerifyFLEXResults*
Participating actor	*User*
Entry condition	1. The results of the FCP extraction procedure is displayed on the screen.
Flow of events	2. The *user* examines each of the shown faces with the FCPs.
Exit condition	3. The user accepts the results by clicking on the OK button or declines it by clicking on the Decline button.

Figure 50: The VerifyFLEXResults use case.

9.2.2 Class diagram

Figure 51 gives the class model diagram that describes the structure of the FLEX system in terms of classes and objects. Classes are abstractions that specify the attributes and behaviour of a set of objects. Objects are entities that encapsulate state and behaviour. Each object has an identity: It can be referred individually and is distinguishable from other objects. Each class will be described individually.

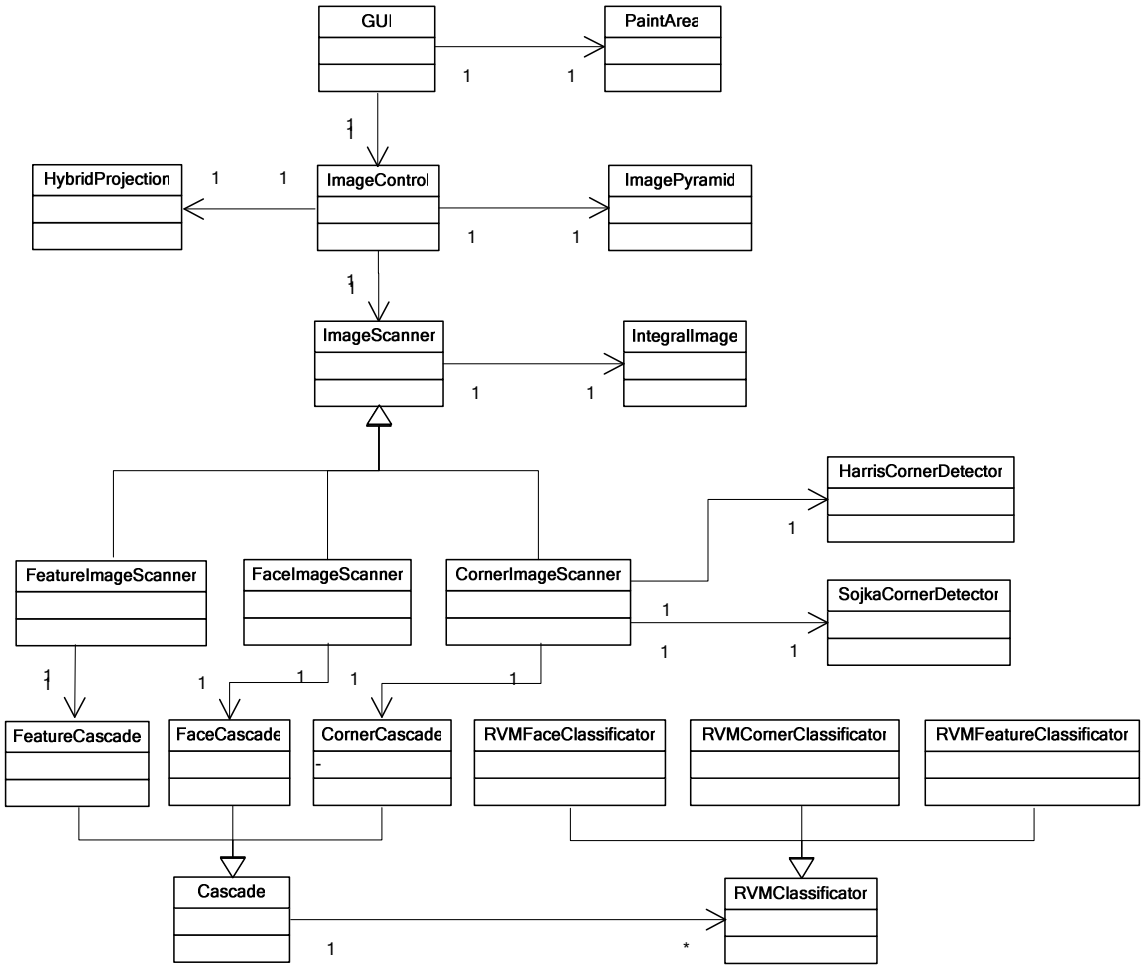


Figure 51: Class diagram of FLEX.

GUI Class

The Graphical User Interface class provides the user the facility to select an input image to be scanned, to start the scanning procedure and to verify the results. The results with graphical output will be handled by PaintArea.

ImageControl Class

The ImageControl class is the central unit in FLEX. All communication and dataflow between the user, face detection module and the FCP extraction module occur via this class. It controls the instantiation of the FaceImageScanner object, CornerImageScanner object and ImagePyramid object. The methods scanLayers4Faces and scanFace4Corners will be invoked by the user via the graphical user interface. If the input image is a 24-bit color image, ImageControl will automatically convert it to an 8-bit gray scale representation for further processing.

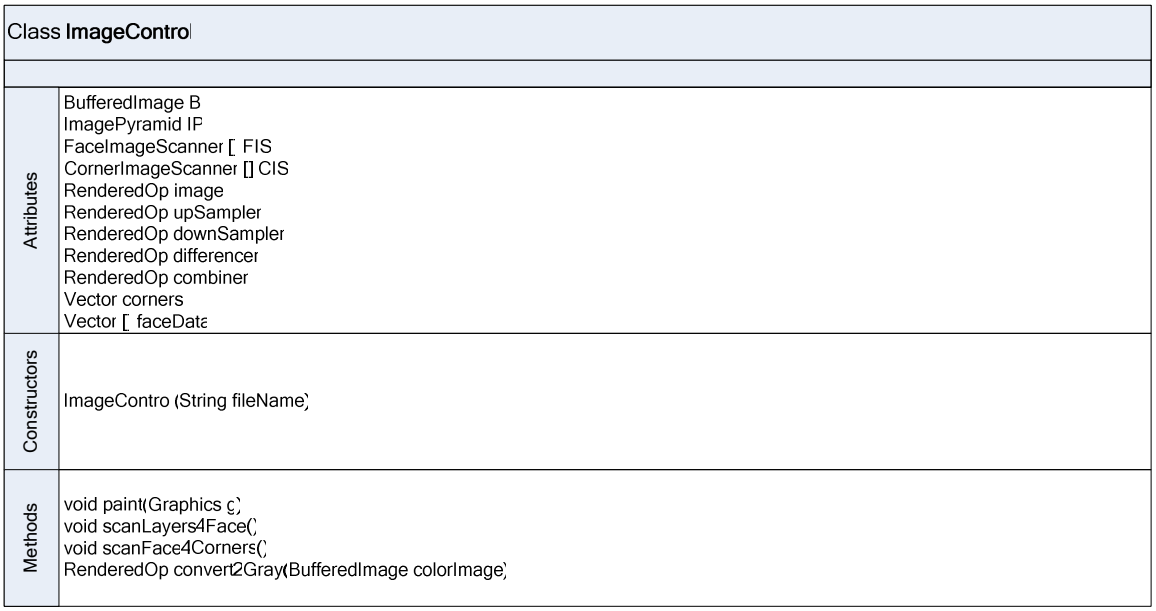


Figure 52: Description of the ImageControl class.

ImagePyramid Class

This class is included in the Java Advanced Imaging (JAI) package that is provided by Sun Microsystems. It represents a multi-resolution image pyramid: i.e. a collection of layers of different resolution of the input image. More information about this class can be found on: http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/javax/media/jai/ImagePyramid.html

ImageScanner Class

The ImageScanner class is an abstract class that provides the interface for the FaceImageScanner class and the CornerImageScanner class.

IntegralImage Class

This class represents the integral image representation of an image. With the method convert2IntImage, a normal image can be converted into its integral representation.

FaceImageScanner Class

An instance of FaceImageScanner takes one layer of the image pyramid into account for face detection. It does so by first scanning the input image with a 24 x 24 scanning window. Then it will convert each subimage into the integral image representation. After that, the subimages will be passed to the FaceCascade which decides whether or not a face is present. Note that in the final implementation of FLEX, this class also controls the scanning process. Instead of scanning

every layer in the image pyramid this class simply scales the classifier by scaling the Haar features.

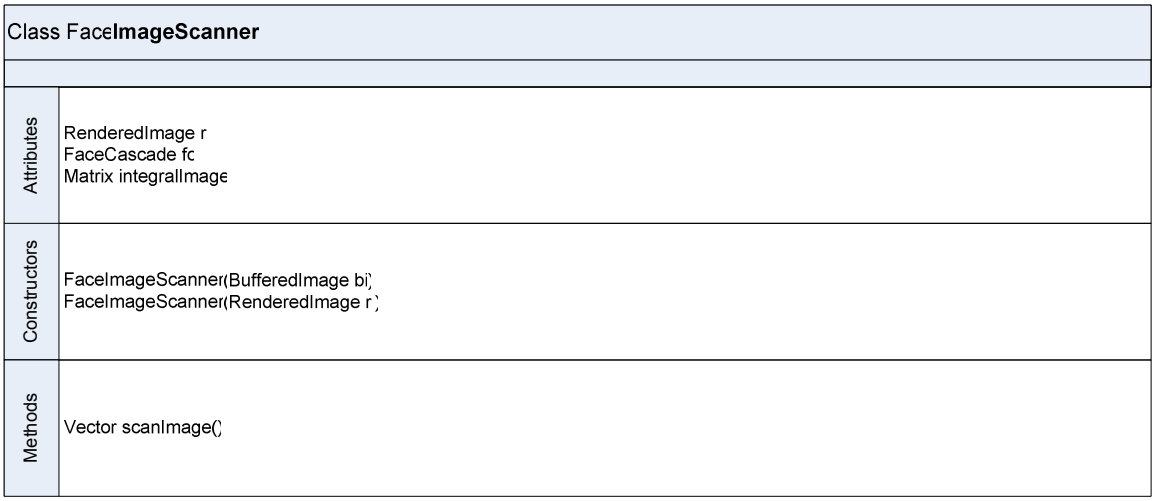


Figure 53: Description of the FaceImageScanner class.

FaceCascade Class

The FaceCascade class is in fact the component that decides whether or not a given image contains a face. The FaceCascade consist of different layers each with a different number of classifiers. Input for the cascade is a collection of all the subimages from FaceImageScanner. They are first passed through the first layer in which all subimages will be classified as faces or non faces. The negative results will be discarded. The remained positive subimages will trigger the evaluation of the next classifier. The same process is performed in every layer. The subimages that reach and pass the last layer are true faces.

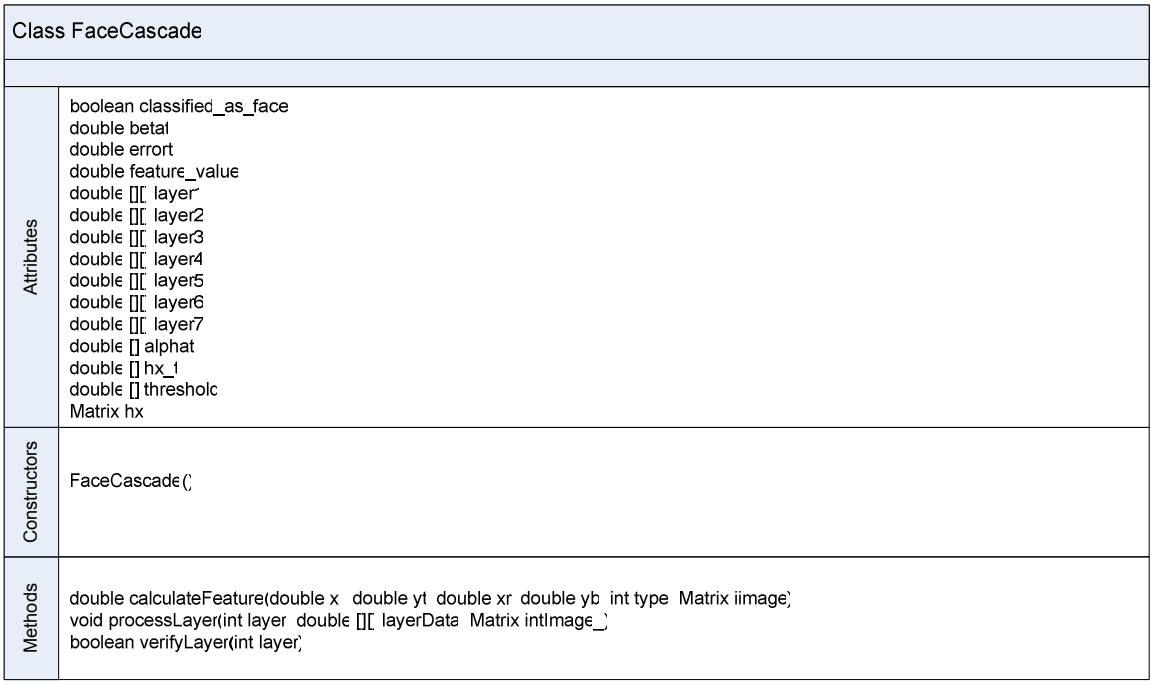


Figure 54: Description of the FaceCascade class.

FeatureImageScanner Class

This class process the image for facial features which can be eyes, eye brows, nose or mouth. Therefore, the input image should be a face image. In this case, the face image is an image detected by the face detector and has a size of 64 by 64 pixels. FeatureImageScanner scans for its feature in a specific region. It will convert each subimage into an integral image representation. After that, the subimages will be passed to the FeatureCascade which decides whether or not the wanted feature is present.

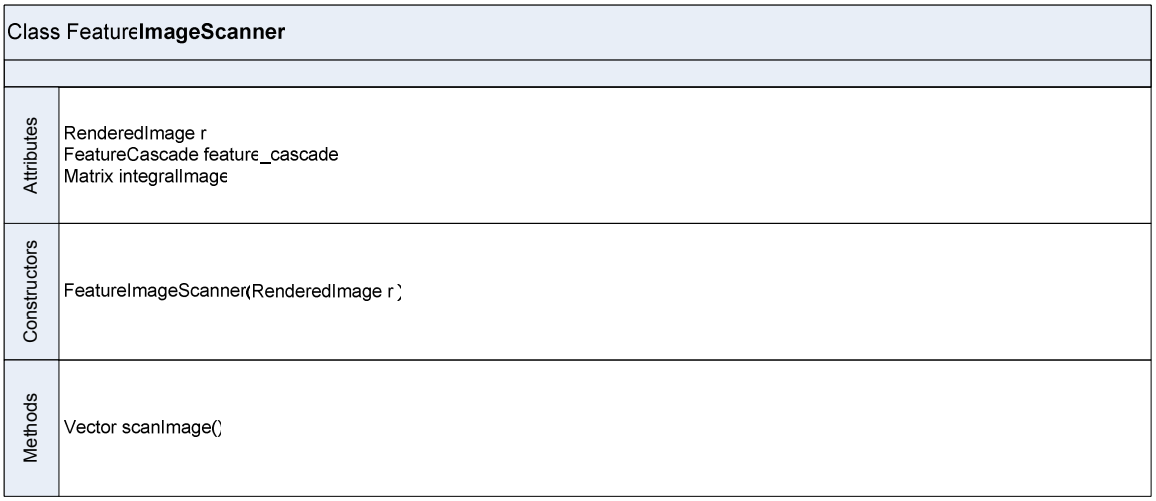


Figure 55: Description of the FeatureImageScanner class.

FeatureCascade Class

Practically the FeatureCascade class performs the same task as FaceCascade. Its structure is also as that of FaceCascade. It decides whether or not a given face image contains the wanted feature. The subimage that passes through all layers of the cascade can be considered as a true positive sample.

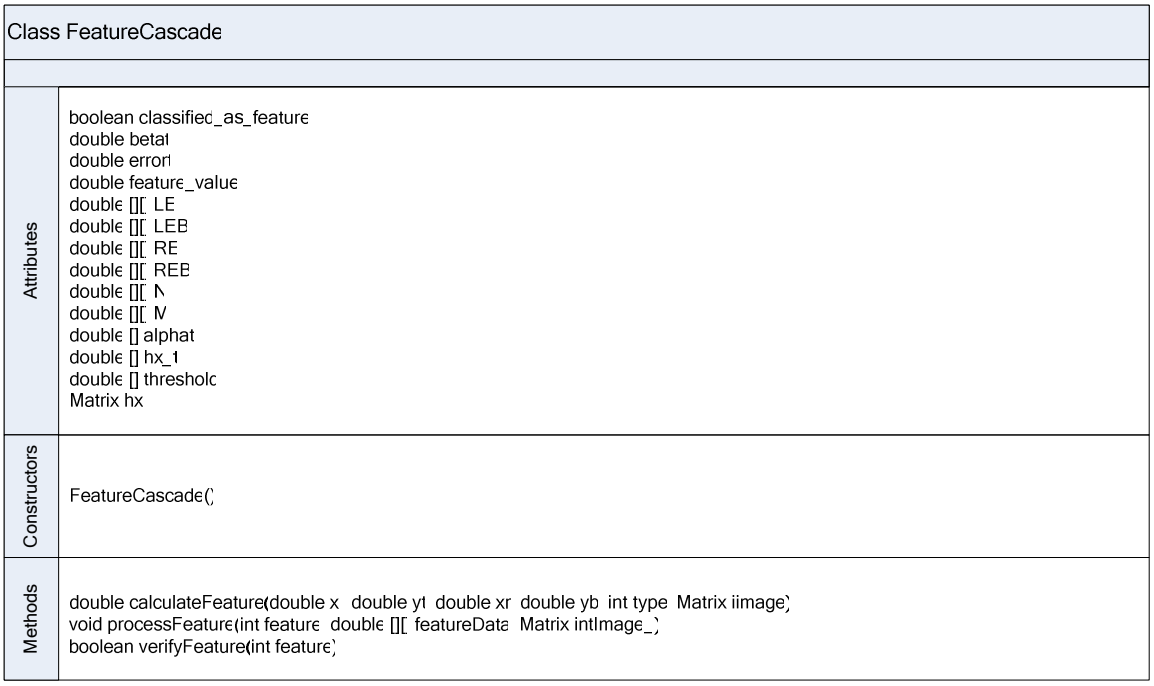


Figure 56: Description of the FeatureCascade class.

HybridProjection Class

The result from face detection and FeatureImageScanner is a vector containing positions of facial features. HybridProjection uses these exact locations to extract the features and corresponding FCPs. This can only be done if the extracted feature does not contain noise in the form of parts of other features.

Class HybridProjectior	
Attributes	Matrix image
Constructors	HybridProjection (Matrix image)
Methods	int verticalGradient (Matrix image int column int bound int bound2) int horizontalGradient (Matrix image int row int bound int bound2) Matrix verticalVarianceProjectionFunctior (Matrix image) Matrix horizontalVarianceProjectionFunctior (Matrix image) Matrix verticalMeanIntegralProjectionFunction(Matrix image) Matrix horizontallMeanIntegralProjectionFunctior (Matrix image) Vector checkFeature() Vector find (Vector v double threshold)

Figure 57: Description of the HybridProjection class.

CornerImageScanner Class

This class process the image, which in this case is a face image, for FCPs. It ensures that corner detection is employed by instantiating the HarrisCornerDetector and SojkaCornerDetector classes. As a result, the number of possible windows to scan for FCPs will be reduced. Classification of the candidate windows is done by the CornerCascade which in turn invoke the RVMClassificator. Variance integral projection too will be carried out on eyes, mouth and nose to increase the accuracy of the detected FCPs.

Class Corner**ImageScanner**	
Attributes	RenderedImage r CornerCascade cc Matrix integralImage
Constructors	CornerImageScanner(RenderedImage r)
Methods	Vector scanImage()

Figure 58: Description of the CornerImageScanner class.

HarrisCornerDetector Class

The given input image, which must be a gray-scale image, is scanned for corners. This class is invoked by the CornerImageScanner class to reduce the number of scanning windows. The parameters upon which this class is called are also controlled by the class that invokes it.

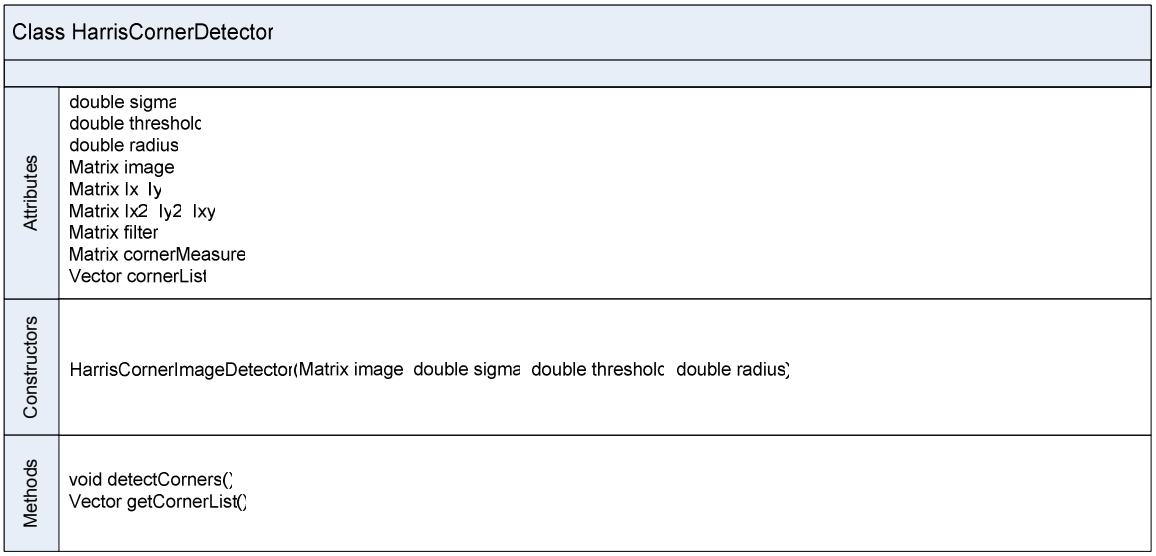


Figure 59: Description of the HarrisCornerDetector class.

SojkaCornerDetector Class

The given input image, which must be a gray-scale image, is scanned for corners. Detection of corners can actually be done in colour and/or gray-scale images. This class is invoked by the CornerImageScanner class to reduce the number of scanning windows. The parameters upon which this class is called are also controlled by the class that invokes it.

Class SojkaCornerDetector	
Attributes	float noiseGradSizeThres; float meanGradSizeThresh; float inertiaRadiusThresh; float halfGradSizeThresh; float tenthGradSizeThresh; float angleThresh; float halfAngleThresh; float apparenceThresh; float sigmaD, sigmaR; float [] gSizes; float [] gDirs; float [] corrDirs; float [] corrs; float [] appars; float [] SNRs; float [] Psgs; float [] Wrs; float [] PsgWrs; float [] dirXs, diffDirXs; float [] intCoefs; float [] pdTable; int xImageSize, yImageSize; int numImagePixs; int halfPsgMaskSize; int numImportantPixs; int numPsgMaskPixs; int halfExtMaskSize; int options; int [] neighbMap; int [] xs; int [] flagsProcess; int [] influenceMap; int [] influences; int [] importantPixs; int [] tobeProcessed; Vector corners;
Constructors	SojkaCornerDetector();
Methods	int [] detectCorners(float [] image_p, int xImageSize_p, int yImageSize_p, int halfPsgMaskSize_p, float angleThresh_p, float noiseGradSizeThresh_p, float apparenceThresh_p, float meanGradSizeThresh_p, float inertiaRadiusThresh_p, float sigmaD_p, float sigmaR_p, int halfExtMaskSize_p, int options_p, int [] cornerList, int cornerListLen);

Figure 60: Description of the SojkaCornerDetector class.

CornerCascade Class

The CornerCascade class is the component which decides whether or not a given image contains a FCP. It does so by invoking the RVMCornerClassificator with the right parameters. One of the parameters is the feature value, which is calculated by simply look up the values in the integral

representation matrix of the image. Another parameter is the type of the FCP to be classified like inner left eye corner.

Class CornerCascade	
Attributes	boolean classified_as_corner double betat double errorl double feature_value double [][LERC double [][LELC double [][RELC double [][RERC double [][NLC double [][NRC double [][MLC double [][MRC double [] alphat double [] hx_1 double [] threshold Matrix hx
Constructors	CornerCascade()
Methods	double calculateFeature(double x double yt double xr double yb int type Matrix iimage) void processCorner(int corner double [][cornerData Matrix intImage_) boolean verifyCorner(int corner)

Figure 61: Description of the CornerCascade class.

RVMClassificator Class

This class is an abstract class that provides the interface for the classification of the input. Actual classification is done by the RVMFaceClassificator class and the RVMCornerClassificator class depending on the classification that needs to be made.

RVMFaceClassificator Class

This class represents the actual RVM classification model for faces. The implementation of the RVM classification model is to differentiate faces from non-faces. In other words the classification is fixed by a two-class classification. This class is invoked by the FaceCascade class which also has to pass the parameters depending on the layer in which the RVM is located.

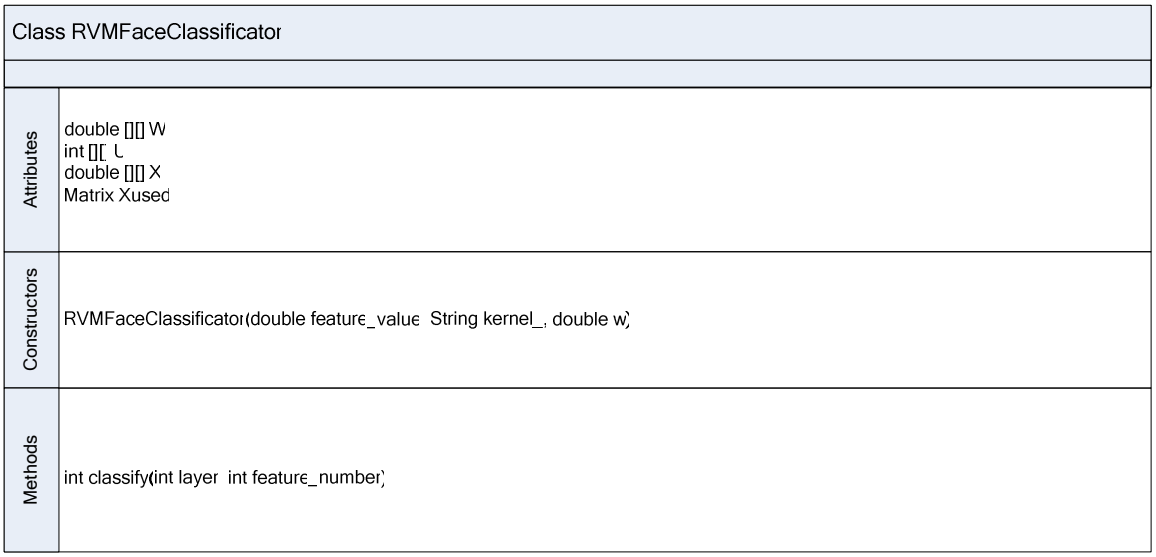


Figure 62: Description of the RVMFaceClassificator class.

RVMCornerClassificator Class

This class represents the actual RVM classification model for FCPs. The implementation of the RVM classification model is to differentiate FCPs from non-FCPs. In other words the classification is determined by a two-class classification. This class is invoked by the CornerCascade class which also has to pass the parameter of which FCP to classify.

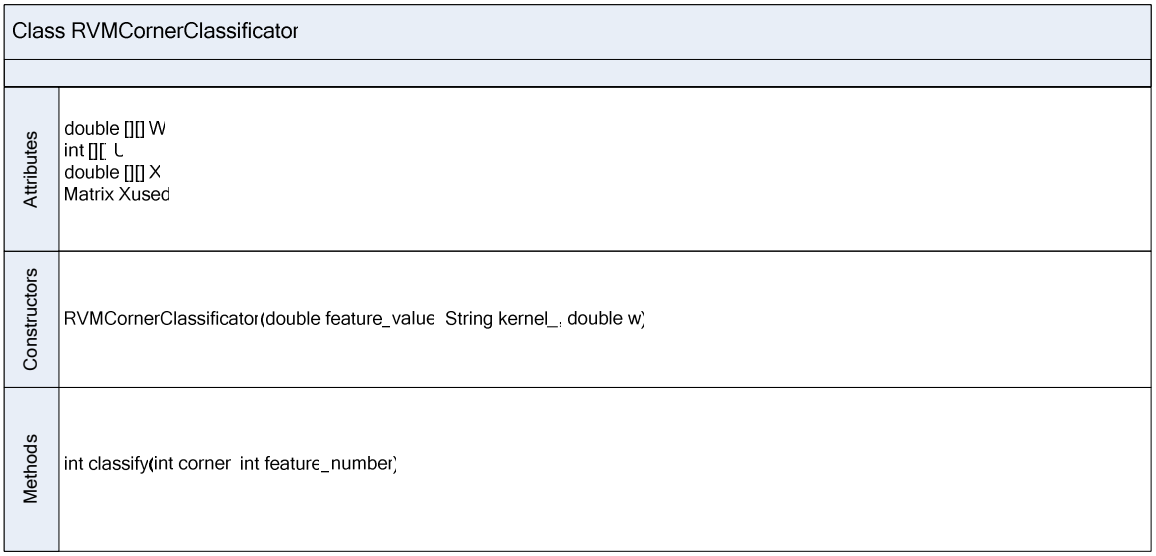


Figure 63: Description of the RVMCornerClassificator class.

RVMFeatureClassificator Class

This class represents the actual RVM classification model for features. The implementation of the RVM classification model is to differentiate features from non-features. In other words the

classification is fixed by a two-class classification. This class is invoked by the FeatureCascade class which also has to pass the parameters depending on the layer in which the RVM is located.

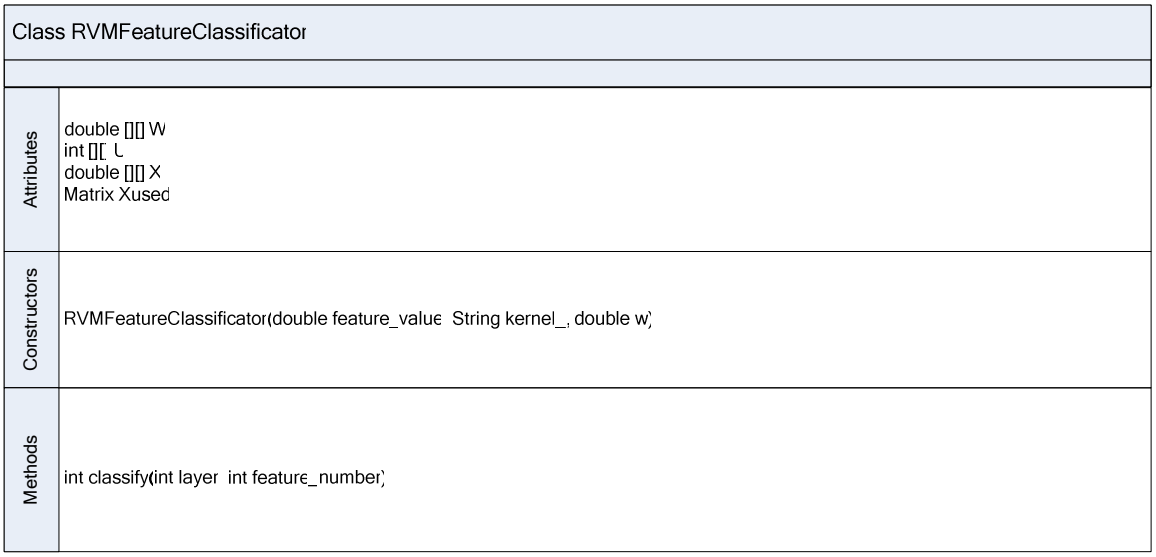


Figure 64: Description of the RVMFeatureClassificator class.

9.2.3 Sequence diagrams

Figure 65, Figure 66 and Figure 67 present the sequence diagrams of FLEX. They formalize the behaviour of FLEX and visualize the communication among the objects. Figure 65 shows the interaction between the objects of the face detection module of FLEX. The ImageControl object invokes and initializes the other objects. When an input image is submitted by the user, an ImagePyramid of the input will be constructed. The user can then give the signal to start the face detection procedure by invoking the scanImg method of FaceImageScanner. In the end of this sequence diagram, the ImageControl object obtains face information result. With this information, ImageControl invokes the scanImg of FeatureImageScanner to extract the facial features. This is shown in the second diagram. The result will be a vector with exact positions of the features. ImageControl can then use this information to invoke the checkFeature method of HybridProjection to extract the FCPs from the feature. In the third diagram the process of corner classification is shown. As shown, the process is analogous to that of face detection and feature extraction. The exception is that it contains corner detectors that reduce the number of subimages to be classified.

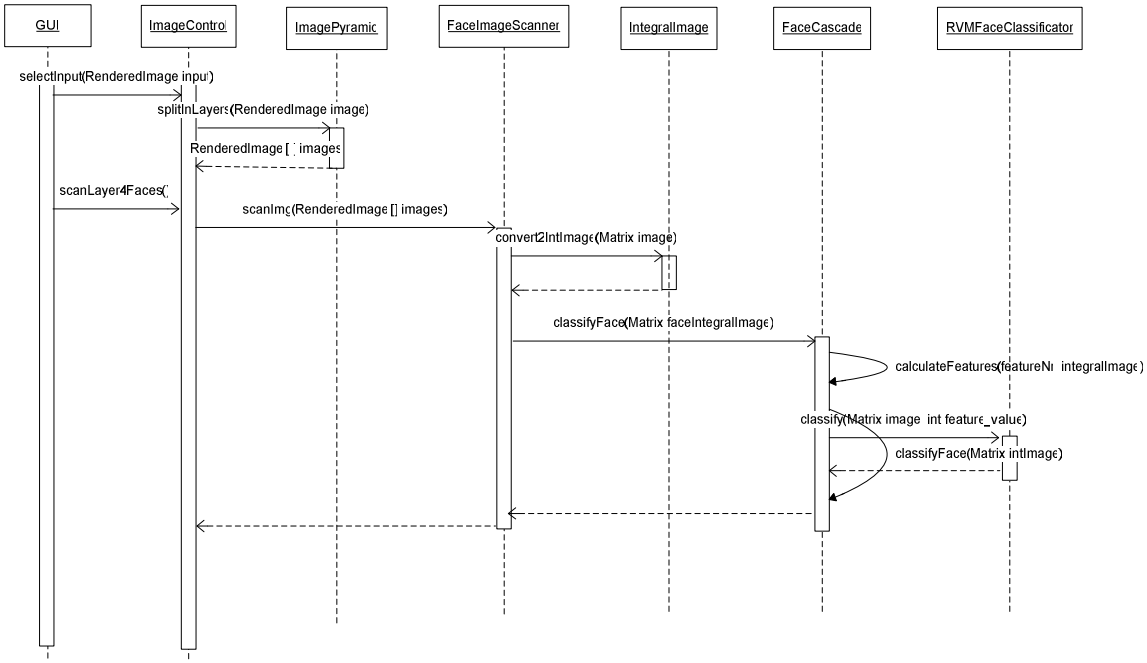


Figure 65: Sequence diagram of FLEX. Part 1: face detection.

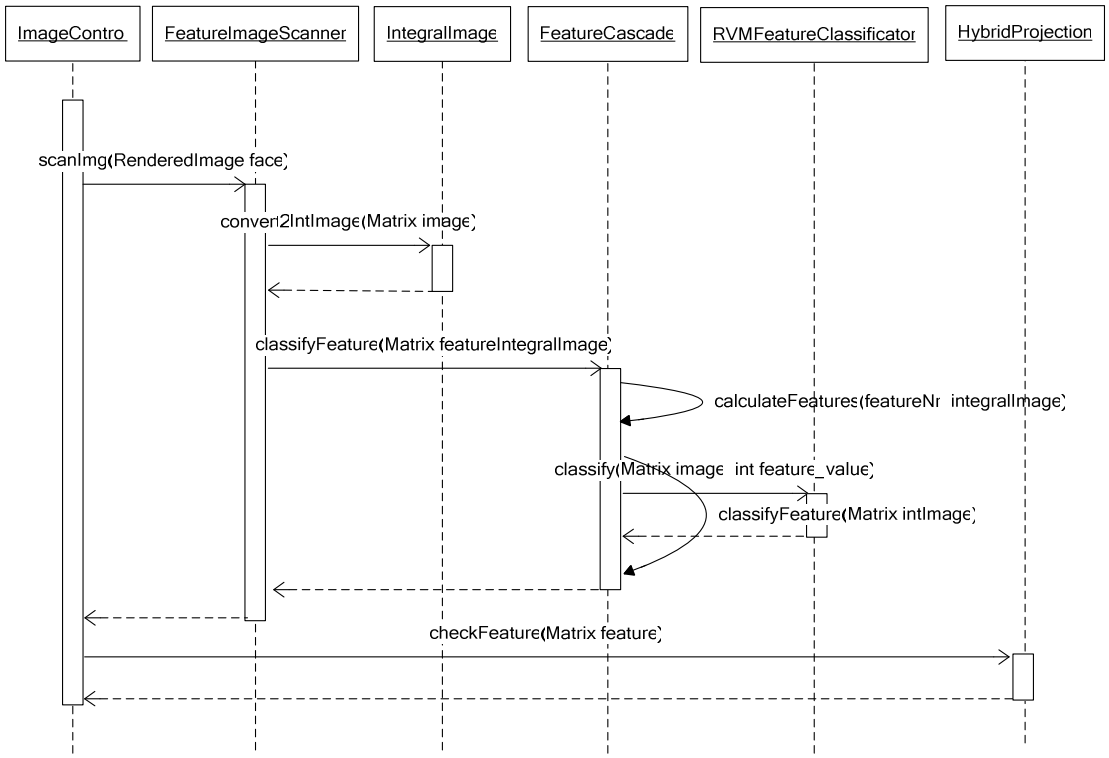


Figure 66: Sequence diagram of FLEX. Part 2: feature detection.

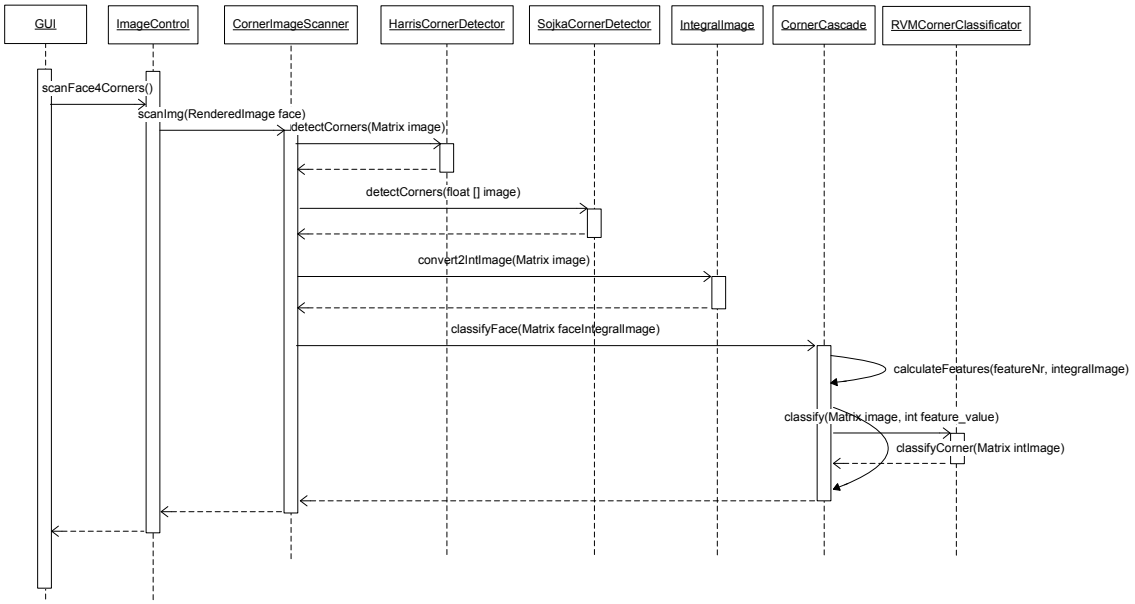


Figure 67: Sequence diagram of FLEX. Part 3: FCP extraction.

9.3 User interface

This section includes some screen shots of the FLEX user interface.

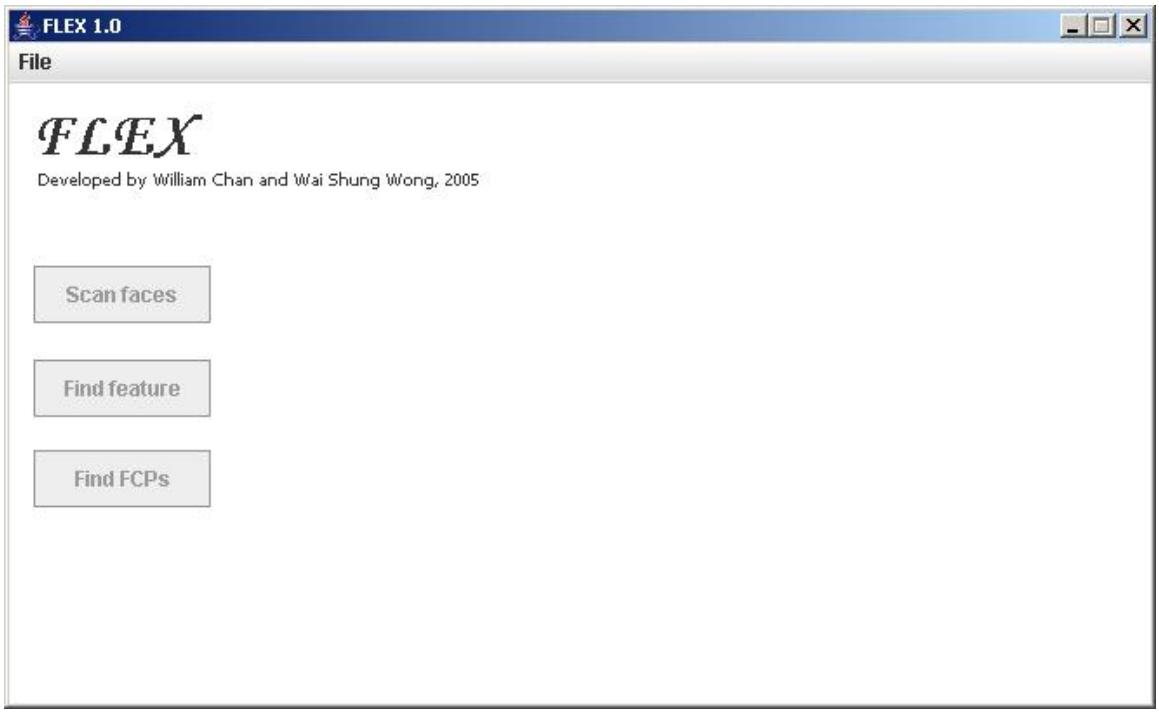


Figure 68: GUI of FLEX at start-up.

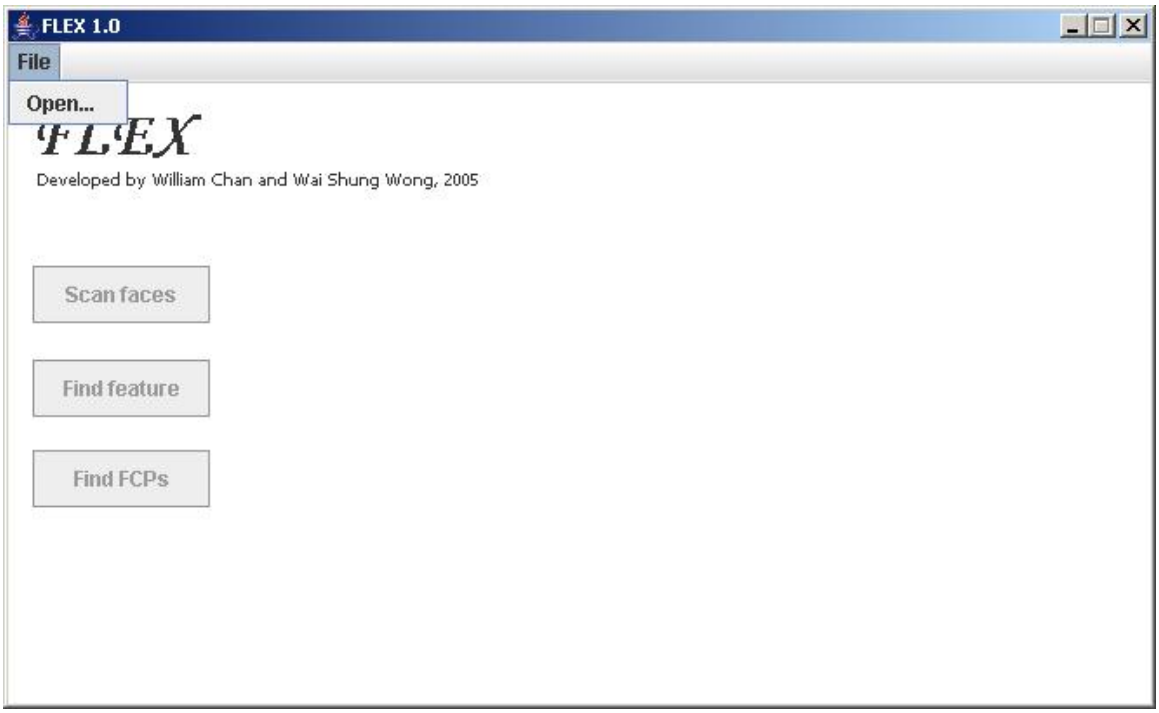


Figure 69: File-Open menu.

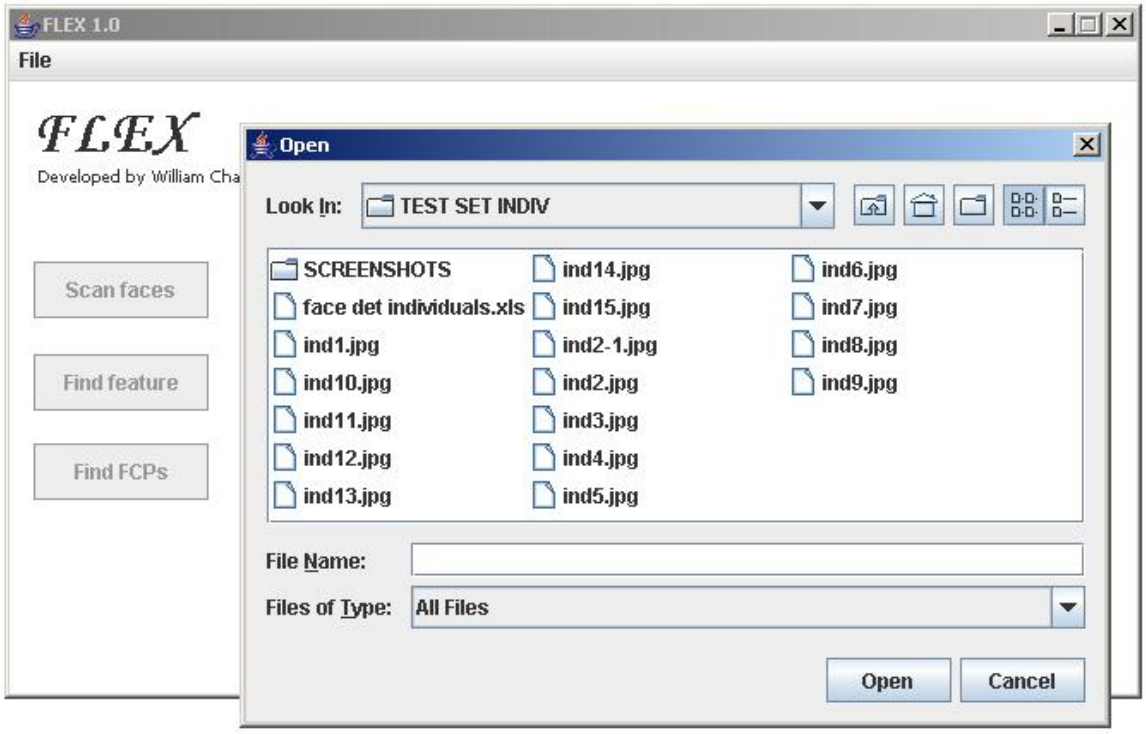


Figure 70: File-open dialog.

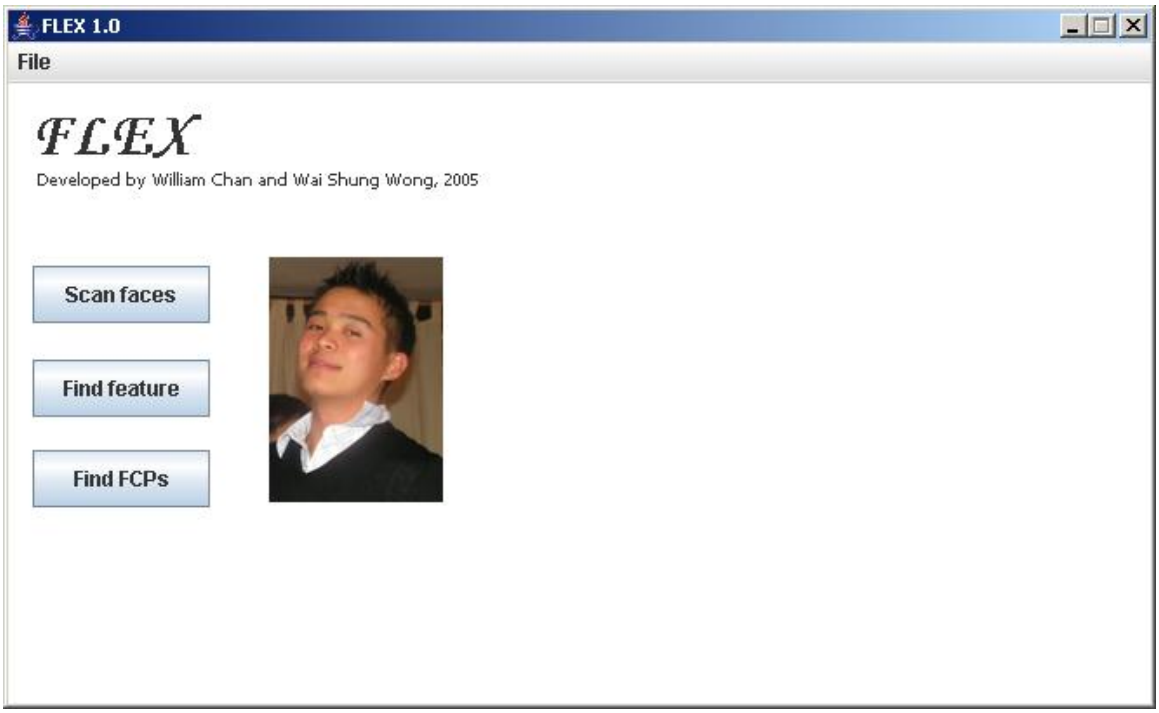


Figure 71: Image selected for detection.

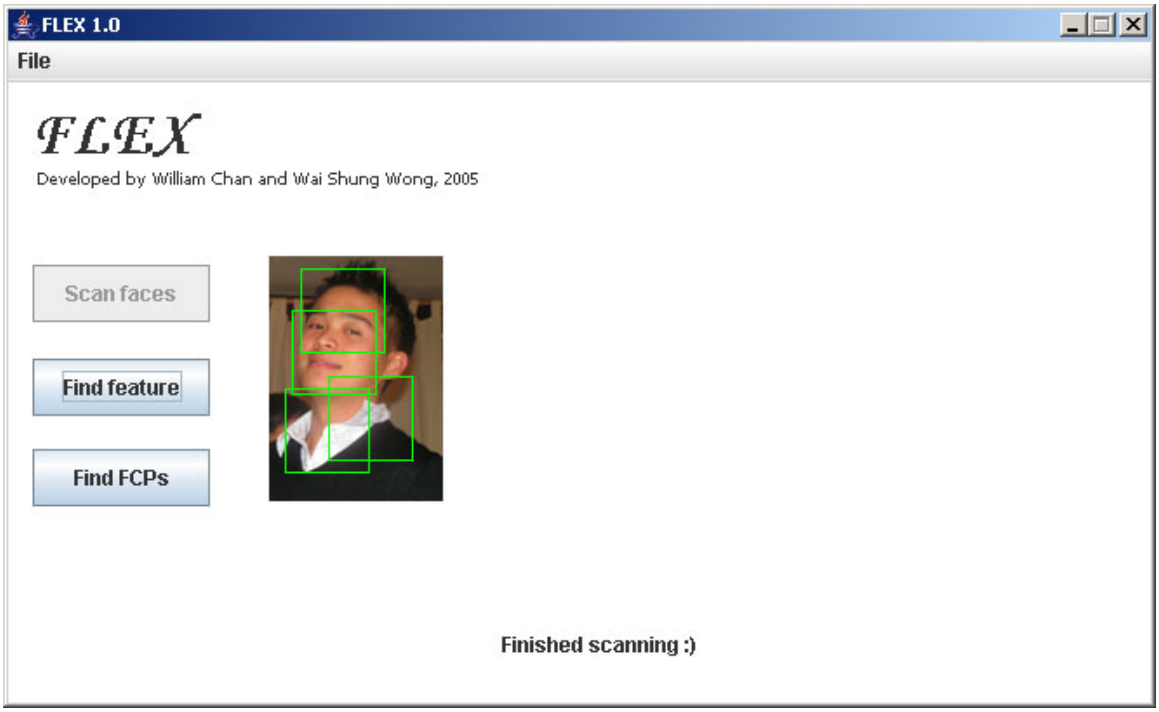


Figure 72: Detection result after pressing the "Scan faces" button.

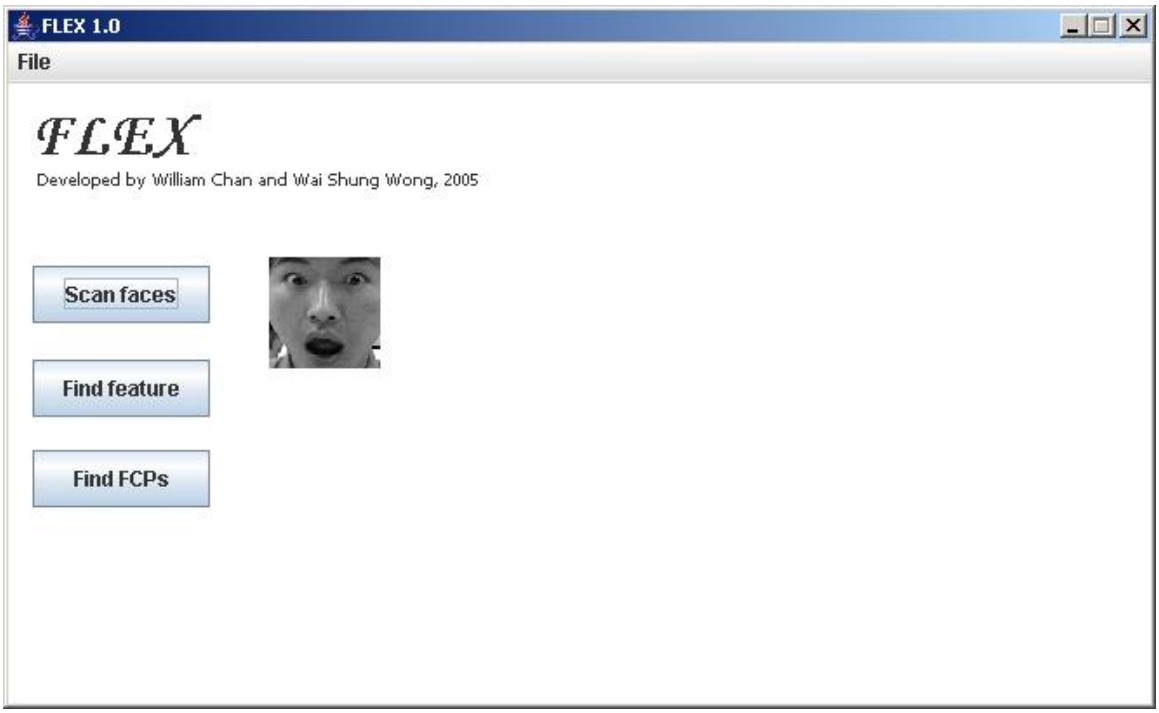


Figure 73: Image selected for FCP detection.

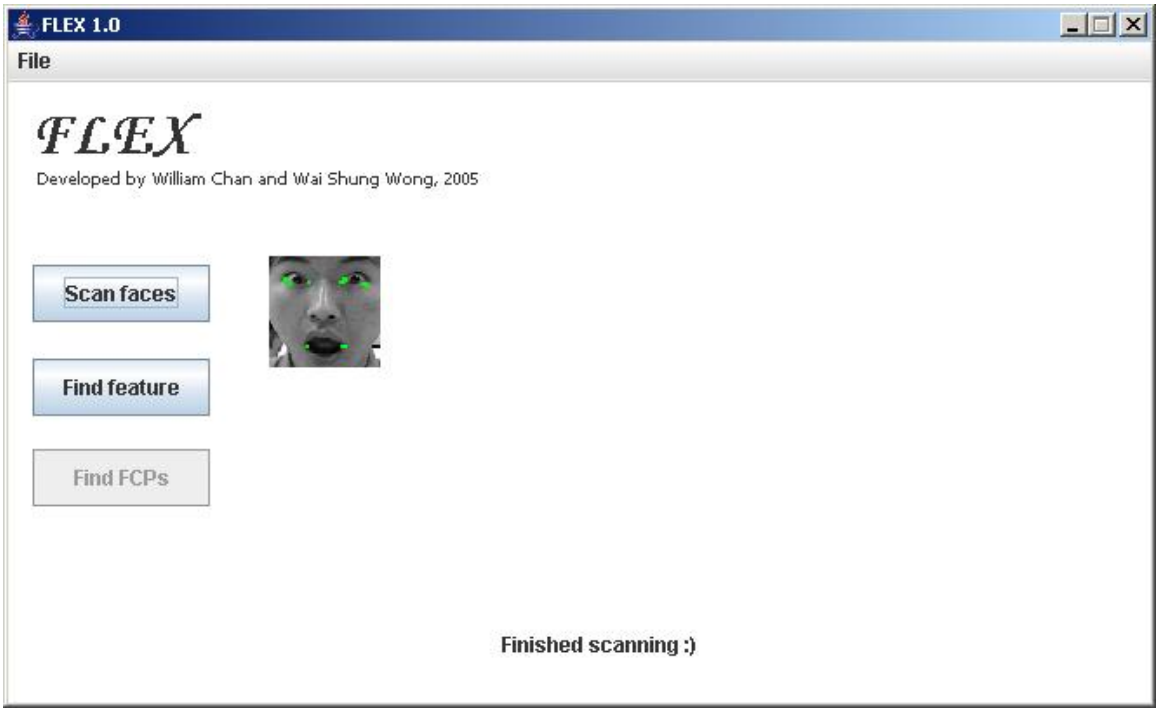


Figure 74: Detection result after pressing "Find FCPs" button.

10

System Test

In this chapter we discuss the testing of the system. To test its performance we designed a test plan. In this test plan, two modules (face detection and FCP detection) are considered separately and as one unit. This plan will be presented in section 10.1. Section 10.2 and 10.3 discuss the test of the face detection module and the FCP detection module respectively. Tests of both modules linked together are given in section 10.4.

10.1 Test Plan

We consider three test objects: face detection, FCP detection and FLEX. In FLEX face detection and FCP detection is combined in the final system. The images contained in the test sets are given in appendix B. The images used for testing the face detection module are randomly collected from the internet and from our own image collection. The images are of different sizes. The images used for testing the FCP detection module are from the BioID and Carnegie Mellon face database.

Face detection

The test for the face detection module can be divided into two parts:

FTO1: test the performance with images of single full-frontal faces. The test set consists of 15 images and will be referenced as Face Test Set1 (**FTS1**).

FTO2: test the performance with images containing two or more faces. The test set consists of 15 images and will be referenced as Face Test Set2 (**FTS2**).

FCP detection

The test for the FCP detection module is structured as follows:

FCPTO1: test the module on input images containing solely a full-frontal face. The test set is referenced as FCP Test Set1 (**FCPTS1**).

Integrated system

ISTO1: test the integration of the two modules. FCP detection is done on the output faces provided by the face detection module. The test set is referenced as Integrated System Test Set (**ISTS1)**.

10.2 Face detection module test

To test this module we choose to test on two sets. The first set (**FTS1**) consists of individual frontal faces and the second set (**FTS2**) consists of images with multiple faces. Each table contains the following columns with data:

- **Test image number (Nr.)**: sequence number of the test image in test set
- **Image name (Image)**: name of the test image
- **Resolution**: the resolution of the image (h x w).
- **Type**: display the type of the image. The image can be either 8-bit or 24-bit.
- **Total number of detected faces (TDF)**: the number of faces detected.
- **True positives (TP)**: number of faces that are detected as faces and are indeed faces.
- **Multiple true positives (MTP)**: number of faces, other than the real face(s), that can be counted as correct detection(s) of the face(s).
- **False positives (FP)**: number of faces that are classified as a face, but are in fact non-faces. This number is equal to TDF – TP – MTP.
- **Missed faces (MF)**: number of faces that has not been detected.

The outputs from the FLEX application are not judged by FLEX itself but by visual inspection. In each test image, we consider only one true face. Other detections which can be interpreted as true faces will be counted as multiple true positive (MTP) detections. The results of the first test (**FTO1**) on the test set (**FTS1**) are given in the following table:

From the results in Table 36 the true positive rate is 93.3%. From the total detected faces there are 15 false positives. The precision of the detector is given by TP/(TP + FP) = 22/(22 + 15) = 0.595. All faces are detected, except for the one contained in test image 2. This missed face can be explained by the grouping function that is implemented in the final version of FLEX.

Recall that the training samples for the face detector (see Figure 11) contain some space on either the left or right side of the face. This means that the training samples have a certain amount of noise that will be observable in classification results. But this way, the final detector can still detect a face when the scanning window is not exactly around a face. Since the face detector is

scanning the input image on every pixel location in the input image, there will be multiple positive detections around one face. The implemented grouping function will group these positive detections into one rectangle. It does so by considering windows that in a range of two pixels distance there are more than three positive detections.

Table 36: test result of FTO1 on FTS1.

Nr.	Image	Resolution	Type	TDF	TP	MTP	FP	MF
1	ind1.jpg	98x70	24	1	1	0	0	0
2	ind2.jpg	101x80	24	0	0	0	0	1
3	ind3.jpg	112x80	24	6	1	1	4	0
4	ind4.jpg	77x61	24	1	1	0	0	0
5	ind5.jpg	96x60	24	1	1	0	0	0
6	ind6.jpg	97x80	24	1	1	0	0	0
7	ind7.jpg	83x110	24	3	1	2	0	0
8	ind8.jpg	88x70	24	3	1	0	2	0
9	ind9.jpg	80x60	8	3	1	1	1	0
10	ind10.jpg	141x100	24	4	1	0	3	0
11	ind11.jpg	69x100	24	2	1	1	0	0
12	ind12.jpg	89x100	24	2	1	1	0	0
13	ind13.jpg	101x100	24	4	1	1	2	0
14	ind14.jpg	70x100	24	2	1	1	0	0
15	ind15.jpg	198x100	24	4	1	0	3	0
			total	37	14	8	15	1

For test image number 2, if the grouping function is switched off, we can see that the face was actually detected by FLEX. The implementation of the grouping function in the final system simply discards this positive detection, because it is assumed that multiple detections should occur around the face. It is possible that the size of the face is missed by the scaled classifiers. This is the trade-off in scaling that cannot be resolved. The relative high number of false positives can be explained by the fact that in the final implementation of FLEX we used only a five layered classifier. For better results, either more layers of classifiers need to be trained or different parameters need to be set for the classifiers. With the current settings, we can get a relative high true positive rate, but the false positive rate is also high. If we want to decrease the number of false positives by adjusting the parameters, the true positive rate will also decrease.

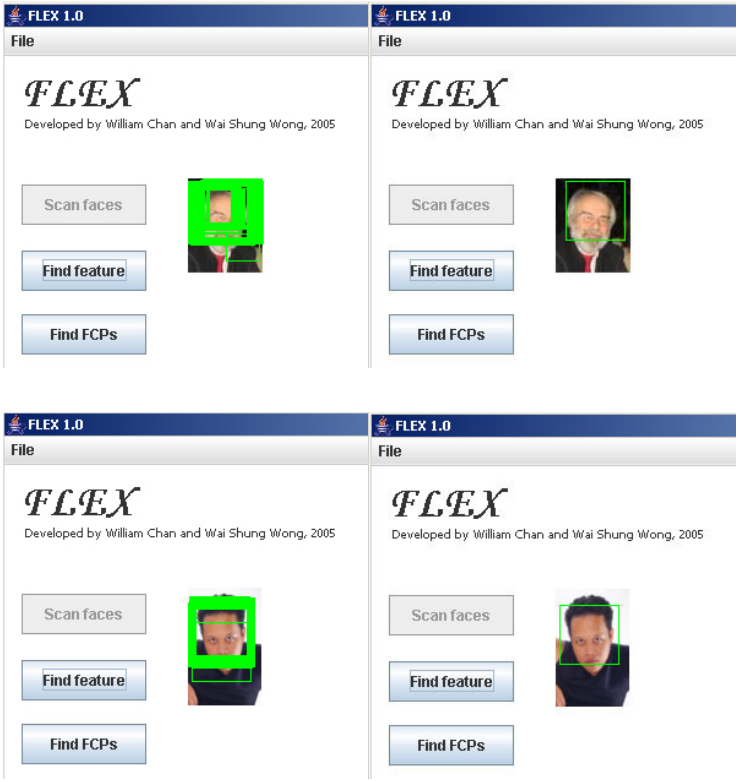


Figure 75: Illustration of the grouping function. The images on the left are scanned without grouping. The images on the right are scanned with grouping.

The results of the second test (**FTO2**) on the test set containing image with multiple faces (**FTS2**) are given in the following table:

Table 37: test result of FTO2 on FTS2.

Nr.	Image	Width	Height	Type	Faces	TDF	TP	FP	MF
1	group1.jpg	558	338	24	8	58	11	47	2
2	group2.jpg	312	226	24	8	35	9	26	1
3	group3.bmp	402	141	24	7	23	15	8	1
4	group4.bmp	240	178	24	8	24	14	10	0
5	group5.bmp	210	174	8	6	20	12	8	0
6	group6.bmp	170	153	24	6	18	7	11	0
7	group7.bmp	302	212	24	4	19	5	14	0
8	group8.bmp	188	110	8	4	5	4	1	2
9	group11.jpg	126	200	24	4	12	7	5	0
10	group12.jpg	253	280	24	3	3	3	0	1
11	group14.jpg	179	354	24	5	12	5	7	1
12	group16.jpg	250	175	24	5	8	7	1	1
13	group19.jpg	381	384	24	5	31	6	25	0
14	group20.jpg	255	190	24	5	26	7	19	0
15	group23.jpg	305	240	24	3	14	4	10	0
				total	81	308	116	192	9

In Table 37 we can see that out of the total number 81 faces in 15 images, 9 are missed. The true positive rate is 88.9% and there are 191 false positives. The number of false positives is relatively high compared to the number of positive detections. Again, this high number of false positives is due to the number of classifiers we trained and implemented in the final version of FLEX. Some of the faces are missed because they are (slightly) rotated or partly occluded by other objects. It is also possible that a face is missed because of the strictness of the grouping function. In the neighbour of two pixels around the face, it is assumed that there are more than three positive detections.

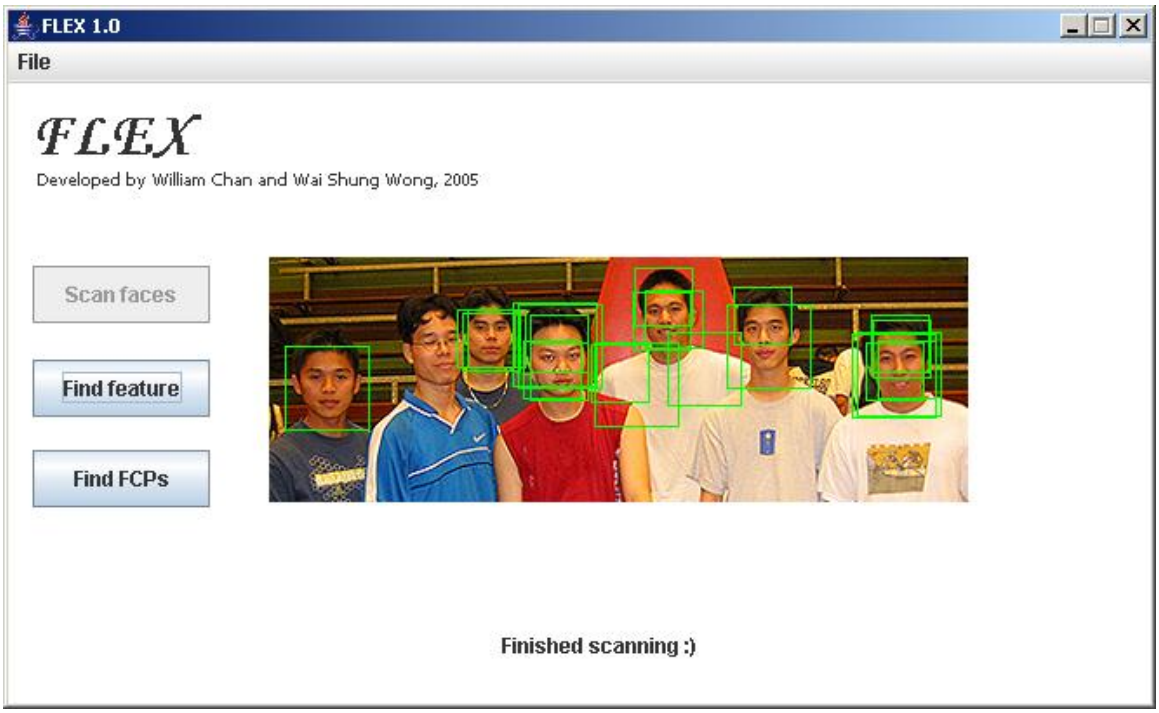


Figure 76: test result on test group3.jpg of set FTS1.

10.3 Facial Characteristic Points detection module test

For this part of the test, we used the test set **FCPTS1**. It contains 22 images of 64x64 pixels, selected from the BioID and Carnegie Mellon face database. 11 persons were selected, each with two different facial expressions. Note that the test set does not contain any faces with glasses, beard or moustache. The **FCPTS1** set is used to test each FCP. The results of the FCPs are arranged separately in a table. Each table contains the following columns with data:

- **Test image number (Nr.)**: sequence number of the test image in test set
- **Image name (Image)**: name of the test image
- **Type**: display the type of the image. The image can be either 8-bit or 24-bit.

- **Number of Candidate FCPs (NOC)**: display the number of corners that are detected within the region of the FCP. These are candidates for the FCP.
- **Total number of detected FCPs (TD FCP)**: the number of candidates FCPs that are classified as FCPs. This number should be smaller than NOC.
- **True positives (TP)**: number of corners that are detected as FCPs and are indeed FCPs.
- **Multiple true positives (MTP)**: number of corners, other than the real FCP, that can be counted as a correct detection of the FCP.
- **False positives (FP)**: number of corners that are classified as FCP, but are in fact non-FCPs. This number is equal to TDFPC – TP – MTP.
- **True negatives (TN)**: number of corners that are classified as non-FCP, which are truly non-FCPs. This number is equal to NOC – TDFCP.
- **False negatives (TN)**: number of corners that are classified as non-FCPs, but they are FCPs. This number has the value 1 in case no true positives are detected (TP = 0), otherwise it has the value 0.

Table 38: Test result for left eye inner corner (LEIC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	52	1	1	0	0	51	0
2	A06.bmp	8	57	2	1	1	0	55	0
3	D00.bmp	8	77	13	1	5	7	64	0
4	D44.bmp	8	60	6	1	5	0	54	0
5	E04.bmp	8	64	2	1	1	0	62	0
6	E52.bmp	8	63	1	1	0	0	62	0
7	F28.bmp	8	36	0	0	0	0	36	1
8	F37.bmp	8	52	2	1	1	0	50	0
9	I03.bmp	8	46	0	0	0	0	46	1
10	I35.bmp	8	53	1	1	0	0	52	0
11	Bioid_0256.bmp	24	59	8	1	7	0	51	0
12	Bioid_0257bmp	24	61	6	1	3	2	55	0
13	Bioid_0419.bmp	24	83	14	1	5	8	69	0
14	Bioid_0420.bmp	24	72	8	1	5	2	64	0
15	Bioid_0657.bmp	24	77	6	1	4	1	71	0
16	Bioid_0658.bmp	24	59	2	1	1	0	57	0
17	Bioid_0717.bmp	24	58	4	1	1	2	54	0
18	Bioid_0721.bmp	24	79	8	1	5	2	71	0
19	Bioid_1079.bmp	24	43	0	0	0	0	43	1
20	Bioid_1083.bmp	24	81	3	0	0	3	78	0
21	Bioid_1517.bmp	24	74	18	1	7	10	56	0
22	Bioid_1518.bmp	24	66	9	1	4	4	57	0
		total	1372	114	18	55	41	1258	3

By definition the true positive rate is the number of true positives divided by the total number of positives. In Table 38 the true positive rate is (tpr = 18 / 22 =) 81.82 %. The false positive rate, which is the number of false positives divided by the total number of negatives, is fpr = 41/ (1372 – 22 – (22 * 8)) = 3.49%.

Figure 77: Screen shots of FCP detection on the left eye inner corner.

Table 39: Test result for right eye inner corner (REIC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	72	7	1	2	4	65	0
2	A06.bmp	8	54	0	0	0	0	54	1
3	D00.bmp	8	80	2	1	1	0	78	0
4	D44.bmp	8	60	0	0	0	0	60	1
5	E04.bmp	8	80	6	1	3	2	74	0
6	E52.bmp	8	67	3	1	2	0	64	0
7	F28.bmp	8	79	1	1	0	0	78	0
8	F37.bmp	8	79	2	0	0	2	77	1
9	I03.bmp	8	60	10	1	3	6	50	0
10	I35.bmp	8	62	7	1	3	3	55	0
11	Bioid_0256.bmp	24	75	18	1	3	14	57	0
12	Bioid_0257bmp	24	78	9	1	0	8	69	0
13	Bioid_0419.bmp	24	36	4	1	1	2	32	0
14	Bioid_0420.bmp	24	57	5	1	0	4	52	0
15	Bioid_0657.bmp	24	73	5	1	2	2	68	0
16	Bioid_0658.bmp	24	81	7	1	2	4	74	0
17	Bioid_0717.bmp	24	33	4	1	0	3	29	0
18	Bioid_0721.bmp	24	60	6	1	1	4	54	0
19	Bioid_1079.bmp	24	57	0	0	0	0	57	1
20	Bioid_1083.bmp	24	68	3	1	0	2	65	0
21	Bioid_1517.bmp	24	63	17	1	2	14	46	0
22	Bioid_1518.bmp	24	54	11	1	1	9	43	0
		total	1428	127	18	26	83	1301	4

Figure 78: Screen shots of FCP detection on right eye inner corner.

For the right eye inner corner, the tpr = 18 / 22 = 81.82 % and fpr = 83/ (1428 – (9*22)) = 6.75%. From the tables and the screen shot pictures, it can be concluded that the inner corners of the eyes can be detected quite good. For the test images in which the eye corners are missed, the eye corners are either blurry by shadow, make-up or baggy eye lids. Also if the intensity difference between the eye and the skin is too minimal, the FCP can not be detected. Following tables show the FCP detection result of LEOC and REOC.

For the left eye outer corner, tpr is 63.64% and fpr is 5.94%. For the right eye outer corner, tpr is 81.82% with an fpr of 16.67% (see Table 40 and Table 41). It can be concluded that the outer corners in our test images are hard to detect. In both cases, we can see that the missed FCPs occur mostly in pictures from the BioID dataset. The faces in BioID are aligned in another way than the faces from the Carnegie Mellon set. From the dataset (see appendix B) we can see that the eye corners of the BioID faces are very close to the border. As a result, FLEX cannot extract a good corner sample to let RVM classify. To test if the eye corners can be detected if they are aligned correctly, we manually shifted the BioID faces some pixels to the left for REOC testing and some pixels to the right for LEOC testing. The result are as we expected, the FCPs can be detected by FLEX. The third picture of Figure 79 and the first and third picture of Figure 80 are from BioID after shifting.

In the case of the left eye outer corner (LEOC) detection, we tried to achieve a low fpr by adjusting some parameters in the classifier. The result is that the tpr also decreasing. For REOC, we tried to do the opposite. The outcome is a high tpr with a high fpr.

Table 40: Test result for FPC detection of left eye outer corner (LEOC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	52	2	1	1	0	50	0
2	A06.bmp	8	31	4	1	2	1	27	0
3	D00.bmp	8	33	8	1	2	5	25	0
4	D44.bmp	8	27	7	1	4	2	20	0
5	E04.bmp	8	26	0	0	0	0	26	1
6	E52.bmp	8	25	2	1	1	0	23	0
7	F28.bmp	8	27	3	1	1	1	24	0
8	F37.bmp	8	51	3	1	2	0	48	0
9	I03.bmp	8	43	5	1	3	1	38	0
10	I35.bmp	8	36	11	1	2	8	25	0
11	Bioid_0256.bmp	24	33	0	0	0	0	33	1
12	Bioid_0257bmp	24	34	0	0	0	0	34	1
13	Bioid_0419.bmp	24	27	0	0	0	0	27	1
14	Bioid_0420.bmp	24	36	1	0	0	1	35	0
15	Bioid_0657.bmp	24	18	0	0	0	0	18	1
16	Bioid_0658.bmp	24	18	0	0	0	0	18	1
17	Bioid_0717.bmp	24	36	3	1	0	2	33	0
18	Bioid_0721.bmp	24	18	2	1	0	1	16	0
19	Bioid_1079.bmp	24	33	1	1	0	0	32	0
20	Bioid_1083.bmp	24	43	5	1	1	3	38	0
21	Bioid_1517.bmp	24	33	4	1	1	2	29	0
22	Bioid_1518.bmp	24	40	4	0	0	4	36	0
		total	720	65	14	20	31	655	6

Figure 79: Screen shots of FCP detection of left eye outer corner.

Table 41: Test result for FCP detection of right eye outer corner (REOC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	22	11	1	7	3	11	0
2	A06.bmp	8	43	7	1	1	5	36	0
3	D00.bmp	8	36	7	1	3	3	29	0
4	D44.bmp	8	24	9	1	5	3	15	0
5	E04.bmp	8	18	8	1	2	5	10	0
6	E52.bmp	8	27	8	1	2	5	19	0
7	F28.bmp	8	31	15	1	4	10	16	0
8	F37.bmp	8	22	3	1	0	2	19	0
9	I03.bmp	8	9	3	1	1	1	6	0
10	I35.bmp	8	27	7	1	5	1	20	0
11	Bioid_0256.bmp	24	42	7	1	5	1	35	0
12	Bioid_0257bmp	24	40	9	1	5	3	31	0
13	Bioid_0419.bmp	24	18	2	0	0	2	16	1
14	Bioid_0420.bmp	24	46	3	0	0	3	43	1
15	Bioid_0657.bmp	24	33	6	1	3	2	27	0
16	Bioid_0658.bmp	24	27	0	0	0	0	27	1
17	Bioid_0717.bmp	24	22	10	1	3	6	12	0
18	Bioid_0721.bmp	24	16	8	1	4	3	8	0
19	Bioid_1079.bmp	24	18	2	0	0	2	16	1
20	Bioid_1083.bmp	24	27	5	1	3	1	22	0
21	Bioid_1517.bmp	24	43	8	1	2	5	35	0
22	Bioid_1518.bmp	24	33	7	1	1	5	26	0
		total	624	145	18	56	71	479	4

Figure 80: Screen shots of FCP detection of right eye outer corner.

Table 42: Test result for FCP detection of mouth left corner (MLC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	302	36	1	7	28	266	0
2	A06.bmp	8	232	9	1	4	4	223	0
3	D00.bmp	8	188	14	1	3	10	174	0
4	D44.bmp	8	165	12	1	3	8	153	0
5	E04.bmp	8	206	32	1	5	26	174	0
6	E52.bmp	8	156	8	1	2	5	148	0
7	F28.bmp	8	200	3	1	0	2	197	0
8	F37.bmp	8	211	3	1	0	2	208	0
9	I03.bmp	8	194	16	1	5	10	178	0
10	I35.bmp	8	248	16	1	5	10	232	0
11	Bioid_0256.bmp	24	236	6	1	1	4	230	0
12	Bioid_0257bmp	24	284	15	1	3	11	269	0
13	Bioid_0419.bmp	24	216	3	1	2	0	213	0
14	Bioid_0420.bmp	24	210	15	1	4	10	195	0
15	Bioid_0657.bmp	24	265	6	1	1	4	259	0
16	Bioid_0658.bmp	24	171	5	1	3	1	166	0
17	Bioid_0717.bmp	24	217	0	0	0	0	217	1
18	Bioid_0721.bmp	24	175	1	1	0	0	174	0
19	Bioid_1079.bmp	24	236	0	0	0	0	236	1
20	Bioid_1083.bmp	24	211	0	0	0	0	211	1
21	Bioid_1517.bmp	24	240	12	1	2	9	228	0
22	Bioid_1518.bmp	24	237	9	1	3	5	228	0
		total	4800	221	19	53	149	4579	3

Figure 81: Screen shots of FCP detection on mouth left corner (MLC).

Table 43: Test result for FCP detection of mouth right corner (MRC).

Nr.	Image	Type	NOC	TD FCP	TP	MTP	FP	TN	FN
1	A01.bmp	8	207	27	1	8	18	180	0
2	A06.bmp	8	261	15	1	3	11	246	0
3	D00.bmp	8	186	23	1	8	14	163	0
4	D44.bmp	8	243	11	1	0	10	232	0
5	E04.bmp	8	213	26	1	8	17	187	0
6	E52.bmp	8	198	28	1	4	23	170	0
7	F28.bmp	8	237	2	0	0	2	235	1
8	F37.bmp	8	258	7	0	0	7	251	1
9	I03.bmp	8	207	13	1	4	8	194	0
10	I35.bmp	8	253	23	1	5	17	230	0
11	Bioid_0256.bmp	24	197	16	1	7	8	181	0
12	Bioid_0257bmp	24	210	15	1	5	9	195	0
13	Bioid_0419.bmp	24	198	13	1	5	7	185	0
14	Bioid_0420.bmp	24	185	22	1	4	17	163	0
15	Bioid_0657.bmp	24	205	14	1	5	8	191	0
16	Bioid_0658.bmp	24	214	15	1	7	7	199	0
17	Bioid_0717.bmp	24	211	11	1	7	3	200	0
18	Bioid_0721.bmp	24	229	12	1	6	5	217	0
19	Bioid_1079.bmp	24	230	21	1	7	13	209	0
20	Bioid_1083.bmp	24	223	9	1	3	5	214	0
21	Bioid_1517.bmp	24	222	9	1	3	5	213	0
22	Bioid_1518.bmp	24	155	2	1	1	0	153	0
		total	4742	334	20	100	214	4408	2

As stated earlier, mouth corners are unstable points, which means the position and the shape of the mouth corners are variable. Still, the tpr and fpr for MLC is 86.36 % and 3.24%, respectively. For MRC these are 90.91% and 4.71% respectively (see Table 42 and Table 43). It can be concluded that FLEX is able to detect mouth corners very well.

Figure 82: Screen shots of FCP detection on mouth right corner (MRC).

From chapter 8 another approach has been applied to extract the FCPs, which are unable to be detected with the corner detector combined with RVM. These points are for example the FCPs on top of the eyes, bottom of the eyes, top of the upper lip and bottom of the lower lip. The integral projection method is then introduced to solve this problem. Following are the results of the integral projection on eyes and mouth. As we know for this method the region must be specified. For this purpose actually an eye and mouth detector is trained. Unfortunately, the results were not acceptable to test the combination of this region detector together with the projection method. For this problem we manually submit the region information of each test sample in **FTS2**.

Table 44: Integral projection results on left eye.

Nr.	Image	Type	TP	FP
1	A01.bmp	8	4	0
2	A06.bmp	8	4	0
3	D00.bmp	8	1	3
4	D44.bmp	8	2	2
5	E04.bmp	8	2	2
6	E52.bmp	8	2	2
7	F28.bmp	8	4	0
8	F37.bmp	8	4	0
9	I03.bmp	8	4	0
10	I35.bmp	8	4	0
11	Bioid_0256.bmp	24	3	1
12	Bioid_0257bmp	24	4	0
13	Bioid_0419.bmp	24	4	0
14	Bioid_0420.bmp	24	3	1
15	Bioid_0657.bmp	24	3	1
16	Bioid_0658.bmp	24	3	1
17	Bioid_0717.bmp	24	4	0
18	Bioid_0721.bmp	24	2	2
19	Bioid_1079.bmp	24	4	0
20	Bioid_1083.bmp	24	4	0
21	Bioid_1517.bmp	24	4	0
22	Bioid_1518.bmp	24	4	0
		total	73	15

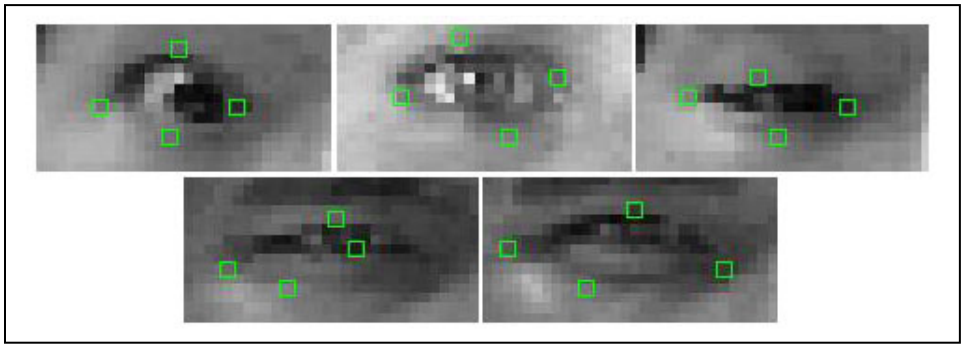


Figure 83: Screen shots of integral projection on left eye.

Table 45: Integral projection results on right eye.

Nr.	Image	Type	TP	FP
1	A01.bmp	8	4	0
2	A06.bmp	8	4	0
3	D00.bmp	8	2	2
4	D44.bmp	8	2	2
5	E04.bmp	8	2	2
6	E52.bmp	8	3	1
7	F28.bmp	8	4	0
8	F37.bmp	8	4	0
9	I03.bmp	8	4	0
10	I35.bmp	8	4	0
11	Bioid_0256.bmp	24	4	0
12	Bioid_0257bmp	24	4	0
13	Bioid_0419.bmp	24	4	0
14	Bioid_0420.bmp	24	4	0
15	Bioid_0657.bmp	24	3	1
16	Bioid_0658.bmp	24	3	1
17	Bioid_0717.bmp	24	4	0
18	Bioid_0721.bmp	24	3	1
19	Bioid_1079.bmp	24	4	0
20	Bioid_1083.bmp	24	4	0
21	Bioid_1517.bmp	24	4	0
22	Bioid_1518.bmp	24	4	0
		total	78	10

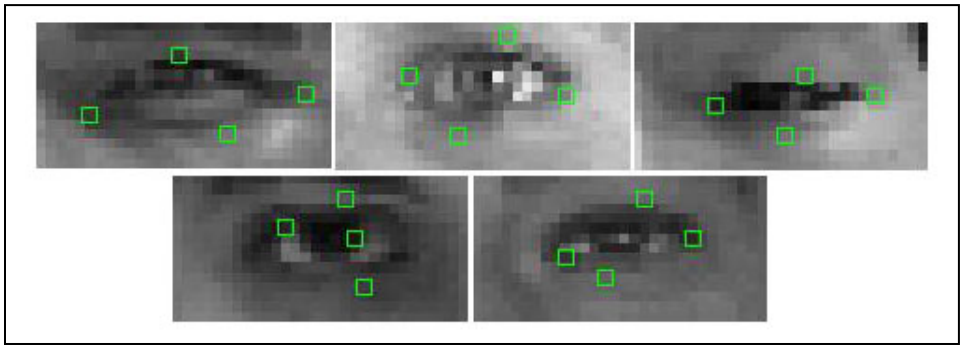


Figure 84: Screen shots of integral projection on right eye.

From the results it can be observed that noise like shadow, make-up, and eye-bag has influence on the results of this method. The results for the left eye are tpr: 82.95 % and fpr: 17.05%. For the right eye, these are tpr: 88.64 % and fpr: 11.36%.

Table 46: Integral projection results on mouth.

Nr.	Image	Type	TP	FP
1	A01.bmp	8	2	2
2	A06.bmp	8	4	0
3	D00.bmp	8	3	1
4	D44.bmp	8	3	1
5	E04.bmp	8	2	2
6	E52.bmp	8	4	0
7	F28.bmp	8	4	0
8	F37.bmp	8	4	0
9	I03.bmp	8	4	0
10	I35.bmp	8	4	0
11	Bioid_0256.bmp	24	4	0
12	Bioid_0257bmp	24	4	0
13	Bioid_0419.bmp	24	4	0
14	Bioid_0420.bmp	24	4	0
15	Bioid_0657.bmp	24	3	1
16	Bioid_0658.bmp	24	3	1
17	Bioid_0717.bmp	24	4	0
18	Bioid_0721.bmp	24	4	0
19	Bioid_1079.bmp	24	4	0
20	Bioid_1083.bmp	24	4	0
21	Bioid_1517.bmp	24	4	0
22	Bioid_1518.bmp	24	4	0
		total	80	8

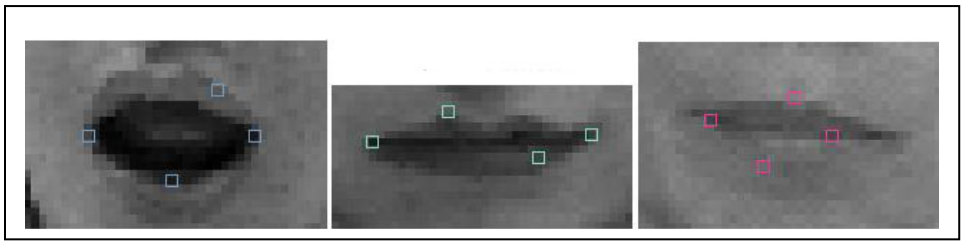


Figure 85: Screen shots of integral projection on mouth.

In the case of applying the projection method on the mouth it can be stated that this method is really sensitive to the region that contains the feature. For example, in Figure 85 the mouth on the left is not fully contained in the region. As a result the FCP on the bottom of the lower lip is detected on the top of the lower lip. This again strengthens the point that if integral projection is applied in FLEX, the region of the feature must be extracted very accurately. Besides the exact location of the points, it can also be observed that the boundaries of the features can be located very precisely. The tpr for mouth is 90.91 % and the fpr is 9.08%.

10.4 Integrated system test

In order to detect the FCPs from the face, the face must satisfy certain conditions. One of these conditions is that the face must be upright full-frontal. The reason for this is that if the face is rotated, the feature might be occluded by other part of the face. At the training of the face detector, a database is used which contains images with slightly rotated faces. This results in faces which cannot be processed further by invoking the FCP detection.

At the design of the FCP detection module it is assumed that full-frontal faces are available at a size of 64x64 image pixels. The face detector is designed to detect faces with a minimal size of 24x24 pixels. Detecting a face in an image of a bigger size is done by scaling the classifier as discussed earlier. In theory, the face detection nodule as we have designed it is suited for application in combination with the FCP detection module. If the face detection module and the FCP detection module are linked together, the performance of the FCP detection is dependant on the performance of the face detector. In the current state the face detector is not as good as we want it to be and needs to be improved. Therefore, it is not meaningful to execute the testing of this test object (**ISTO1**).

Part V

Conclusion and Future Works

11

Conclusions, Discussion and Future works

We are now concluding this thesis and propose some ideas to improve the current system. We really hope that you enjoyed reading this thesis as much as we enjoyed writing it. It all comes down to one thing. We started with a problem and soon made a thesis assignment of it. From then on, we are off to the battlefield: the battle of proving yourself worthy of being a scientist.

11.1 Conclusions and discussion

We have presented an approach using a sparse learning model as the first step towards a fully automatic facial expression recognition system. This system already exists as an online Facial Expression Dictionary. In the current state of the system, providing a face image as input requires the user to manually select all the FCPs for further processing. This interaction is not desired since there are 30 FCPs and is certainly not making the system user-friendly. In our thesis project we tackle this problem by automating the FCP detection process. Therefore, the thesis assignment is defined as follows:

- *Literature survey*: research the related works on the topics of face detection, facial characteristic point detection, facial expression recognition and classification methods.

- *Model design*: design a model as a solution to the problem of automatic facial characteristic point detection. This model is built of multiple methods and algorithms.

- *Prototype*: implement the designed model.

- *Tests*: write a test plan to test the prototype and depicts the results.

We did a research on the topics of face detection, facial characteristic point detection, facial expression recognition and classification methods by examining scientific papers and reports especially on the subject of using the sparse learning Relevance Vector Machine. After concluding the literature survey, we continued with the second part of our thesis assignment: the design of a model as a solution to the problem of fully automatic facial characteristic point detection.

The model we designed consists of a face detection module and a FCP detection module. The former allows the user to input an arbitrary image containing one or multiple faces. The face detection module extracts the faces and invokes the FCP detection module on these faces. On its turn, the FCP detection module automatically extracts the predefined FCPs. From the FCPs a corresponding facial expression can be matched.

Face detection

We designed a learning model: WUX-values Training Application (WUXTRAP) to boost the performance of RVM. This model consists of different techniques and algorithms. To detect faces from images, we first need to learn the RVM to differentiate between faces and non-faces. The first requirement to learn the RVM model is to have a dataset of faces and non-faces. These databases can be of influence on the training results since the RVM has to learn from these samples. There exist numerous face databases from which we can choose to use as our dataset.

The learning procedure is based on the AdaBoost learning algorithm. This algorithm is perfectly suited for the selection of the best features that boost up the performance of the classifier. As known for AdaBoost training, it is slow since it contains a brute force search. In addition, training of the RVM itself is relatively slow. And since they are combined, there is a continuous feedback from RVM to AdaBoost and the other way around. A genetic search algorithm is added to improve the learning speed. Instead of a training time in the order of weeks/months, this is reduced to hours/days (on an AMD Athlon™ XP 2200+ 1.80 GHz processor with 512 MB RAM). Note that the size of the chosen training dataset is also significant for the speed of the training.

After the learning procedure, faces can be distinguished from non-faces using the trained RVMs. Remind that we want to find faces in an input image by scanning the whole image. The number of scanning windows is huge. Most of these are non-faces. So, a cascaded structure of classifiers is introduced to quickly discard most of these non-faces. A cascade consists of several layers of classifiers. Each classifier is a combination of a number of RVMs. A practical problem that we

encounter incorporating the cascade technique is that a lot of RVMs need to be trained. In our case, since we are limited to the time we have for this project, our solution is to use more computers and all simultaneously for training. Training more and better RVMs makes the final system more robust.

We have managed to apply the RVM for face detection. However, the test results show that improvement needs to be made. In the current state, the face detector consists of only five layers of classifiers. Recall that in [Viol01] a cascade of 32 layers with over 4000 features is used. To get better results, more classifiers need to be added to FLEX.

Facial characteristic point detection

The same learning model for training the face detection module is used for the FCP detection module. Unlike in the case of face detection, no databases of FCPs exist which we can use as our dataset. These databases are extracted manually by us from the BioID and Carnegie Mellon face database. Note that it is really hard to be very precise at clicking the right FCP since there are noise and fuzziness around these points. As we know, the quality of the datasets is of influence on the training results.

For the detection of the FCPs, a corner detection algorithm is used to filter out the non-FCPs. We have chosen for a combination of the Harris corner detection algorithm and the Sojka corner detection algorithm. Unfortunately, not all of the non-FCPs can be filtered out by these corner detectors. For this, we rely on the corresponding RVMs. The performance of the RVM in the final system is actually determined by that of the corner detectors.

For the FCPs that cannot be detected by the corner detectors, we use the Hybrid Projection technique. This technique is applied on the corresponding facial feature (eye, eye brow and mouth) on which the FCP is localized. Therefore, RVMs are trained to extract these facial features before applying the projection method.

The results in the final system show that some of the FCPs can be detected better than others. This is concluded by looking at its positive detection rate and its false positive rate. The reason for the relative poor performance of some FCPs is probably because the FCP itself is non-stable. For instance, the mouth corners can take different shapes at different expressions. To detect the FCPs we need to account that noise is very probable at corner regions. Taking this into account it means that at the training of the RVM noise is included in the training samples. This affects the final performance of the RVM. It is a trade-off that needs to be made. In the case of invoking the

projection method, finding the boundaries is proven to be very robust, except if the feature boundary is distorted.

11.2 Future works

For future research the following items are be recommended:

- In the current situation, a detected face cannot be further processed by the FCP detection module if the face is slightly rotated. Some of the FCPs can be occluded by other parts of the face. The face detection module is trained on a database with unaligned faces. Some of them are slightly rotated to the left, some to the right, some looking up, etc. For the two modules to work together perfectly, the face detection module should be trained strictly on full frontal aligned faces. This is because the FCP detection module is designed to work with these faces.

- The WUXTRAP model may be improved by considering a faster implementation of the training application. This means that the training of AdaBoost and RVM can be improved. The current implementations of these algorithms are done in Matlab 6.5, which is known for its computational power but not for its speed. Also other variants of the AdaBoost learning algorithm can be considered. They differ in the updating schemes for the weights.

- In the face detection module, the scanning process can be speed up by other techniques. Using edge detectors plain backgrounds might be filtered out and pruned from being scanned. This reduces the overall scanning time on different resolutions.

- The performance of the system can also be improved by using an extended set of the Haar-like features. In our training model, we used only 5 simple features.

- The detection rate during training may be increased by incorporating the bootstrapping method. This method uses misclassified samples as training input in the next iteration. This way we can force the learning algorithm to adapt the output results from previous training rounds. We have not implemented this procedure in the current training model because this would certainly affect the training time negatively.

References

[Beau78] P.R. Beaudet. *Rotationally invariant image operators*. Proceedings of Fourth International Joint Conference on Pattern Recognition, Tokyo, 1978, p 579-583.

[Bish04] C.M. Bishop, M.E. Tipping. *Bayesian Regression and Classification*. Advances in Learning Theory: Methods, Models and Applications, NATO Science Series III: Computer and Systems Sciences, Vol. 190, 2004.

[Burg97] C.J.C. Burges, B. Schölkopf. *Improving the Accuracy and Speed of Support Vector Learning Machines*. Advances in Neural Network Information Processing Systems 9, Cambridge, MIT Press, 1997, p 375-381.

[Chan04] W. Chan, W.S. Wong. *Literature Survey: Relevance Vector Machine for Face Detection and Face Model Feature Extraction*. TU Delft, 2004.

[Deri93] R. Deriche, G. Giraudon. *A computational approach for corner and vertex detection*. International Journal of Computer Vision, Vol. 10-2, 1993, p 101-124.

[Dick03] A.R. Dick, M.J. Brooks. *Issue on Automated Visual Surveillance*. University of Adelaide, CRC for Sensor, Signal and Information Processing, Technology Park, Australia, 2003.

[Dikk04] H.J. Dikkers, M.A. Spaans, *Facial Recognition System for Driver Viligence Monitoring*. IEEE Computer Society Press, 2004.

[Duma98] S. Dumai, J. Platt, D. Heckermann, M. Sahami. *Inductive Learning Algorithms and Representations for Text Categorization*. Proceedings of Conference on Information and Knowledge Management, 1998.

[Ekma78] P. Ekman, W. Friesen. *Facial Action Coding System*. Consulting Psychologists Press, Inc., Palo Alto California, USA, 1978.

[Feng98] G.C. Feng, P.C. Yuen. *Variance projection function and its application to eye detection for human face recognition*. Elsevier Science B.V., 1998.

[Feng99] G.C. Feng, P.C. Yuen. *Multi Cues Eye Detection on Gray Intensity Image*. HK Baptist University, Department of Computer Science, HK, 1999.

[Hara97] F. Hara, H. Kobayashi, K. Tanaka and A. Tange. *Automatic Feature Extraction of Facial Organs and Contour*. IEEE International Workshop on Robot and Human Communication, 1997, p 386-391.

[Harr88] C.G. Harris, M. Stephens. *A combined corner and edge detector*. Proceedings 4th Alvey Vision Conference, Manchester, 1988, p 189-192.

[Huan04] W. Huang, Q. Xia. *Screen Estimation for a Novel Pointing Device Based on Corner Detection and Classification*. TU Delft, 2004.

[Joac97] T. Joachims. *Text Categorization with Support Vector Machines*. Techinical Reports, LS VIII Number 23, University of Dortmund, 1997.

[Jong02] E.J. de Jongh. *FED: An online facial expression dictionary as a first step in the creation of a complete nonverbal dictionary*. TU Delft, 2002.

[Jung02] D.-J. Jung, C.-W. Lee, Y.-C. Lee, S.-Y. Bak, J.-B. Kim, H. Kang, H.-J. Kim. *PCA-Base Real-Time Face Detection and Tracking*. International Technical Conference on Circuits/Systems, Computers and Communication, Vol. 1, 2002, p 615-618.

[Kear93] G.D. Kearney, S. McKenzie. *Machine interpretation of emotion: design of a memory-based expert system for interpreting facial expressions in terms of signalled emotions (JANUS)*. Cognitive Science 17, Vol. 4, 1993, p 589–622.

[Kitc82] L. Kitchen, A. Rosenfeld. *Gray-level Corner Detection*. Pattern Recognition Letters, Vol. 1, 1982, p 95-102

[Koba97] H. Kobayashi, F. Hara. *Facial Interaction Between Animated 3D Face Robot and Human Beings*. IEEE Computer Society Press, 1997, p 3732-3737.

[Mori93] S. Morishima, H. Harashima. *Emotion Space for Analysis and Synthesis of Facial Expression*. IEEE International Workshop on Robot and Human Communication, 1993, p 674-680.

[Osun97] E. Osuna, R. Freund, F. Girosi. *Training Support Vector Machine: an Application to Face Detection*. Proceedings of CVPR'97, 1997.

[Roth00] L.J.M. Rothkrantz, M. Pantic. *Expert Systems for Automatic Analysis of Facial Expressions*. Elsevier, Image and Computing, Vol. 18, 2000, p 881-905.

[Rowl98] H.A. Rowley, S. Baluja, T. Kanade. *Neural Network-Based Face Detection*. IEEE Computer Society Press, 1998.

[Smit97] S.M. Smith, J.M. Brady. *SUSAN – A New Approach to Low Level Image Processing*. International Journal of Computer Vision, Vol. 23, 1997, p 45-78.

[Sojk03] E. Sojka. *A New Approach to Detecting Corners in Digital Images*. Accepted for Publication in IEEE ICIP, 2003.

[Sojk03b] E. Sojka. http://www.cs.vsb.cz/sojka/. 2003.

[Sung98] K.-K. Sung, T. Poggio. *Example-Based Learning for View-Based Human Face Detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20-1, 1998.

[Tipp00a] M.E. Tipping. *The Relevance Vector Machine*. Advances in Neural Information Processing Systems. Vol. 12, 2000, p 652-658.

[Tipp00b] M.E. Tipping, C.M. Bishop. *Variational Relevance Vector Machines*. Uncertainty in Artificial Intelligence, 2000, p 46-53.

[Tipp01] M.E. Tipping. *Sparse Bayesian Learning and the Relevance Vector Machine*. Journal of Machine Learning Research, Vol. 1, 2001, p 211-244.

[Trep03] A. Treptow, A. Zell. *Combining Adaboost Learning and Evolutionary Search to Select Features for Real-time Object Detection*. University of Tuebingen, Department of Computer Science, Germany, 2003.

[Vapn96] V. Vapnik, B. Schölkopf, C.J.C. Burges. *Incorporating Invariances in Support Vector Learning Machines*. Artificial Neural Networks – ICANN'96, Berlin, 1996, p 47-52.

[Viol01] P. Viola, M. Jones. *Robust Real-time Object Detection*. Second International Workshop on Statistical and Computational Theories of Vision-Modeling, Learning, Computing, and Sampling, 2001.

[Zhao96] J. Zhao, G. Kearney. *Classifying facial emotions by backpropagation neural networks with fuzzy inputs*. in: International Conference on Neural Information Processing, Vol. 1, 1996, p 454–457.

[Zhou02] Z.-H. Zhou, X. Geng. *Projection Functions for Eye Detection*. State Key Labaratory for Novel Software Technology, NU, China, 2002.

Appendices

Appendix A: Cross-validation for RVM Kernel Selection

Table 11: RVM 2-fold cross validation result trained on feature 38978.

Kernel	# False +	# False -	# True +	# True -	Error rate %	SD
Gauss 0.5	112 122	182 139	322 357	384 382	27.75	2.33
Gauss 1.0	166 191	105 120	394 381	335 308	29.10	2.83
Gauss 2.0	138 150	131 107	375 387	356 356	26.30	0.85
Gauss 4.0	90 131	219 92	272 417	419 360	26.60	6.08
Gauss 5.0	143 113	73 178	444 305	340 404	25.35	5.30
Laplace 0.5	162 51	90 405	408 97	340 447	35.20	14.42
Laplace 1.0	112 142	129 124	379 368	380 366	25.35	1.77
Laplace 2.0	145 127	78 235	431 256	346 382	29.25	9.83
Laplace 3.0	99 161	216 93	292 399	393 347	28.45	4.31
Laplace 4.0	152 129	84 132	413 371	351 368	24.85	1.77
Laplace 5.0	85 178	192 69	333 406	390 347	26.20	2.12

Table 11: RVM 2-fold cross validation result trained on feature 28893.

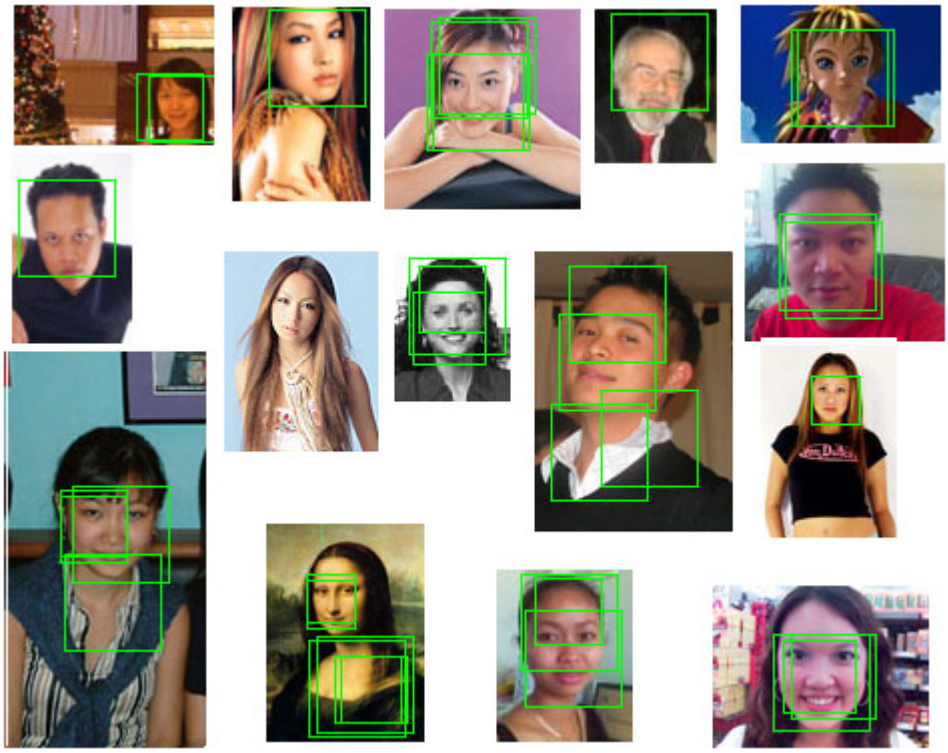
Kernel	# False +	# False -	# True +	# True -	Error rate %	SD
Gauss 0.5	20 96	475 248	39 238	466 418	41.95	10.68
Gauss 1.0	110 398	232 52	265 451	393 99	39.60	7.64
Gauss 2.0	159 75	141 334	342 183	358 408	35.45	7.71
Gauss 3.0	145 317	151 57	345 447	359 179	33.50	5.52
Gauss 4.0	136 92	129 277	374 220	361 411	31.70	7.35
Gauss 5.0	151 247	153 97	356 394	340 262	32.40	2.83
Laplace 0.5	107 174	167 86	330 417	396 323	26.70	0.99
Laplace 1.0	127 382	163 39	331 467	379 112	35.55	9.26
Laplace 2.0	311 150	62 117	444 377	183 356	32.00	7.50
Laplace 3.0	160 66	125 346	358 171	357 417	34.85	8.98
Laplace 4.0	124 138	116 144	391 349	369 369	26.10	2.97
Laplace 5.0	173 140	99 106	404 391	324 363	25.90	1.84

Table 11: RVM 2-fold cross validation result trained on feature 45297.

Kernel	# False +	# False -	# True +	# True -	Error rate %	SD
Gauss 0.5	20 90	474 346	39 141	467 423	46.50	4.10
Gauss 1.0	151 465	248 32	260 460	341 43	44.80	6.93
Gauss 2.0	205 183	194 190	292 324	309 303	38.60	1.84
Gauss 3.0	178 150	177 182	328 313	317 355	34.35	1.63
Gauss 4.0	155 196	174 185	314 327	357 292	35.50	3.68
Gauss 5.0	81 175	310 165	187 338	422 322	36.55	3.61
Laplace 0.5	25 186	470 173	41 316	464 325	42.70	9.62
Laplace 1.0	188 174	178 172	309 341	325 313	35.60	1.41
Laplace 2.0	418 177	53 184	454 309	75 330	41.60	7.78
Laplace 3.0	67 217	361 127	142 370	430 286	38.60	5.94
Laplace 4.0	64 108	374 305	116 205	446 382	42.55	1.77
Laplace 5.0	177 111	144 317	350 189	329 383	37.45	7.57

Appendix B: Test Sets

Face Test Set 1 (FTS1):



Face Test Set 2 (FTS2):

FCP Test Set 2 (FCPTS1):