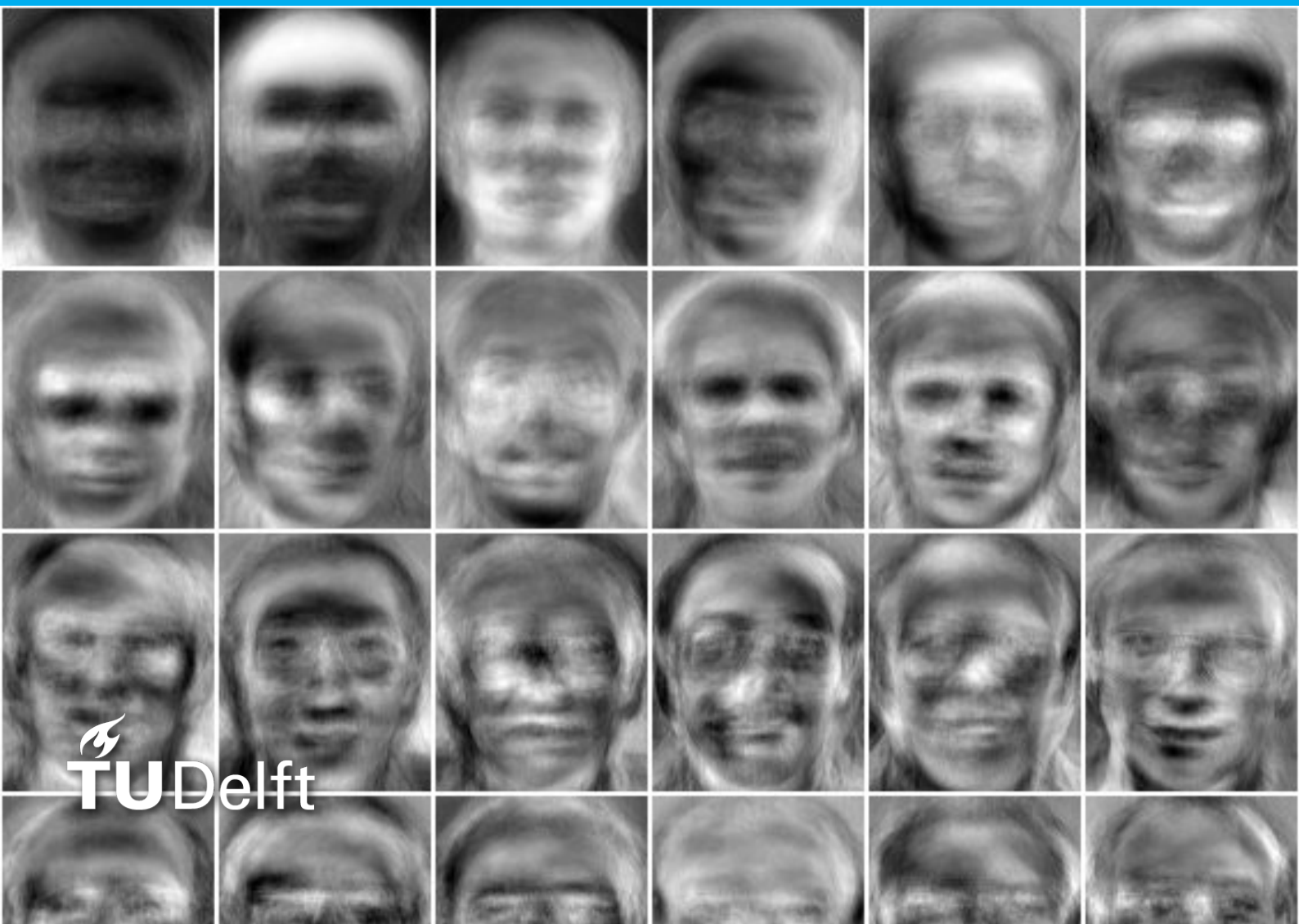
Login with Face and Speaker Recognition

TI3806 Bachelorproject - Final Report

Sebastiaan Brand, Tom Catshoek and Joost van Oorschot



TU Delft

Login with Face and Speaker Recognition

TI3806 Bachelorproject - Final Report

by

Sebastiaan Brand
Tom Catshoek
Joost van Oorschot

Client	David Tax	TU Delft
TU Coach	Thomas Abeel	TU Delft
Bachelor Project Coordinator	Martha Larson	TU Delft

An electronic version of this document is available at
`http://repository.tudelft.nl/`.

TU Delft

Preface

This report documents the findings of our bachelor thesis project, during which we developed face and speaker recognition modules which can be used to authenticate a user, along with a desktop application which acts as a mock login application. Because of some hiccups at the start we only had nine weeks to complete this project instead of ten, and we would like to thank our coach and our client for accommodating some last minute changes so that we could start the project without too much of a delay. We would also like to thank Ling Feng from the Technical University of Denmark for granting us access to the English language speech database for speaker recognition (ELSDSR), which we used for training and testing our speaker verification system.

Sebastiaan Brand
Tom Catshoek
Joost van Oorschot

Summary

We have created a system, at the request of our client, which can authenticate users by their face and their voice. For the authentication we have two subsystems: one for face recognition, and one for speaker recognition. Besides these modules we have a third subsystem which acts as a mock login application which allows us to test and demo the system.

For the face recognition module the most notable algorithms we looked at are eigenfaces, Fisherfaces, and local binary patterns (LBP). These three algorithms are implemented in the OpenCV library we eventually settled on using. Our tests showed that local binary patterns perform best, which is why we use this algorithm as the core of our face authentication module. Before we apply the LBP algorithm a number of preprocessing steps, like cropping, rotating, and illumination normalization, are taken to improve the performance. The final classification is done by a nearest neighbor algorithm. Aside from that we also check for blinking in order to prevent photos of a user from being authenticated.

For the speaker verification module we did not have any ready-made implementations which did the entire classification for us, but we did have some libraries which implemented certain sub-steps of the process. The speaker verification starts by preprocessing the input audio. It is divided into overlapping frames and voice activity detection is performed, after which Mel-frequency cepstral coefficients (MFCC) feature vectors are extracted. These MFCC feature vectors are used to train two Gaussian mixture models: a speaker and universal background model. The model is then able to classify unknown MFCC vectors by matching them to either the speaker or the background, based on the log-likelihood ratio between the two models. Our voice activity detection algorithm works well under conditions with relatively little noise. We expect performance under noisy conditions can be improved by using more sophisticated voice activity detection methods. The performance is generally good however, so while there is definitely some room for improvement the system is already very usable.

Both modules have been implemented and have been integrated into a mock login application. Using the application, users can register and review biometric data, and they can log in to a mock protected environment. Additional features of the application include a custom blink detection system to prevent attackers from circumventing the face verification by showing a photograph of a verified user and the ability to learn online, by adding facial images to the face verification model when a user is authenticated.

Contents

1 Introduction

Our client has requested us to build a login application which can authenticate a user using facial or speaker recognition, ideally a combination of both. Research has already been done on face and speaker recognition, and while some of this research was useful for us, this research is mainly geared towards the classification of multiple people, rather than the accurate authentication of a single one. Currently there are few existing applications which use facial recognition as an authentication method, and even fewer which combine this with speaker recognition. And most of the software which does exist is proprietary and closed source.

The problem which we will be focusing on is "Can we create a secure login application that uses face and speaker recognition as authentication?" The main focus here lies on creating a system which can reliably authenticate a user, and less on embedding this in a working login application which actually allows the user to log into their computer. A number of open source libraries exist which do parts of what we want, although there is no open source solution which solves our problem as a whole. In order to create our login application, we combine these existing libraries into our own system and complement or extend them where necessary.

In section 2 we discuss the findings of the research phase of the project, during which we accumulated most of the information necessary for us to develop this system. Section 3 shows the design and implementation of our system, while section 4 shows the performance of the face and speaker recognition, as well as comparisons between different algorithms and parameters for us to find out which work best. In section 5 we look at the system we made and evaluate whether we met our initial requirements and design goal and we evaluate our process. Finally, in section 6 we give an overall conclusion of the entire project, and in section 7 we give our recommendations for future work.

2

Research

This section is the result of the research phase of this project. In it, we define the problem that we are trying to solve, we compare possible solutions to the problem, and we explain what possible solutions seem best suited to solve the problem. The problem in question is "Can we create a secure login application that uses face and speaker recognition as authentication?"

In section 2.1 we define and analyze this problem. Then, in section 2.2 we look at how this problem is solved in existing applications. In section 2.3 we outline the design goals that we have for this project. From these design goals, we have created a list of requirements that is given in section 2.4. In section 2.5 we explain what development methodologies we use. After this, we look at algorithms and libraries for face and speaker recognition, in section 2.6 and 2.7 respectively. In section 2.8 we discuss the desktop application that we will build and we provide a rationale for the choice of a login manager to possibly integrate our application with. Finally, in section 2.9 we give the conclusion of this research report.

2.1. Problem Definition and Analysis

Our client has asked us to solve the following problem: can you build a login application that can identify and authenticate a user, by his face, his voice, or both. The application should consider the biometric information of the user, and determine whether that information matches with that of any user on the system. If there is a match, the user should be logged in to the matching account. The problem that we try to solve is: can we create this application so that it is user friendly, but still operates in a secure way.

In order to solve our client's problem, we will develop a login application that has a face recognition module and a speaker recognition module. If both modules assert that the biometric data of the unknown user that tries to login matches that of a known user, the user will be logged in. The face and speaker recognition modules will use machine learning to authenticate the user. We will discuss machine learning techniques for face and speaker recognition in sections 2.6 and 2.7.

The input data of our system consists of a picture, or a set of pictures, of the face of the user, and a sound clip of the user's voice. The system will compare the input with known values of users of the system. From this comparison follows the output of our system, which is boolean: a user is authenticated or not.

2.2. Existing Applications

There are few existing applications that are similar to the application that we are going to build for this project. We have only one application that uses both face and speaker

recognition as a method for authentication. The application is proprietary and developed by a company called KeyLemon. The version of the application with speaker recognition is a paid application, so we have not been able to test it.

There are also some applications that only use face recognition to authenticate users. One of the prime examples is the Android operating system, which offers face recognition as an authentication option to log into a device. Windows 10 offers a similar feature, but only to devices equipped with a special camera with an infrared sensor. On Linux, there is an open source project called 'pam-face-authentication'[6], but it has not been maintained since 2013.

It seems that there are very few applications that use speaker recognition as a method of authentication. The only application that we have been able to find during our research is the Android operating system, which provided authentication by recognizing the voice of a user that says 'OK Google'. One of the downsides of this application is that this does not only unlock the device, but also starts a voice search. This application is also proprietary and closed source.

2.3. Design Goals

In this section we outline the design goals that we have for this project. We will use these design goals to distill a list of requirements for the project and as a way to align our expectations with that of the client. Also, we will refer to these design goals during the design and implementation of the application.

Usability

One of the main benefits of a biometric login application over a traditional username-password login application is the ease of use of not having to enter a password. Because of this, usability is an important design goal for this project. The user does not need to memorize multiple, long passwords for each different application that requires login. Instead, the user is authenticated by his physical characteristics. The design goals we have for usability are related to two things: the performance, and the user interface design.

A decent performance of the login application required to offer a good user experience, i.e. not to have the user wait several minutes before being able to log in. Our application will identify and authenticate users within one minute.

Additionally, the user should be able to log in independent of slight changes in appearance like different hair style, glasses, and facial hair. This is to prevent the user from having to add new face pictures each time his appearance changes slightly.

Aside from that, the login application needs a clear, simple user interface so that users do not need to have any prior knowledge on how to use the application. On screen, the user is given instructions on what to do, for example to sit straight in front of the webcam, or to say a specific phrase. Ideally, the application will identify and authenticate a user without any keyboard and mouse user input by the user.

Security

Making a login application more user friendly comes with the risk of weakening the security of the application. Our application needs to reach a level of security, for which we have outlined the following design goals.

First of all, we aim to make the application secure by combining face recognition and speaker recognition techniques, which creates redundancy. The idea is that if one technique gives a false positive, the other technique will most likely not. Only if both face recognition and speaker recognition give a positive result, the user will be logged in.

Secondly, the system should have a fallback to password, a last resort in order to prevent the user getting locked out of his system. After a certain amount of failed tries at

authenticating the user, the application will show a username-password login screen. If the user correctly enters the username and password, the webcam footage and voice clip used for the failed authentication will be added to the database as recognized biometric data.

Maintainability

In order to keep our application easily extendable and maintainable in the future by ourselves or other developers, we have set maintainability as a design goal for this project. In order to achieve this we focus on three things: keeping the system modular, properly documenting the code, and testing.

Firstly, we want to create a modular application, so it is easy to change one module without having to change the other modules. For example, if we choose to change the speaker recognition module, this will have no effect on the face recognition module. This is essential for being able to quickly identify and fix bugs, and for extending the application once there is an established code base.

We also plan on documenting the function of all classes and methods, by inline comments. In a situation where it is not clear why certain code is used, we will write additional comments next to the snippet of code in question. The main reason for documenting the code is so that another developer could take over development in the future, so as a way to future proof our work. In the documentation we will also specify which external dependencies the project has, as well as on which systems the application runs.

Finally, it is important that we know that we did not break any existing functionality after we modify the application. Because of the non standard nature of our primary modules (face and speaker verification) we will not be doing a lot of unit testing. Instead we will focus on structured top-down testing from the interface to check if everything works as intended and to find out if anything breaks during integration.

2.4. Requirement Analysis

In this section we define the requirements that the project has. We have defined the requirements with the client, to ensure that the client will receive the end product that he has in mind, while keeping the project in proper scope.

We have classified the requirements as one of the following options: must have, should have, could have. Must haves are defined as requirements that must be met to in order to get a working product. Should haves are defined as requirements that should be met as discussed with the client, but these requirements are not necessary for a working product. Could haves are defined as requirements that will only be met if there is time available after implementing the must haves and the should haves.

Requirement	Classification
The user is able to login using face verification	Must have
There is a fallback to username-password login.	Must have
The user can register.	Must have
The user can look into the data that is used for login.	Must have
False positive rate is not above 5%.	Must have
A genuine user can log in within one minute, while an impostor cannot log in within five minutes.	Must have
The application has a modular software architecture.	Should have
Speaker verification	Should have
The user can be identified by face or voice.	Should have
Th user can be identified independent of hair style, glasses, and facial hair.	Should have
The system is integrated into a login manager.	Could have
The application has a clean user interface.	Could have
The application is extendable with additional biometric techniques.	Could have
The application is able to learn 'online', by adding successful authentication attempts to its data set.	Could have
The application has a mouth movement detection algorithm, that checks whether the mouth moves if the user speaks.	Could have

Table 2.1: Requirements

2.5. Development Methodology

Development Cycles

After the research phase, during which we have produced this report, we will begin an iterative development phase. During this phase, we will produce working prototypes weekly, as well as weekly presentations for our coach and additions to the final paper. Because of this iterative approach, we will be able to find out quickly which methods and algorithms do not give the desired results, rather than finding out at a late stage in the development.

Testing and Evaluation

Since we are creating an authentication system it is very important to properly test and evaluate the classification models we make. Because of the nature of the system we should use the false positive rate as a primary benchmark. A high true positive rate is less important, though for our application to function properly still something which needs to be taken into account. Because of the importance of having few false positives, we will focus on minimizing this metric during the evaluation of our models. For each model that we test, we will draw a Receiver Operating Characteristic (ROC) curve to visualize the model's performance and to be able to easily compare the performance of models.

To be able to train and test these models, we need training and testing data. Luckily there are freely available data sets for training speaker and face recognition models. We will use various data sets to test specific scenarios that our face detection module will encounter. For example, there are data sets that focus on lighting conditions, while others focus on different facial expressions. In table 2.2 we list data sets of facial images, and for what purpose we might use the data set.

Dataset	Features
FERET	1199 individuals, 14126 images total. Very extensive, but only a subset is usable for our purpose. Different facial expressions, lighting conditions, and orientations.
AR Face Database (Ohio State University)	126 individuals, over 4000 images total. Different facial expressions, lighting conditions and occlusions. Two sessions per person. Features (e.g glasses wearing, specific facial expression) are labeled.
AT&T - The Database of Faces	40 individuals, 10 images each. Different times, lighting conditions, facial expressions and facial details.
EURECOM Kinect Face Dataset	52 individuals, 9 images each. Different facial expressions.
Face Recognition Data, University of Essex	395 individuals, 20 images each.
Georgia Tech Face Database	50 individuals, 15 images each. Different facial expressions and head orientations.
Radboud face database	67 individuals, 8 images each. Different facial expressions.

Table 2.2: Face data sets

Data sets that can be used to train speaker recognition models are harder to find. We have gotten access to one data set, called the English Language Speech Database for Speaker Recognition (ELSDSR)[4]. The ELSDSR features sound clips from 22 different individuals.

2.6. Face Recognition

The first main submodule of our system is the face verification module. This module will have two roles: verification and recognition. Primarily, it needs to be able to verify that a certain input face belongs to a specific person it already knows. Minimizing false positives here is a priority, although ideally it should be able to verify a genuine user in no more than a few attempts. Besides that, it should be able to recognize an input face and match it to one of a few faces it knows. This is to allow a user to log in without selecting a specific account first.

Most methods we discuss here are primarily aimed at recognition, although they can be adapted to suit our verification needs as well.

2.6.1. Algorithms

Direct Template Matching

Possibly the simplest way to identify faces would be to use a pixel by pixel nearest neighbor classifier on images[15]. This method however, has a number of drawbacks[14]. Firstly, it has a difficult time matching images taken under different lighting conditions. To fix this you would need for the training data to contain all possible lighting conditions which could occur in the test data. Besides that, it is computationally expensive because of the length of the vectors you're comparing. Even "small" 100x100px images would result into vectors of length 10,000.

Direct template matching as described here isn't really used in real world applications of face recognition, and because there are plenty of better alternatives we do not plan to use it in our system, but we cover it here for the sake of completeness.

Eigenfaces
The so called eigenfaces method[25] is one of the most popular examples of methods used for recognizing faces. In short, eigenfaces uses principal component analysis (PCA) on a set of face images to reduce dimensionality and then uses a distance based classifier (e.g. k-nearest neighbor) for classification. To elaborate a bit more: the method starts off by representing a (grayscale) picture of a face as a vector with the same length as number of pixels. From a set of training faces a number of principal components (eigenvectors) is calculated. These eigenvectors are called eigenfaces. A given face can be "(re)constructed" by taking the the weighted sum of these eigenfaces. These weights are used to identify a face, so to recognize a specific face, the weights of an input face are compared to those of the known faces. Representing the faces only as the weights corresponding to a fixed set of eigenfaces solves the high dimensionality problem template matching has. The eigenfaces method's accuracy, however, still suffers from variations in lighting. This lighting issue can be partly solved by throwing out the first three principal components[14].

While eigenfaces might not be the most accurate compared to the other methods we discuss here, we would still like to involve it in the face recognition, because (with the right kind of pre-processing (i.e. normalization)) it can still be a useful contribution to a combined prediction[18].

Independent Component Analysis
Independent component analysis (ICA) is a generalization of PCA and tries to solve PCA's problem that it relies on pairwise relationships between pixels, and therefor fails to capture information from higher-order relationships (latent variables)[12]. Where PCA tries to find a feature space which maximizes variance, ICA tries to find a feature space which minimizes the statistical dependence between components[16].

ICA has a similar performance as the PCA based eigenfaces on images which are very alike. On more different images (e.g. taken on a different day, different facial expression) it has higher accuracy than PCA[12] but because it is not widely used for face recognition it might be difficult for us to find a library which implements it.

Linear Subspaces
The idea of linear subspaces rests on the basis that a face is three-dimensional and responds to light from different directions differently. Because of that, if you take 3 pictures of a face with linearly independent light source directions, you can use those 3 pictures to reconstruct any picture of that face regardless of the lighting[13].

While this method is very accurate and only requires 3 images in your learning set to be able to handle most lighting conditions, there are obvious drawbacks. It will still have issues with variables like different expressions, glasses, and hairstyles. Also, in our case, it is not really feasible to have the user take 3 photos under very specific lighting conditions, and therefor we do not plan to use this method.

Fisherfaces
The goal of the Fisherfaces method[14] is to create an algorithm which is insensitive to lighting variations, different facial expressions, etc., as opposed to the eigenfaces method. The Fisherfaces method is an implementation of Fisher linear discriminant analysis (LDA), used for face recognition. LDA takes PCA as a basis, that is, it wants to maximize the variance captured *between* classes. However, knowing that there are different classes (i.e. different people, each with a number of images) it aims to *minimize* the variance *within* each class[27].

The Fisherfaces method is more accurate than Eigenfaces when it comes to identifying faces with different lighting conditions and facial expressions[14], and seems like a good method for us to use.

Elastic Bunch Graph Matching
Elastic Bunch Graph Matching (EBGM) looks for specific points on the face (eyes, mouth, etc.), obtains data from those areas, and puts it all into a graph representation[28]. The length of the edges of the graph capture the facial structure, while the nodes contain information about the texture at that specific point.

There is not a lot of independent comparisons between EBGM and other algorithms, but performance does not seem to be a great improvement over PCA, and it still struggles with the same different-lighting problem PCA has[10]. EBGM should have less issues with face rotations than PCA[28], but for our use case that is not really necessary, and that is why we do not plan to use it in our system.

Local Binary Patterns
With the local binary patterns (LBP) method the image of a face is divided into a number of smaller segments. From each segment histograms are extracted, which are concatenated into a single histogram which represents the face[10]. A nearest neighbor classifier on the calculated histogram is used to recognize faces. An in-depth explanation of how LBP works can be found in appendix A.1.

LBP has superior performance over PCA, Bayesian classifiers and Elastic Bunch Graph Matching on every aspect (different lighting, facial expressions, a lot of time between photos)[10]. For LBP to work properly it does require a higher resolution image than the previously described holistic methods. In our case we are dependent on the resolution of the built in web cam. Although, the resolution of modern devices should be sufficient for this method, and we plan to include it in our system.

2.6.2. Libraries
In this section we will introduce and compare three open source face recognition libraries, and motivate our choice of one of these libraries. Our requirements for a face recognition library are that it is open source, actively maintained, and well documented. While there are many performant proprietary face recognition libraries and applications, there certainly is no wealth of good open source face recognition libraries. We have however found three libraries that meet our requirements.

The first potential library is OpenCV, which is a general purpose computer vision library, with a multitude of features. One of its features is face recognition, with built-in support for eigenfaces, Fisherfaces and local binary patterns.

The second potential library is OpenBR, which is a biometrics library, with a focus on face recognition. OpenBR is based on OpenCV, but it offers a broader selection of classification and pre-processing algorithms than OpenCV. A unique feature of OpenBR is its ability to define new algorithms, by pipelining 'transforms', which can be pre-processing steps and classification algorithms. This would allow us to easily build our own algorithm from built-in transforms, without having to implement it from the ground up. Out of the previously discussed algorithms OpenBR supports eigenfaces, Fisherfaces, and LBP. Aside from that it has its own Spectrally Sampled Structural Subspaces Features (4SF) algorithm[19], which makes use of a combination of these three algorithms.

The final potential library is OpenFace, which is a face recognition library based on classification with deep neural networks. OpenFace also offer pre-trained models, but we do not know how well these models will fit our data. Training our own deep neural network is not feasible, as it requires a lot of computing power.

For our project, OpenBR seems best suited. The ability to create new face recognition algorithms built up from built-in transforms, as well as the option to implement our own transforms, gives us the option to quickly build prototypes with different algorithms, which gives us the opportunity to find an algorithm that is best suited for our use case. OpenBR also gives us the option to visualize the face data at any point in the pipeline, giving us greater insight into the workings of the algorithm.

2.7. Speaker Verification

The main goal of this part of the project is to provide an easy to use speaker verification module.

Speaker verification is not to be confused with speaker identification. For speaker verification, only a one to one check is necessary to see if the speaker attempting to identify themselves is actually the person they are claiming to be. Identification is a one to many problem, where a speaker has to be identified out of many possible speakers.

There are multiple ways to build a speaker verification system. What exact methods are most suitable depends on a few factors. We assume cooperative speakers, and high quality speech recorded from the same channel every time. Therefore the main distinction is whether the verification is text dependent, or independent.

Text dependent verification only accepts if a certain word or phrase is said by the right person. Independent verification does not take into account what is said, and only depends on the voiceprint of said person.

For our project, we chose to implement verification instead of identification because we will only match the voice of the person who is attempting to log in to the voiceprint of the person they are claiming to be.

2.7.1. Algorithms

Research into speaker recognition methods is not as common and holistic as with face recognition. We will be focusing here on three type of methods which each solve one part in the speaker recognition problem: pre-processing, feature extraction, and classification.

Preprocessing

To make sure our model will not be biased by non-speech audio during training, we will filter our training data to exclude silent parts. This filtering can be done in several ways, which will be detailed here. What method works best depends on the data and how noisy it is.

The first method we look at here is energy based voice activity detection. Given that the dataset in use has a high signal-to-noise ratio, a simple energy based speech activity detector can be used to prevent non-speech audio from influencing the training. This detector will track the noise floor of the signal and discard any audio frame under a certain threshold.

A more robust method for speech activity detection is called long term spectral divergence, which is based on the long term spectral envelope of a signal[21]. This method discriminates noise and speech based on the fact that those have different spectral envelopes. Voice activity detection is not the main point of this project, so for more information we refer the reader to the previously mentioned paper.

Feature extraction

For our speaker verification system, we need to be able to extract a voiceprint from a given audio recording. In other words, we need to extract features from audio.

A widely used way to accomplish this is by using Mel-frequency cepstral coefficients (MFCCs). The Mel-frequency scale is based on the sensitivity of human hearing, so it is useful for capturing the phonetically interesting parts of human speech[11]. The process of calculating the MFCCs of a signal can be summarized as follows[23][9]:

1. Divide your signal into 20-40ms frames.
2. Apply discrete fourier transform to those frames
3. Apply Mel filterbank to the result of the DFT

4. Take the logarithm of the values in the filterbank
5. Take the discrete cosine transform of the log filter values
6. The results of the DCT are your MFCCs

The end result of this process is a set of feature vectors describing the power spectrum of the frames of the original signal.

As an extension of MFCC there are delta and delta-delta MFCC, which append differential and acceleration coefficients to the MFCC feature vector[1]. Because delta/delta-delta MFCC take into account the first and second order differences between the MFC Coefficients over time, additional useful information about the dynamics of the signal can be taken into account during classification. We expect this to result in an increase in accuracy. On the other hand, it might introduce a dependency on the temporal aspects of the signal, e.g. what word/phrase is spoken. Additional research is required.

Classification

Speaker classification can be divided into two different types: text dependent and text independent. Text independent methods do not rely on the content of what is spoken, instead they rely on the more general characteristics of a persons voice that can be used for verification. Text dependent speaker verification listens more closely to what is said by the user, in addition to the characteristics of the speakers voice. This generally results in higher accuracy[11]. Here we will be looking at a number of both text dependent, as well as independent classification algorithms.

First of all, Gaussian mixture models. A Gaussian mixture model (GMM) is a statistical model that assumes all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters[2]. It is similar to k-means clustering in how the Gaussians are assigned, but where k-means defines hard clusters, Gaussian mixture models can express a level of certainty about whether or not a certain data point belongs to a component. To create a Gaussian mixture model from a given set of data points, an algorithm called expectation maximization is generally used. There does not seem to be an optimal way of determining the amount of components to use in a GMM, so we will empirically evaluate what the optimal amount of components is in our case. As stated in [22], all recent top performing systems in the NIST Speaker Recognition Evaluations have been GMM-based. Or more specifically, GMM-UBM, where UBM stands for universal background model. A GMM-UBM based speaker verification system typically consists of two GMMs, one trained on the speaker to be verified, and the other one trained on a big dataset of a lot of different speakers. Using these two models, the likelihood ratio of whether or not a certain recording of speech belongs to the person the first GMM was trained on can be calculated by dividing the likelihood from GMM1 by the likelihood from GMM2. Then, if this ratio is above a specified threshold, the speaker is accepted.

Compared to Gaussian mixture models, hidden Markov models are far more suitable for text dependent speaker verification as they are able to take the temporal aspects of the input signal into account. This is because they work with hidden states, which have probabilistic transitions between them while each state has a different emission probability distribution. Training a hidden Markov model on a sequence of MFCC vectors using an EM training algorithm is possible, the training should find a good estimation of the required parameters. The optimal amount of hidden states will have to be investigated empirically. While it is easy to train a single model for a speaker, it is a difficult task to train a background model as that would require a big database of recordings of different speakers saying the same utterance. Working with a single model and implementing a probability threshold may be a more feasible option, although accuracy might suffer.

Finally, we could look at doing the classification with neural networks. We propose a relatively simple method to classify MFCC vectors into two classes: the correct speaker,

or all other speakers. To do this one could use a standard feed forward multilayer perceptron. The output layer would have two neurons, each outputting the 'confidence' a certain input vector belongs to the correct speaker, or someone else. Then, when the ratio of positively classified feature vectors versus negatively classified ones over a certain timeframe reaches a predetermined threshold, the system accepts. It would be interesting to compare this method to the standard GMM based one. A downside to this method is the need to train a network for every person added to the system.

2.7.2. Libraries

Seeing as MFCC are one of the more popular methods of extracting features from audio for the purpose of speech recognition and related tasks, there are a lot of libraries that implement it (see table 2.3). We tried libmfcc but found it lacking in functionality, so we settled for Aquila. Aquila is a fully featured C++ digital signal processing library which does a lot more than just calculate MFCCs. It implements several windowing functions, different methods for Fourier transforms, and pattern matching using dynamic time warping.

We found two libraries that implement Gaussian mixture models. This is certainly not an exhaustive list, but these are sufficient for our purposes. In fact, mlpack is built on top of armadillo. Armadillo is a high performance linear algebra library for C++. It is similar in syntax to Matlab. We found it to be very fast and easy to use. Mlpack is a C++ machine learning library built on top of armadillo. It implements a lot of useful functionality like Gaussian mixture models, hidden Markov models, and principal component analysis. As it is built on top of armadillo, it delivers the same great performance while abstracting away some of the lower level details.

algorithm	**library**
MFCC	libmfcc
	kaldi
	aubio
	aquila
GMM	armadillo
	mlpack
HMM	mlpack

Table 2.3: Speaker verification libraries

2.8. Application

This section will give a brief overview of the applications that we will develop, and the technologies that will be used to develop them. We will develop two applications: a desktop application that can be used to register and review biometric data, and a login application that authenticates users.

With the desktop application, the user will be able to register new biometric data and to review existing biometric data. The user can improve the face and speaker recognition by adding new photos and sound clips. To develop this application, we will use Qt[7]. The reason for this is twofold. Firstly, Qt will allow us to quickly prototype the application, as it decreases the amount of boilerplate code that is necessary. Secondly, we want to make an application that is easily portable to another platform, and Qt supports all major operating systems.

We also want to integrate the application into an actual login manager, also called a display manager. LightDM[5], a Linux dislay manager, seems best suited for this. LightDM supports custom 'greeters', or login screens, so we could develop a greeter that uses our face and speaker detection modules to authenticate the user. Another reason

for picking LightDM is that it is ubiquitous: it is used in major Linux distributions such as Ubuntu, Debian and OpenSuse.

2.9. Research Conclusion

For face recognition, we have found a number of suitable algorithms, most of which are supported by the OpenBR library. We will use OpenBR's default face recognition algorithm (4SF) to implement a simple prototype, which we will extend to fit the requirements.

For speaker verification, we have found a number of suitable methods to investigate. We will use Mel-Frequency Ceptrum Coefficients as audio feature vectors, and apply text-independent and -dependent classification methods. For text independent verification we will use Gaussian mixture models, and investigate the feasibility of a simple neural network. For text dependent verification we will use hidden Markov models.

We will build an application with which a user can register a profile and can add pictures and voice clips that will be used to authenticate the user. It is our goal to integrate this application into the LightDM display manager, so that it can be used as a login mechanism in Linux.

3

Design and Implementation

We have developed the face and speaker recognition application in an iterative fashion, starting with a simple prototype that only supported basic face recognition. While we had on initial idea of the requirements of the application, as described in multiple sections of chapter 2, many of the specifics were still unclear. In this section we discuss the process of the iterative development of the desktop application, the problems that we faced during the development, and how we solved the problems.

3.1. Choice of programming language

The programming language that we chose to develop the desktop application with is C++. The primary reason for choosing C++ is that both OpenBR and OpenCV as well as the libraries that we have used for speaker recognition have C++ APIs. Another reason is that C++ is multiplatform, which is a requirement from our client.

3.2. Face recognition library

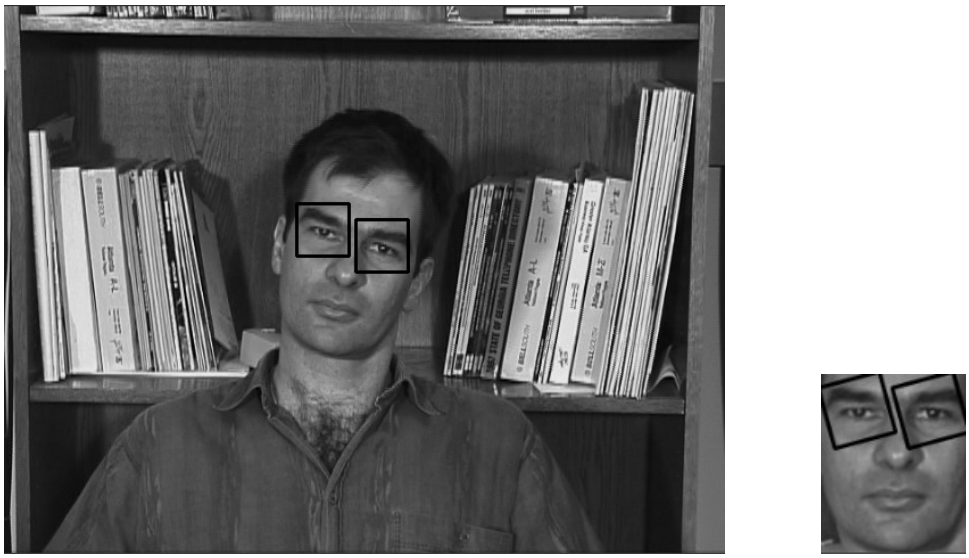
As creating our own implementation of a face recognition algorithm is not within the scope of this project, we use a library for this. As stated in the research conclusion, we set out with the goal to use OpenBR as the face recognition library for the prototype. However, we soon ran into trouble with using OpenBR. The OpenBR documentation focuses on using OpenBR as an application rather than as a library, which is how we wanted to use it. When we eventually got OpenBR working as a library, we ran into problems with OpenBR and our version of Qt. At this point we made the decision to step away from OpenBR for the time being and use OpenCV instead, to be able to finish our first prototype timely.

Compared to OpenBR, we got OpenCV working effortlessly. We trained an eigenface recognizer on the AT&T data set, with some of our own pictures added, after which the eigenface recognizer was able to correctly recognize our faces. The pictures that we added were manually converted from color to grayscale and cropped to only show the face. In order to use this in our application, we had to automate the process of adding and preprocessing pictures.

3.3. Image Preprocessing

The face recognition techniques that we use in our application need their input images to be preprocessed for good results. The main preprocessing steps that are required for good results are cropping to only show the face, converting the image to grayscale, and normalizing the location of the faces.

Cropping and Rotation
For the preprocessing of the webcam frames, we use a Python script from the OpenCV documentation[8] as reference, which we have ported to C++. The script takes the coordinates of the eyes as input, which we detect using cascade classifiers. Luckily, OpenCV provides a classifier for eyes, so detecting them is pretty straight forward. A technical description of OpenCV's cascade classifier can be found in appendix A.2. Figures 3.1a and 3.1b show the input and the output of the preprocessing script.



(a) Original (b) Processed

Figure 3.1: Example of the input and output of the preprocessing algorithm. The black rectangles show the locations of the eyes, as detected by a cascade classifier. These are normally not rendered on the output.

The script places the left eye at a given offset from the borders of the image, and rotates the image around the left eye so that the eyes are parallel. Then, the image is cropped to a specified size.

Illumination normalization
In early tests of our application, we found that the performance of our face recognition was very dependent on the illumination conditions. To improve the performance under varying illumination conditions, we changed the eigenfaces algorithm to the local binary (LBP) patterns algorithm, because LBP features are invariant to homogeneous illumination changes[24]. However, LBP features are still affected by illumination changes that affect only parts of the face. As such, our application still needed dedicated training images for basically every room in which it was used.

Ideally, our application requires users to register pictures of their face only once, so we wanted to improve the performance of our face recognition under various lighting conditions. We found a method of illumination normalization that was developed specifically for use with LBP, called TanTriggs Preprocessing[24].

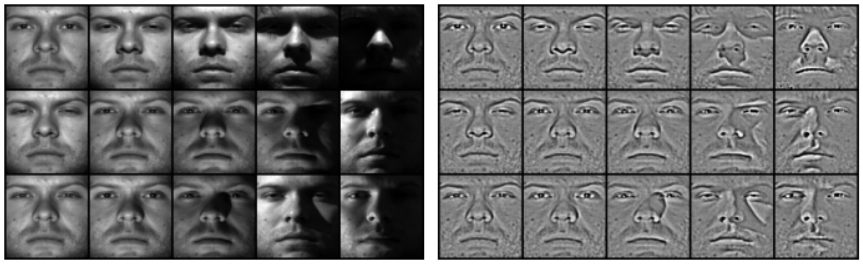


Figure 3.2: Input and output of the TranTriggs Preprocessing algorithm[24]

This preprocessing algorithm consists of three steps. Firstly, a simple gamma correction is applied to the image, which enhances the dynamic range in dark areas, while suppressing it in light areas. Secondly, difference of gaussians filtering is applied, which reduces local contrast in shadowed regions. The final step of the algorithm is contrast equalization, which increases the contrast in the image overall. The combination of these three steps leads to an image that is very consistent under various lighting conditions. Tests on the data set of Face Recognition Grand Challenge 1.0.4 experiment done by Tan and Triggs show that the recognition rate went from 41.6% with only LBP to 79.0% with TranTriggs Preprocessing and LBP.

Top half of face
Finally, we looked at cropping out and only using the top half of the face (figure 3.3) in order to get rid of influences of facial hair and different mouth expressions. While throwing out the entire lower half of the face is sure to lower dependency on facial hair, the negative impact on overall performance (see section 4.1) was for us a reason not to use it in the final implementation.

(a) Whole face (b) Cropped half

Figure 3.3: Cropping half the face loses dependency on facial hair and mouth expression

3.4. The Classification Model

For the classification of the preprocessed images we settled on OpenCV's local binary patterns algorithm (LBP), because out of the models OpenCV offers it showed the best performance on our tests (see section 4.1). A technical description of the algorithm is given in appendix A.1.

For the prototype of the desktop application, we used the AT&T data set train the base model. With the prototype, the user can take additional photos of himself and retrain the model with the images included.

In a later version of the application we added the possibility of selecting a different data set to train the model on. In the final version of the application we have trained the model on a subset of images from the FERET data set. This subset contains only images of individuals directly facing the camera, as they would using our application. We chose to use the FERET data set over the AT&T data set, because the FERET data set consists images that are not cropped, unlike images from the AT&T data set. This is beneficial

because it allows us to crop the images using the same algorithm that is used in our application, which prevents our model from classifying images based on their cropping.

3.5. Authentication

The way we use our face and voice recognition systems to authenticate a user is an important part of the security of the application. Here we give a general overview of our authentication strategy. For the way we determine where to set the authentication thresholds see section 3.8.

We start with face recognition and, after we have detected a number of blinks, try to match the input image to one of the known images (figure 3.4). If the confidence is higher than a certain threshold the user is logged in directly. If the confidence is not high enough, we lower the threshold for the face recognition, but also request voice. We then compare the input voice to the known voice belonging to the user the face was matched to, and if both the face and voice fall within an accepted threshold the user is logged in. If either the face or the voice verification isn't confident enough, the user gets 14 new tries. If all of those fail, we fall back to a password login.

In our authentication strategy, the face verification algorithm only needs verify one out of fifteen faces. We have chosen for this strategy, because we set the threshold of the face verification algorithm to such a confidence that we are very certain that the user is a genuine user.

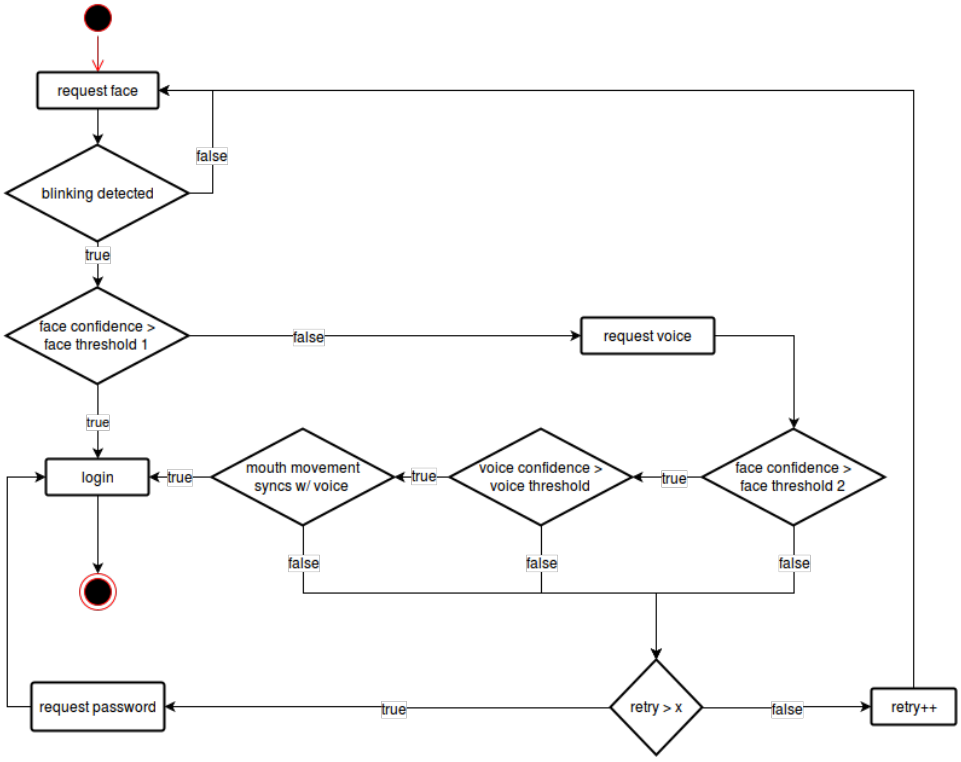


Figure 3.4: Authentication strategy flowchart

With the model which is trained on pictures of the user, the user can be authenticated by the application. Figure 3.5 show how the authentication works in the application.

The model gives a label and a distance. The label determines as which individual the user is classified. If the label is the label of the genuine user, and if the distance is below a certain threshold, the user is authenticated. How we determine this threshold is discussed in section 3.8.

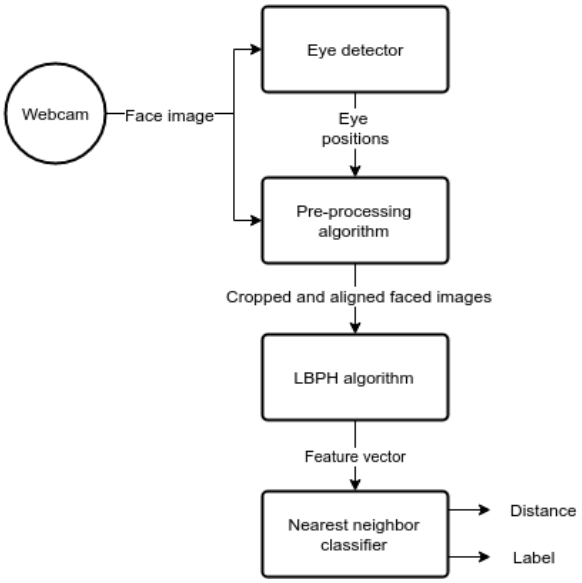
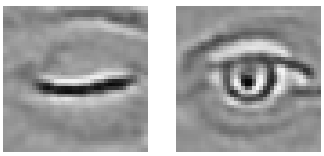


Figure 3.5: Authentication in the prototype

3.6. Blink and Mouth Movement Detection

Blink Detection
In order to prevent someone being able to log in as a certain user by holding a picture in front of the camera, we want to be able to detect if the user blinks. In order to do this we start by capturing images from the camera, in which we detect and crop out the eyes (see figure 3.6). The eye detection happens with the same classifier we use to find the eyes in order to rotate the faces in section 3.3. Also, in order to minimize the effect of different lighting conditions during the classification, the captured eyes are ran through the TranTriggs Preprocessing algorithm[24] which we also use on the faces.



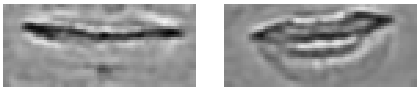
(a) Blink (b) Open eye

Figure 3.6: Eye images captured and preprocessed by the application

The problem that we are left with is the ability to determine whether a captured eye is open or closed. This binary classification problem is much simpler than our facial recognition problem and there are many classifiers which could solve it. We, however, already have three classifiers at hand in the OpenCV library we are using: eigenfaces, Fisherfaces, and local binary patterns. Because this is a binary classification problem with a relatively clear differences between the classes, a PCA based classifier like eigenfaces with a low number of components seems like a good candidate. For the evaluation of the performance and the choice of parameters see section 4.2. For the best performance the classifier needs to be trained on eyes of the user.

Mouth Movement Detection
Besides blink detection, we also check if the user's mouth moves at roughly the same time the speech is detected. The process here is nearly exactly the same as for the eyes:

detect, crop, and preprocess the mouth (figure 3.7), and match it to either an 'open' or a 'closed' class. Only the exact classifier (section 4.2) is a somewhat different because both are optimized do do their own thing. And, unlike with blinking, the mouth movement must also be matched to the moment the speech is detected. Note that we just look whether the mouth is moving, we do not lipread.



(a) Closed mouth (b) Open mouth

Figure 3.7: Mouth images captured and preprocessed by the application

Face Stillness Detection
While testing we found that during (sudden) movements of the face the classifiers we use to locate the eyes and the mouth have difficulty keeping track. During the face recognition this is not really a problem because an unrecognizable blur doesn't have much more of a chance of being mistaken for a genuine user than an impostor. In the case of blink and mouth movement recognition however, sudden movements introduce undesired false positives. This happens because the blink and mouth movement classifiers assume images of eyes and mouths as input. During movement something which isn't actually an eye might get recognized as one anyway, and the classifier then tries to determine whether the "eye" is open or closed. Training a third "other" class or increasing the required confidence during the decision making could take away these false positives, but would also take away from our true positives. Instead, we check if the face has been moving more than a certain threshold in the last couple of frames, simply by looking at the positions the face cascade classifier locates the face at. When there is too much movement, the blink and mouth movement analyzers don't return open or closed, but indicate that the image is not still enough.

3.7. Speaker verification

Our initial prototype of the speaker verification module consists of a simple audio processing pipeline which takes in raw audio pcm data and outputs MFCC feature vectors. These vectors are then used to train a model consisting of two GMMs, one with MFCC vectors for the speaker, and the other one for the background model. The model can then take in other MFCC vectors and classify them between speaker and background depending on the log-likelihood ratio between the two GMMs.

3.7.1. Preprocessing

In order to train our model, we first have to preprocess our audio data in order to divide it into frames and filter out unwanted noise and silent parts. We then use these audio frames to calculate MFCC feature vectors which are used to train the GMMs. A high level overview of the system can be seen below:

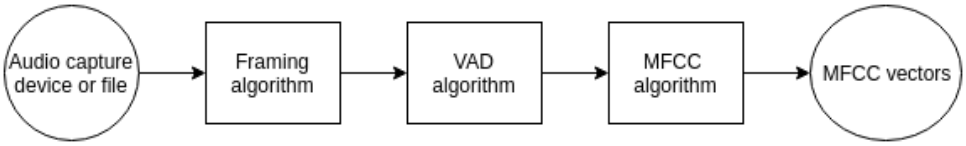


Figure 3.8: MFCC pipeline

Framing
The first part of the pipeline divides the input signal into (partially overlapping) frames. We have found frames between 20-40ms in length with a 50% overlap to work adequately. Further evaluation is needed to find optimal parameters. We need to divide the data into frames because we need feature vectors over small parts of the speech signal. Without framing this pipeline would only output one feature vector for the entire input signal, which obviously would not capture any significant amount of usable information.

Voice activity detection
The second part of the pipeline performs voice activity detection. In the initial prototype we used an energy based VAD algorithm. To initialize this algorithm, we feed it a small part of recorded speech. The algorithm calculates the average energy E of this signal X consisting of n samples as follows:

$$E_X = \frac{1}{n}\sum_{i=1}^{n}|x_i|$$

A threshold θ with $0 < \theta < 1$ is set after which audio frames can be processed by the algorithm and discarded based on the threshold and average energy of the frame F.

$$E_F \begin{cases} > \theta * E_X & \text{keep} \\ \leq \theta * E_X & \text{discard} \end{cases}$$

We have found a threshold of 0.3 to work well for training our model on the ELSDSR[4] database. What threshold works well depends on the data the algorithm is initialized with and how noisy the input data is. In section 4.3 we show how we come to the selected threshold.

Calculating MFCC vectors
As implementing our own MFCC algorithm is outside of the scope of this project, we decided to use Aquila[3], a DSP library which provides a class to calculate MFCC vectors. We have wrapped it in a helper class which takes in a set of audio frames, calculates the MFCC vector for every frame, and returns those.

3.7.2. Gaussian mixture models
Our models consists of two gaussian mixture models, one speaker and one universal background model (UBM). Once the two models are trained, we use their evaluation results to set a threshold based on the maximum allowable level of false positives. Implementing our own GMM, as well as training algorithm is outside the scope of this project, so we use the implementation provided by mlpack[17].

Training
After the training audio has been preprocessed, we use the resulting MFCC vectors to train a GMM using the expectation maximization algorithm. This algorithm attempts to maximize the likelihood of the model parameters given the input data. There are two parameters that can be manually set, the dimensionality and the amount of components of the GMM. As our prototype MFCC algorithm does not implement delta or delta-delta MFCCs, we set the dimensionality to 12, which is the amount of features in the MFCC vectors. Our model seems to function best with one or two components. This is likely a result of overfitting due to insufficient training data. We refer the reader to the evaluation chapter for more information.

Putting it all together
Once the two models are trained and the desired threshold is set, we can use them to classify audio frames as either speaker or impostor using the log-likelihood ratio between the two GMMs. Let the probability $p(F|\text{GMM}_{sp})$ be the likelihood that audio frame F belongs to the speaker model, and $p(F|\text{GMM}_{ubm})$ be the likelihood that F belongs to the background model. The log-likelihood ratio between the two models can then be calculated by

$$\Lambda(F) = log(p(F|\text{GMM}_{sp})) - log(p(F|\text{GMM}_{ubm}))$$

The decision rule based on threshold θ is:

$$\Lambda(F) \begin{cases} > \theta & F \text{ is from speaker} \\ < \theta & F \text{ is not from speaker} \end{cases}$$

With this we have a working, but basic speaker verification model. The VAD algorithm could be replaced by a more sophisticated one, for example a long term spectral divergence based one, or a model much like the model in question here, but instead trained to be able to discern background noise from human speech.

3.8. Determining Thresholds

In order to combine various biometric authentication methods, we want all methods to produce a uniform output. We have chosen to use the posterior probability for this. We can estimate the posterior probability of a user being an impostor from the output of the face and speaker verification algorithms. We can then combine these probabilities to decide if the user is authenticated. For example, we can determine that we don't want the posterior probability of a user being an impostor to be higher than 5%, and calculate the distance associated with that probability and use that distance as a threshold. Also, by using the posterior probability of a user being a genuine user, we can tell how certain our system is of its classification.

We come to the posterior probability by first calculating $P(D|I)$ and $P(D|G)$ using kernel density estimation, where D = distance, I = impostor, G = genuine. We set the prior probabilities $P(I)$ and $P(G)$ to 0.5. Then we calculate the posterior probability as follows:

$$P(I|D) = \frac{P(D|I) \times P(I)}{P(D|I) \times P(I) + P(D|G) \times P(G)} \tag{3.1}$$

With this we obtain the probability distribution of the chance that the user is an impostor for a given distance or ratio. Now we can obtain a proper threshold by retrieving the distance that is associated with for example a 1% or 5% chance that the user is an impostor.

An example of the posterior probability distribution of the face verification module is given in figure 3.9. In this example, the threshold for $P(I|D) = 0.05$ would be a distance just under 45.

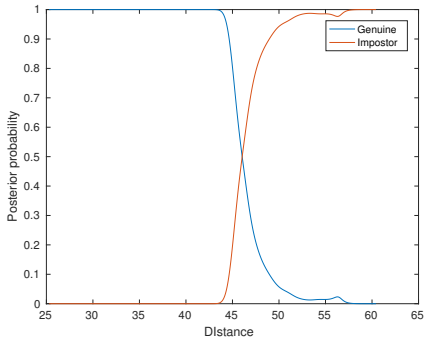
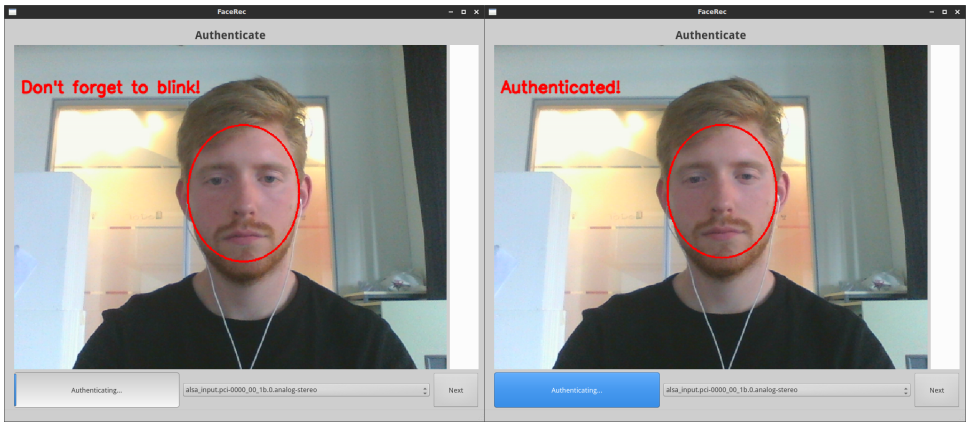


Figure 3.9: Posterior probability distribution for the face verification module

3.9. Application Run-through

In this section we describe a run-through of the application, for a successful and a failed authentication, to illustrate the working of the final product. We also show how a user can register new facial images to improve the face verification.

The first time the user launches the application, he can add facial images, after which the face verification model is trained on the images of the user. The next time the user starts the applications, he will see the authentication screen, as shown in figure 3.10a. This screen has background processes that try to authenticate the user. The user is reminded to blink, so that he can be verified by the blink detector.



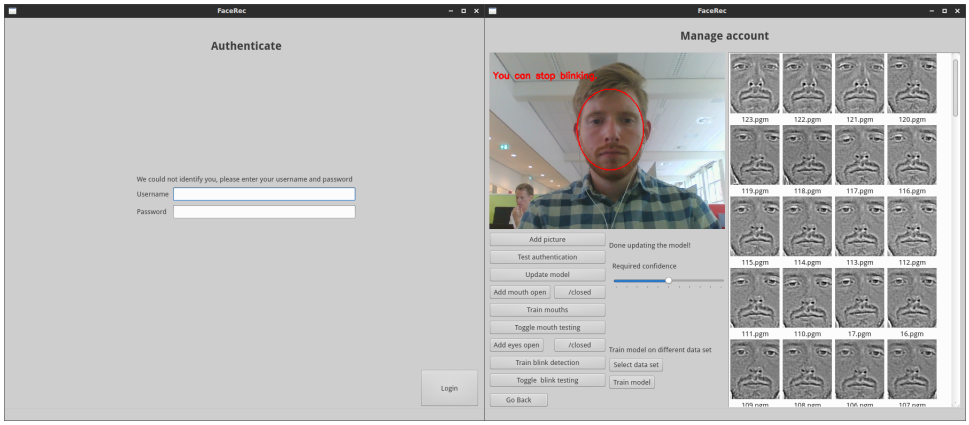
(a) The user has just launched the application.

(b) The user is authenticated.

Figure 3.10

If the blink detector has verified that the user has blinked, the user is notified that he can stop blinking. If the user is subsequently authenticated, he will be shown the screen in figure 3.10b.

If the user can not be authenticated by biometrics, the application will fall back to username-password authentication, as shown in figure 3.11a.



(a) The user has to enter his username and password. (b) The account management screen.

Figure 3.11

If the user is eventually authenticated by facial and voice verification, or by username-password login, the user can access the protected environment. From here, the user can also manage his account, as shown in figure 3.11b. On the account management screen, the user can add new facial images and images for the blink and mouth movement detectors. The user can also set the required confidence for the face verification model, and train the face verification model on a completely different data set. Once the user is done in the protected environment, he can log out.

4

Testing

Because our system relies largely on machine learning algorithms, our focus lies primarily on performance testing and evaluation. In this chapter we show the results of the evaluation of both our face and speaker verification algorithms, along with a visualization of our speaker verification dataset.

4.1. Face Recognition Performance

We have evaluated the three face recognition algorithms that are available in OpenCV, the computer vision library that we use for this project. In this section we discuss the evaluation of the local binary patterns algorithm we use in the application. The evaluation of eigenfaces and Fisherfaces algorithms, which both performed worse than LBP, can be found in appendix B. The difference between these three algorithms lies within the way they do feature selection/reduction, as they all use a nearest neighbor algorithm for classification.

We use scatter plots to get a general idea of the performance of the algorithm, by looking at the distribution of the distance of correctly identified users and impostors. A favorable algorithm has a little overlap between these two distributions. Another way we test performance is by looking at ROC curves and the area under the curve (AUC). When we started testing, we used the complete curve and AUC. However, because we are only interested in a fairly low false positive rate, we decided to only use the part of the ROC curve under a 5% false positive rate. Consequently we use the partial AUC up to a 5% false positive rate, scaled as $\text{AUC}_{0.05} = x$ so that x is 1 if the partial AUC is 0.05.

OpenCV's Local Binary Patterns Algorithm

For the evaluation of the local binary patterns algorithm we ran 25 times repeated random sub-sampling on the AT&T dataset. The results are shown in figures 4.2 and 4.3. With a training set of both 20 and 3 people, LBP outperforms Fisherfaces. With a training set of 3 people, LBP clearly outperforms eigenfaces, and with a training set of 20 people LBP edged out eigenfaces.

LBP has four parameters; radius, number of neighbors, and the width and height of the window. The default parameters in OpenCV are $(r = 1, n = 8, w_x = 8, w_y = 8)$. Figure 4.1 shows the effect of the radius on the average distance between genuine correct images and impostors. From this we can take that a radius of 4 would be a good option, as a higher radius has no positive effect on the average distance. Further evaluation showed that a radius of 4 did improve accuracy slightly. To quantify this, the partial AUC has gone from 0.766 with the default parameters, to 0.788 with $r = 4$.

During our evaluation, a higher neighbor amount (12) with radius 2 and default window size achieved a partial AUC of 0.720, thus lower than with a default number

of neighbors, but it did increase computation times significantly. If we increase the window size to 16 by 16, the accuracy decreases further: the partial AUC was 0.674.

We decide to go with parameters ($r = 4, n = 8, w_x = 8, w_y = 8$). A radius of 4 slightly outperforms the default radius, and a non-default amount of neighbors and window size does not seem to have a positive effect on the accuracy of the algorithm.

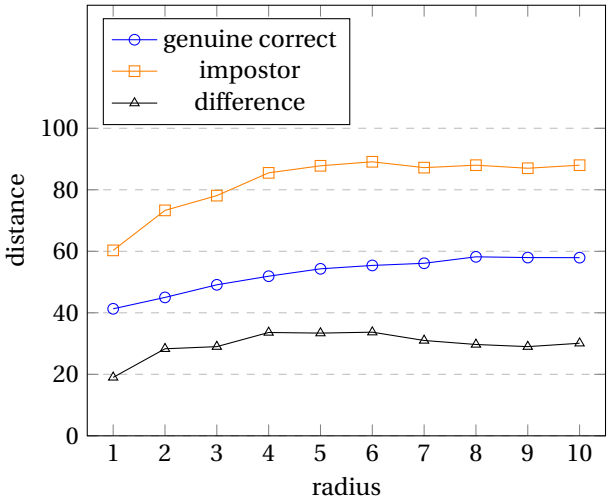
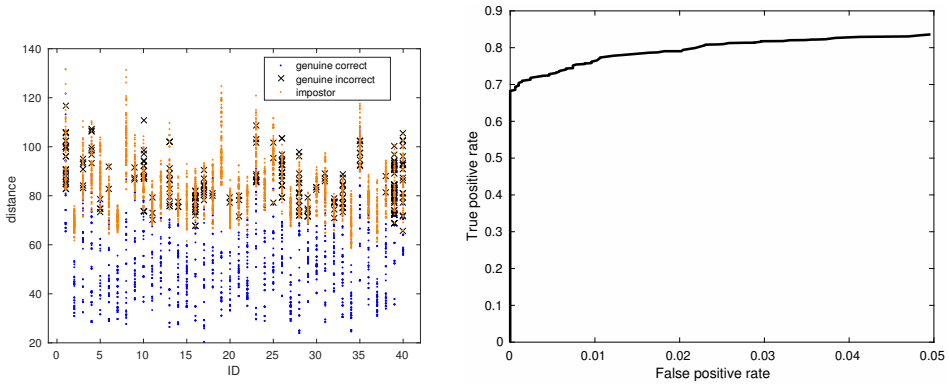


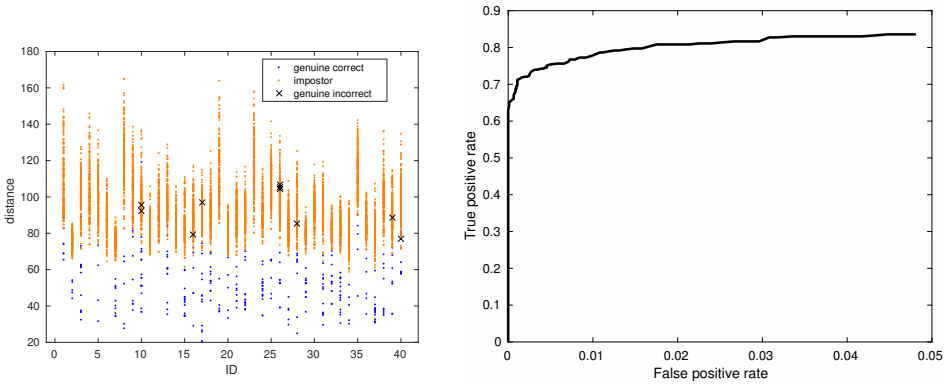
Figure 4.1: Average distances of matches for different radii



(a) Scatter plot

(b) ROC curve, $AUC_{0.05} = 0.788$

Figure 4.2: Genuine vs impostor matches on a training set of 20 people, 5 pictures each, from the AT&T dataset with OpenCV's LBP algorithm, with parameters ($r = 4, n = 8, w_x = 8, w_y = 8$).

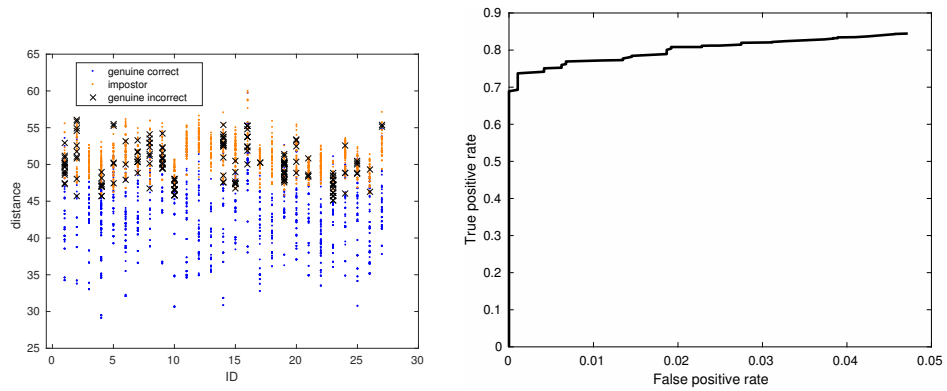


(a) Scatter plot (b) ROC curve, $AUC_{0.05} = 0.772$

Figure 4.3: Genuine vs impostor matches on a training set of 3 people, 5 pictures each, from the AT&T dataset with OpenCV's LBP algorithm, with parameters ($r = 4, n = 8, w_x = 8, w_y = 8$).

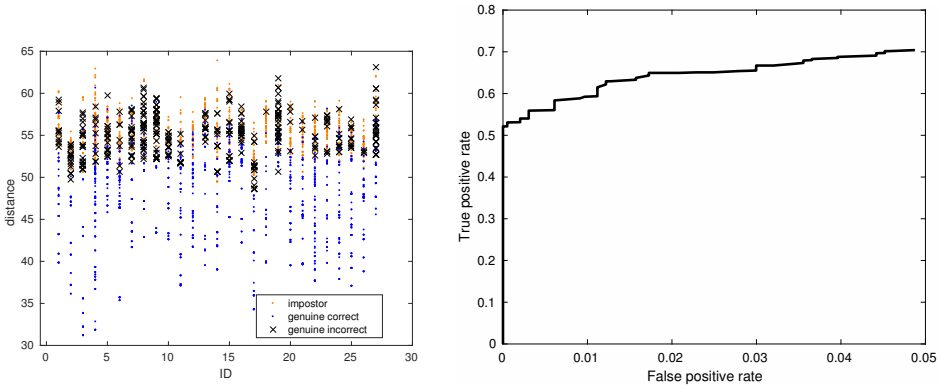
Half vs. Whole Faces

For the evaluation of the performance difference between using the entire face or only the top half we use the part out of the FERET dataset we use to train our model. Two versions are used: one with the whole faces, and one with the cropped out top halves. Similarly to previous tests we used repeated random sub-sampling to divide the data into a training and a test set a number of times. The results can be found in figure 4.4 for the whole faces, and figure 4.5 for the cropped faces. While the classification is still quite usable with partial $AUC_{0.05}$ of 0.626, there is a substantial drop in performance compared to using the entire face, where the $AUC_{0.05}$ is 0.756. The false positive rate also stays 0 a lot longer when using the whole face compared to using the crop (up until a true positive rate of around 0.7 compared to 0.5). Because of this difference in performance we have decided to accept the performance loss which comes with different facial hair and use the whole face for its better overall performance.



(a) Scatter plot (b) ROC curve, $AUC_{0.05} = 0.756$

Figure 4.4: Performance of LBP on whole faces



(a) Scatter plot

(b) ROC curve, $AUC_{0.05} = 0.626$

Figure 4.5: Performance of LBP on the cropped out top halves of faces

4.2. Blink and Mouth Movement Detection Performance

For the evaluation of of our blink and mouth movement detection we would like to use a dataset of mouths and eyes, labeled whether they are open or closed. A dataset like this isn't readily available so made one ourselves using the same function which captures the eyes in the application. An advantage here is that the crop and pre-processing of these test images is now exactly the same as those of the ones used in the application. The dataset we use consist out of roughly 100 images for each class: closed eyes, open eyes, closed mouths, and open mouths. The dataset is split 50/50 into training and evaluation subsets, with 100 times repeated random sub-sampling to average out performance of individual runs.

Figure 4.6 shows the performance of OpenCV's eigenfaces and Fisherfaces algorithms on our blink test set. The false positive rate here is the fraction of open eyes which are classified as blinks. The true positive rate (the faction of blinks which are correctly classified as blinks) hovers around 45% for all three of OpenCV's classifiers and isn't strongly dependent on the amount of components.Therefore we could say that the eigenfaces algorithms performs best when using 7 principal components. Because we are only comparing two classes, the fact that the performance of Fisherfaces (being based on LDA) doesn't change with a different number of components, was to be expected. Local binary patterns with default parameters performed significantly worse than either, with a false positive rate around 4.8%.

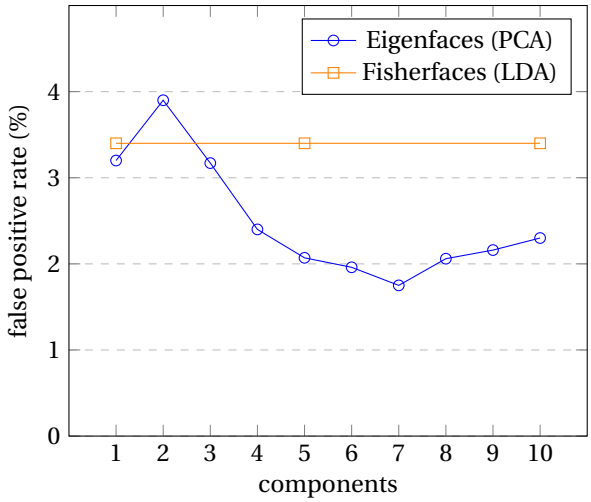


Figure 4.6: Blink detection performance with different algorithms

Compared to the blink classification, the classification of open and closed mouths (figure 4.7) goes a lot worse when using a low number of components, possibly because the differences between open and closed mouths are more subtle than with eyes (figures 3.6 and 3.7). Also, where the blink classifier shows a minimum error at some point, the mouth movement detection seems to converge to a false positive rate just above 2% when increasing the number of components. For the mouth classifier the eigenfaces algorithm using around 30 components seems like a good choice, because after that performance hardly increases. And just like with the eyes the true positive rate didn't vary strongly, and local binary patterns didn't perform better than the results we got with eigenfaces.

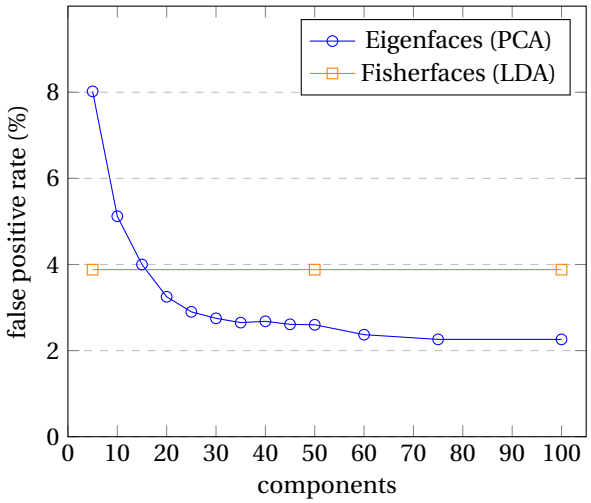


Figure 4.7: Mouth movement detection performance with different algorithms

4.3. Speaker Verification Performance

In this section we evaluate the performance of our speaker verification model on the ELSDSR database. Unless stated otherwise, ROC curves take into account results from all models, one for each speaker. This is done to even out performance between speakers, as our model performs very well on some speakers, and a bit less on others. This is probably because some speakers in the database are very similar to each other, while others are far more distinct.

Energy based VAD

To evaluate our energy based VAD implementation, we vary the threshold and plot an ROC curve of our speaker verification model trained on every single speaker in the dataset. The results of this experiment can be seen in fig. 4.8. To clarify, the VAD threshold determines what the minimum energy is for an audio frame to be accepted. For more information on how this calculation works please refer to section 3.7.1.

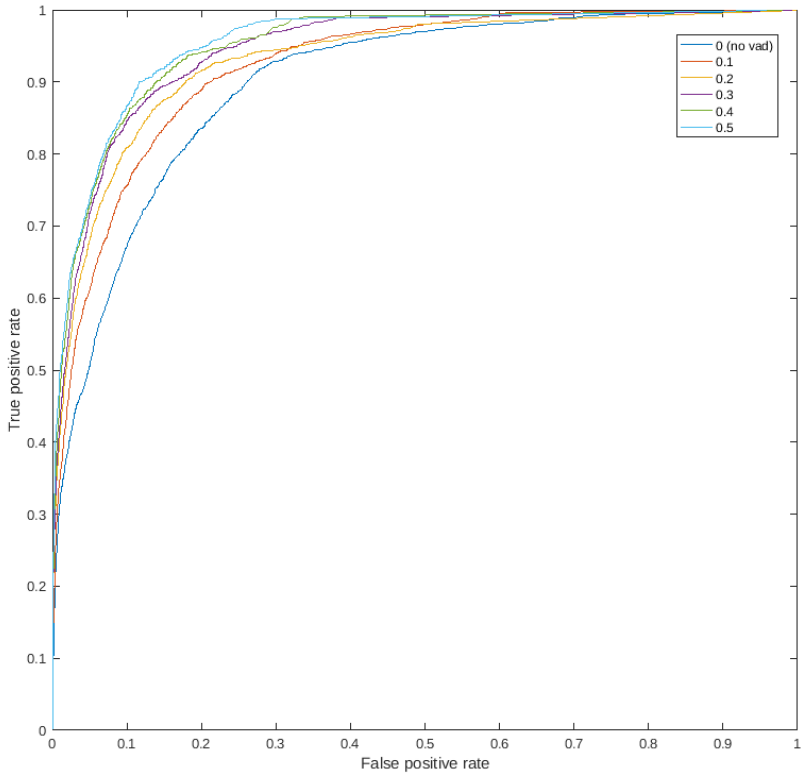


Figure 4.8: Varying VAD thresholds

VAD Threshold	AUC
0 (no vad)	0.894
0.1	0.920
0.2	0.930
0.3	0.938
0.4	0.931
0.5	0.913

As can be seen in fig. 4.8, increasing the VAD threshold seems to have a positive effect on classification performance, especially on the model's ability to reject false positives. This is as expected, as more background noise and silent frames are filtered out the higher the threshold is. However, one can not keep increasing the threshold as eventually there would be no voice frames left to classify. For our use, too many frames are dropped when the threshold is set above 0.3. And while it might not be obvious from fig. 4.8, fig. 4.9 also shows a decrease in the AUC as soon as the threshold gets above 0.3.

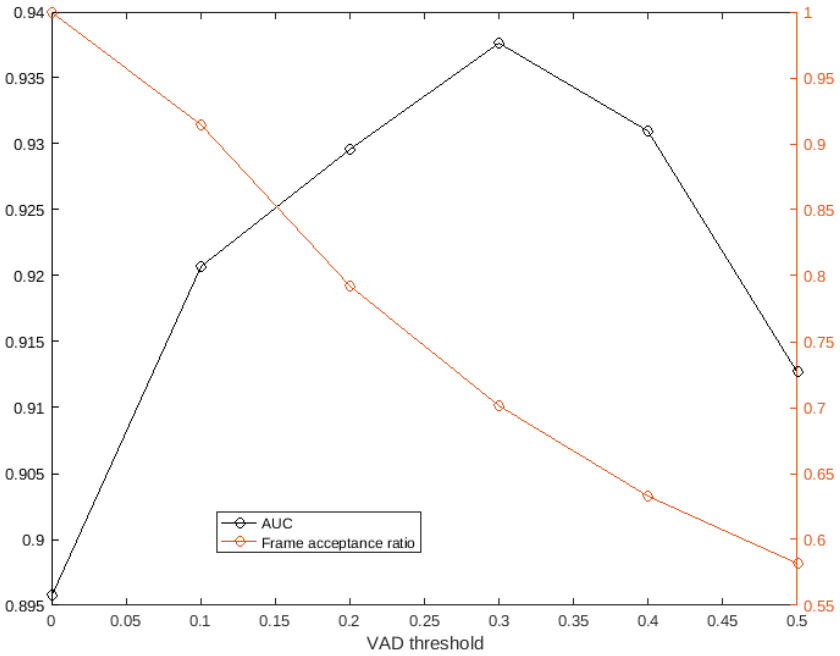


Figure 4.9: Varying VAD thresholds

Prototype GMM-UBM

In this section we evaluate the core functionality of our GMM based speaker verification model, which is calculating the probability a certain fragment of speech belongs to the speaker it is claimed to be from. With the VAD threshold set to 0.3, we will vary the amount of components in our GMMs to see how that affects performance.

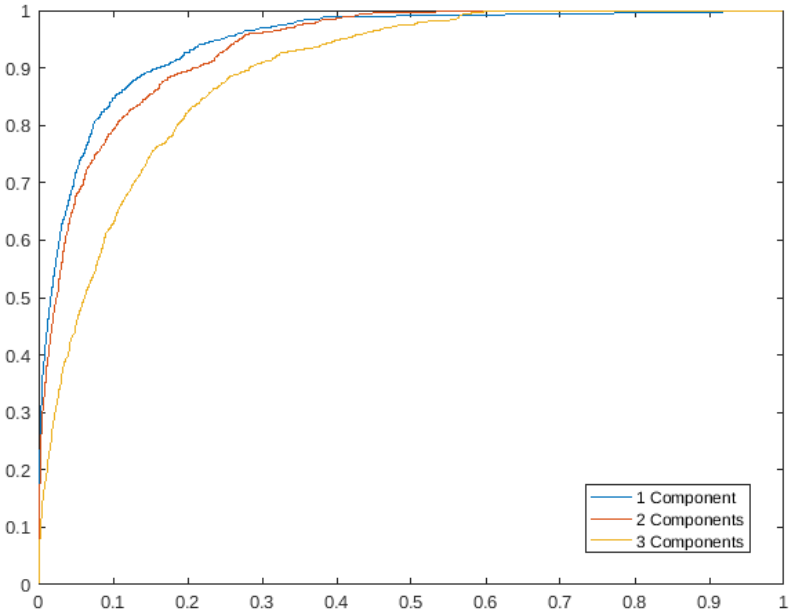


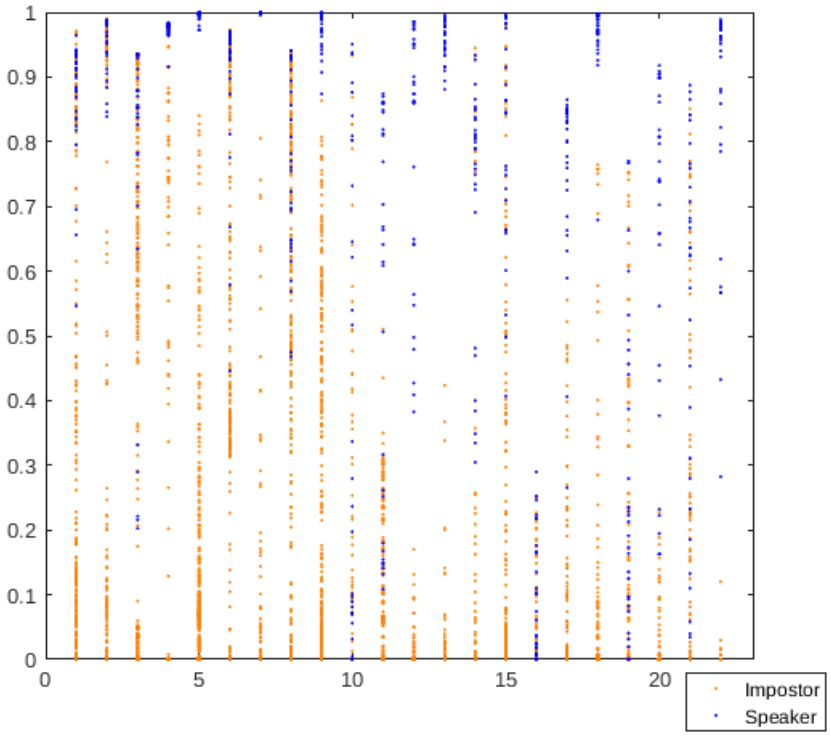
Figure 4.10: Varying amount of components

Amt components	AUC
1	0.938
2	0.923
3	0.888

It seems that performance only degrades with increasing the number of components. This is likely due to the fact that we only have a relatively small dataset available, and the GMMs are using a full covariance matrix. This causes the model to overfit as soon as more than one component is used.

Dataset evaluation

In this section we visualize the data we use for speaker verification in a few ways. We will use kernel density estimations to visualize the distribution of the log-likelihood ratio output of our model, and attempt to use dimensionality reduction techniques to visualize our feature space. First, here is a scatter plot showing the probability our model thinks that a certain data point is a speaker, for both impostor and genuine speaker samples.



#	Speaker
1	FAML
2	FDHH
3	FEAB
4	FHRO
5	FJAZ
6	FMEL
7	FMEV
8	FSLJ
9	FTEJ
10	FUAN
11	MASM
12	MCBR
13	MFKC
14	MKBP
15	MLKH
16	MMLP
17	MMNA
18	MNHP
19	MOEW
20	MPRA
21	MREM
22	MTLS

Figure 4.11: Scatter plot of genuines and impostors vs posterior probability

As can be seen in fig. 4.11, our model is able to confidently distinguish most of the speakers from their corresponding impostors. However, there are some speakers for which our model does not perform adequately, notably for #10, #16 and #19. On the other hand, our model is able to achieve near perfect classification on #5, #7, #18 and #22. If we visualize the posterior probability distributions of #7 and #16 (fig. 4.12 and 4.13 respectively) we can clearly see the difference. As could be expected, there is a much greater overlap between the genuine and impostor KDEs for #16 than for #7. This means the models ability to make confident predictions is diminished, which supports our findings for these speakers. Please note, for visualization purposes these KDEs are made on test data, not training data. It is possible that this discrepancy in performance is caused by insufficient test data, as the ELSDSR database only includes a small amount of test audio for some of the speakers.

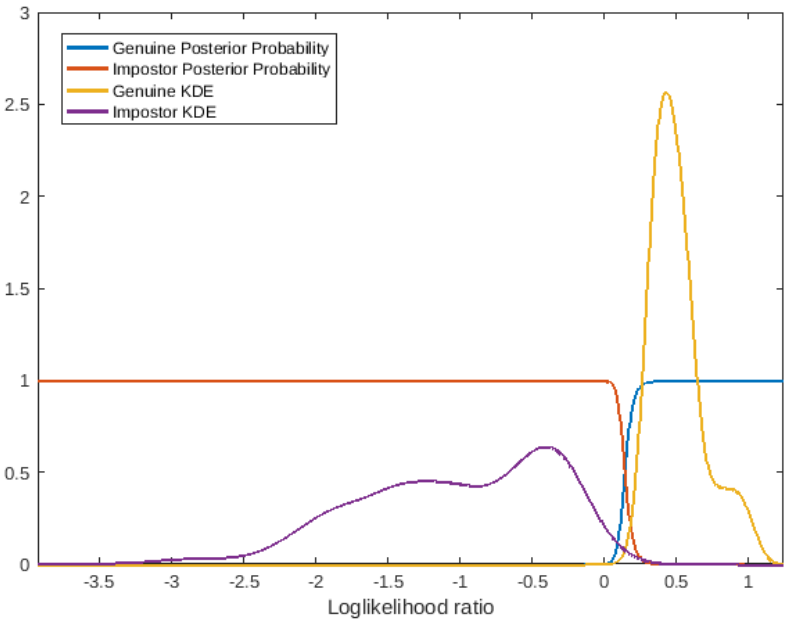


Figure 4.12: FMEV Posterior distribution and kernel density estimate

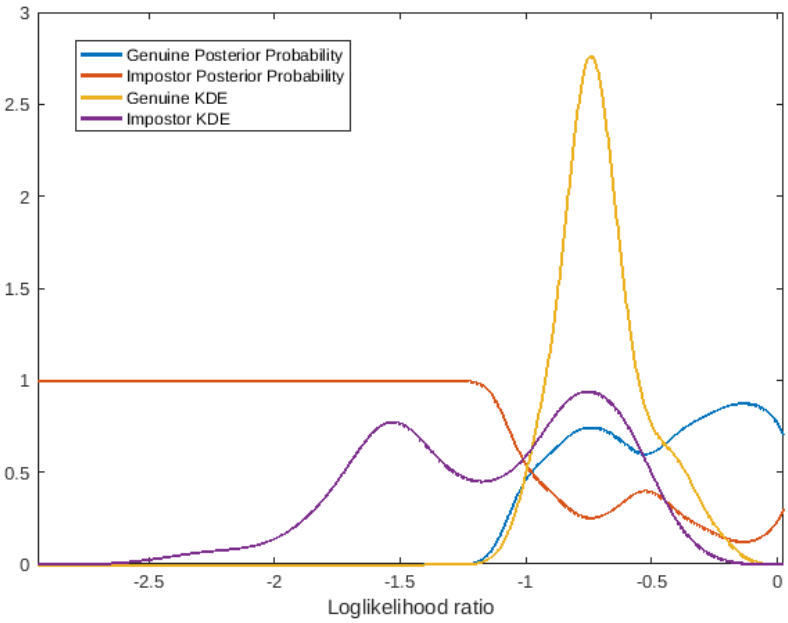


Figure 4.13: MMLP Posterior distribution and kernel density estimate

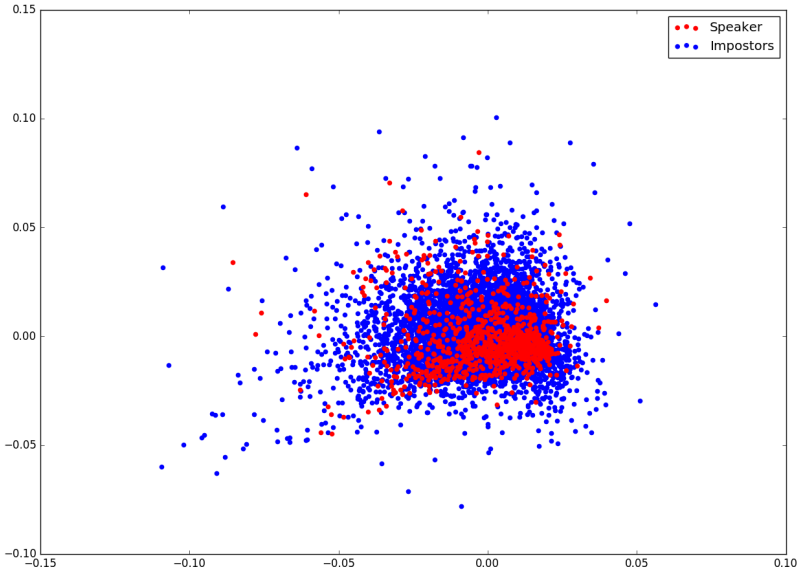


Figure 4.14: FAML Principal component analysis visualization

We also used principal component analysis to visualize our feature space. We projected the MFCC vectors generated from speaker #1's audio onto two dimensions, with all other speakers as impostors. As can be seen in fig. 4.14, the speaker audio occupies a small portion of the space containing all impostors. However, the speaker samples almost all overlap the impostor samples. This is to be expected, as all samples belong to recorded speech, and differences between speakers can be relatively subtle. This figure still shows there is a distinction between the two classes, even in just two dimensions. It might be worth it to investigate PCA as a possible noise reduction technique, but that is future work.

5

Product and Process Evaluation

In this section we look back on the design goals and requirements, to determine which goals and requirements have been met and which have not been met. We discuss all *must have* and *could have* requirements but only the *should have* requirements that we have put considerable effort into. Additionally, we evaluate the process of the project.

5.1. Product Evaluation

5.1.1. Evaluation of Requirements

Requirement	Met?	Explanation
The user is able to log in using face verification.	Yes	Our application uses local binary patterns for face verification.
There is a fallback to username-password login.	Yes	After 15 authentication attempts, the application allows the user to log in with username-password.
The user can register.	Yes	The first time a user opens the application, he can register photos and sound clips to register his biometric data.
The user can look into the data that is used for login.	Yes	There is an overview of all the biometric data that is used by the application. The user can add and remove his own data.
False positive rate is not above 5%	Yes	The user can set the maximum false positive rate between 0% and 10%, with a default of 5%.
A genuine user can log in within one minute, while an impostor cannot log in within five minutes.	Yes	A genuine user can always log in within one minute, even if our face and speaker verification fail, because of the fallback to password. However, generally the face and speaker recognition will work in under 15 seconds for a genuine user. If an impostor is not authenticated within 15 seconds, he will not be able to log in at all, again because of fallback to password. Users can set the confidence required to log in, in a range from 90% to 100%, and at 100% required confidence it is fair to say that an impostor would not be able to log in within 5 minutes.

Table 5.1: Must have requirements

Requirement	Met?	Explanation
The application has a modular software architecture.	Yes	We have divided the project into two libraries and an application. The two libraries contain the face and speaker recognition respectively and the application can authenticate a user with these two techniques.
Speaker verification	Yes	We have developed our own speaker verification system using Gaussian mixture models with MFCC feature vectors.
The user can be identified by face or voice.	Partial	The user can be identified by face or voice, but because a laptop microphone adds a considerable amount of noise, there are some practical limitations to our system. To overcome this, we would have had to record our own dataset with the same laptop microphone for all speakers, or added sophisticated noise reduction and normalization techniques which we did not do due to time constraints.
The user can be identified independent of hair style, glasses, and facial hair.	Partial	Our system is hair style independent, as long as the hair does not occlude the face. The system works independent of facial hair and glasses if the system already has pictures with the user wearing them. If there is no picture of a user wearing glasses already, the user probably will not be recognized wearing glasses. The same holds for large variations in facial hair. If only the top half of the face is used, as discussed in section 4.1, the system does work independent of facial hair, but we chose not to use this technique as it worsens the overall performance of the face verification.

Table 5.2: Should have requirements

Requirement	Met?	Explanation
The application is able to learn online, by adding successful authentication attempts to its data set	Yes	On a successful login attempt, whether by face and voice verification or by username and password, the system adds a face picture to its model.
The application is extendable with additional biometric techniques	Yes	Because of the modular design of our application, it is possible to add additional biometric techniques to the authentication system
The application has a mouth movement detection algorithm, that checks whether the mouth moves if the user speaks	Partial	We have developed a mouth movement detection algorithm as discussed in section 3.6. However, we have not integrated the algorithm in our application due to time constraints.
The system is integrated into a login manager.	No	Integration into a login manager would be a large investment of our time, as it would require us to write a custom Pluggable Authentication Module and Greeter. We have looked into this, but our client decided that we should prioritize improving performance and adding more features to the application over integrating it into a login manager. In the end we did not have sufficient time to meet this requirement.

Table 5.3: Could have requirements

5.1.2. Evaluation of Design Goals

In this section we evaluate if we have succeeded in meeting the design goals that we have defined in section 2.3.

The first design goal that we defined is usability. We have met this design goal by automatically starting a background thread that continuously tries to authenticate the user, and automatically authenticates him once it is sure enough that the user is genuine. In this way, no user action is needed to be authenticated. This generally works in under fifteen seconds, but if after fifteen seconds the system has not yet verified the user, the user has to log in using username and password. In the most common use case, where a genuine user is identified within fifteen seconds, the user needs to perform no actions, other than to sit in front of his webcam.

The second design goal is security. We have met this goal by creating an authentication system that can authenticate users with a false positive rate between 10% and 0%, depending on the wishes of the user. If a user chooses for a lower false positive rate, the application will become more secure, but it might take longer for a genuine user to log in. However, we have found that with a very low false positive rate (between 5% and 0%) the application generally still authenticates genuine users in under fifteen seconds.

Additionally, by using both face and speaker recognition in our application, we have created redundancy that leads to a more secure application. Moreover, by implementing a fallback to password, we prevent users from being locked out of the application.

The final design goal is maintainability, which we have met by designing and implementing our application in a modular fashion. We have created multiple independent projects that come together in a thin UI application. This improves the maintainability of our application and makes it easier to add additional biometric authentication methods in the future. We have also documented our code thoroughly to aid maintainability.

5.1.3. Product Evaluation Conclusion

In section 2.4 we have defined *must have* requirements as requirements that must be met in order to get a working product and *should have* requirements as requirements that should be met as discussed with the client. So to be able to call our project a success we should meet all *must have* and *should have* requirements. Following this standard for a successful project, we can conclude that our project was a success. We have met the aforementioned requirements, with only minor remarks on two *should have* requirements.

5.2. Process Evaluation

Due to some issues not entirely within our control, we started our project a week late, leaving us with only nine weeks. However, we managed to make up for lost time during the research phase and the development of the first prototype.

A number of times we ran into some issues during integration, mainly related to different dependencies of different parts of the system, but we managed to solve them eventually. Even though we lost some time fixing these issues we did not run into any major hurdles during the course of the project.

Because we had three fairly distinct subsystem, i.e. face recognition, speaker recognition, and the application, we had little trouble dividing tasks and working in parallel. At the beginning of the project, we looked at each member's area of expertise and interest to divide the tasks. Communication within the group also went very well. We worked all weekdays on location at the TU Delft, which greatly benefited communication. If one member could not make it, we kept in contact by using chat applications. Because we had a clear separation of tasks, we kept each-other up to date on the progress of our own tasks.

Our weekly meetings with our coach and client made sure we stayed on track, both in the right direction as well as at a sufficient pace, and provided a lot of structure to

a project with a relatively large amount of freedom. Meeting regularly with the client helped us continuously prioritize what we should focus on during development. For all meetings we prepared a small presentation, which helped us develop a slide deck that we could use for the final presentation. We also kept the final report up-to-date over the course of the project so that we wouldn't have any last minute problems there.

All in all we are very satisfied with the process of this project. The co-operation and communication between team members was generally good. Hiccups in the process were mostly of technical nature. Despite a few minor issues, which are to be expected, the project went well.

6

Conclusion

The task assigned to us at the start of this project was to create a system which allows a user to log in using their face or voice. We set out by mapping the design goals and requirements of our client. To get a thorough understanding of the problem, we researched face and speaker verification techniques. Keeping our design goals, requirements and research in mind, we created an initial prototype. Using the prototype as a base, we iteratively developed the system until it satisfied all requirements.

We have found local binary patterns to be the best performing face recognition algorithm for our application. It is especially useful that LBP is invariant under heterogeneous illumination variations, as this is a big problem for eigenfaces and Fisherfaces. Together with cropping and illumination normalization algorithms, we have created a robust face verification system. Our custom blink detection system increases the security of the face verification module by preventing against attackers that try to get authenticated using a photograph of a genuine user.

We were also quite successful in implementing a speaker verification algorithm. After researching the possibilities, we decided to implement a Gaussian mixture model with a universal background model to classify Mel-frequency cepstrum coefficient feature vectors made using preprocessed audio frames. Performance on the ELSDSR database is generally good, but real life performance through a low quality laptop microphone leaves a lot to be desired. However, this is no insurmountable obstacle, and we expect real life performance to improve with more sophisticated voice activity detection and noise reduction techniques.

We did not integrate our two modules into a login manager such as LightDM, but instead created our own mock login application. We, together with our client, feel that this was the best course of action, because it gave us more time to focus on our authentication modules, which were most important to our client.

In conclusion, we have created a login application using face and speaker verification that meets the client's requirements and expectations. For this reason, we see the project as a success.

7

Recommendations

In this section we give our recommendations to the client and ideas for future work.

Regarding face verification, there are some ways in which the user experience and performance of our verification algorithm can be improved. In the current version of the application, users have to register images of their open and closed eyes, as training data for the blink detection algorithm. The same goes for the mouth movement detection algorithm. This could be improved by creating a data set of open and closed eyes from many different individuals and under many different lighting conditions, to create a generic blink detection algorithm.

Our face verification algorithm uses the LBP algorithm. While the performance of LBP is quite good, we expect that better performance can be achieved by using a combination of various classifiers, e.g. eigenfaces and LBP. This could create a face verification algorithm that is more robust under different conditions and use cases.

Our face verification model authenticates users based on fifteen independent facial images. More sophisticated authentication strategies, that use the aggregate confidence of a set of pictures could yield better performance. This requires no modification of our face verification module, so it could be implemented directly in the application that uses the module.

Initially we wanted to integrate our face and speaker verification into a login manager, but we decided against this because it did not fit the scope of our project. However, because of the modular software architecture or the application, one could use our face and speaker verification modules as libraries and integrate them into a login manager.

The application adds a picture of the user to the face verification data set on each successful login. However, sometimes this image will not necessarily improve face verification. For example, the user might have his eyes closed on the image, or the image might be moved. We recommend a system that checks whether adding the image to the data set improves the performance of the face verification algorithm.

Our speaker verification module can be improved in several ways. First of all, we have only used energy based voice activity detection. While this works well for high quality recorded speech with little noise, performance of this method is lacking as soon as background noise increases or recording quality decreases. This energy based method could be replaced by a long term spectral divergence based one for better performance in noisy environments, or another GMM classifier trained on noise and speech could be used. Determining what method works best is future work.

Secondly, we have not been able to evaluate our model with delta/delta-delta MFCC feature vectors due to time constraints. It is expected this would further improve performance, as useful additional features capturing temporal information could be incorporated in the MFCC feature vectors that way.

Lastly, we have experimented with classifying the MFCC vectors using neural networks, but these were not very extensive. Our trial with a simple multilayer perceptron for classification seemed promising, but due to time constraints we did not investigate this further.

Bibliography

[1] Mel frequency cepstral coefficients. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#deltas-and-delta-deltas. Accessed: 2016-04-28.

[2] Gaussian mixture models. http://scikit-learn.org/stable/modules/mixture.html. Accessed: 2016-04-28.

[3] Aquila homepage. http://aquila-dsp.org/about/. Accessed: 2016-05-25.

[4] Elsdsr homepage. http://www2.imm.dtu.dk/~lfen/elsdsr/. Accessed: 2016-05-25.

[5] Lightdm homepage. https://www.freedesktop.org/wiki/Software/LightDM/. [Accessed 2016-04-26].

[6] pam-face-authentication. https://code.google.com/archive/p/pam-face-authentication/. Accessed: 2016-05-02.

[7] Qt homepage. https://www.qt.io/. Accessed: 2016-05-03.

[8] Aligning face images script. http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#aligning-face-images. Accessed: 2016-05-11.

[9] Sayed Jaafer Abdallah, Izzeldin Mohamed Osman, and Mohamed Elhafiz Mustafa. Text-independent speaker identification using hidden markov model. *World of Computer Science and Information Technology Journal (WCSIT)*, 2(6):203–208, 2012.

[10] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *Computer vision-eccv 2004*, pages 469–481. Springer, 2004.

[11] Mohd. Manjur Alam, Md. Salah Uddin Chowdury, and Niaz Uddin Mahmud. Article: Text dependent speaker identification using hidden markchov model and mel frequency cepstrum coefficient. *International Journal of Computer Applications*, 104(14):33–37, October 2014. Full text available.

[12] Marian Stewart Bartlett, Javier R Movellan, and Terrence J Sejnowski. Face recognition by independent component analysis. *Neural Networks, IEEE Transactions on*, 13(6):1450–1464, 2002.

[13] Aziz Umit Batur and Monson H Hayes III. Linear subspaces for illumination robust face recognition. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–296. IEEE, 2001.

[14] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[15] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 15(10):1042–1052, 1993.

[16] Pierre Comon. Independent component analysis. *Higher-Order Statistics*, pages 29–38, 1992.

[17] Ryan R. Curtin, James R. Cline, Neil P. Slagle, William B. March, P. Ram, Nishant A. Mehta, and Alexander G. Gray. mlpack: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[18] Kresimir Delac, Mislav Grgic, and Sonja Grgic. Independent comparative study of pca, ica, and lda on the feret data set. *International Journal of Imaging Systems and Technology*, 15(5):252–260, 2005.

[19] Joshua C Klontz, Brendan F Klare, Scott Klum, Anubhav K Jain, and Mark J Burge. Open source biometric recognition. In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pages 1–8. IEEE, 2013.

[20] Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582–585. IEEE, 1994.

[21] Javier Ramırez, José C Segura, Carmen Benıtez, Angel De La Torre, and Antonio Rubio. Efficient voice activity detection algorithms using long-term speech information. *Speech communication*, 42(3):271–287, 2004.

[22] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1):19–41, 2000.

[23] Md Sahidullah and Goutam Saha. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech Communication*, 54(4):543–565, 2012.

[24] Xiaoyang Tan and Bill Triggs. Enhanced local texture feature sets for face recognition under difficult lighting conditions. In *Analysis and Modeling of Faces and Gestures*, pages 168–182. Springer, 2007.

[25] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[26] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[27] Max Welling. Fisher linear discriminant analysis. *Department of Computer Science, University of Toronto*, 3, 2005.

[28] Laurenz Wiskott, Jean-Marc Fellous, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):775–779, 1997.

A

Algorithms

A.1. OpenCV's LPB algorithm

The local binary patterns algorithm finds its origin in texture classification[20] and was later found to also be a good algorithm for face recognition.

The LBP algorithm is a way to extract low-dimensional features from much higher dimensional images. First, the image is divided into $n \times m$ frames. For each frame, a histogram is constructed by using the local binary pattern operator.

In figure A.1 a radius = 1, neighbors = 8 local binary pattern operator is shown, but LBP operators with a larger radius and more neighbors also exist. For the purpose of explaining the workings of the LBP algorithm we take the easiest case, which is the case shown in A.1.

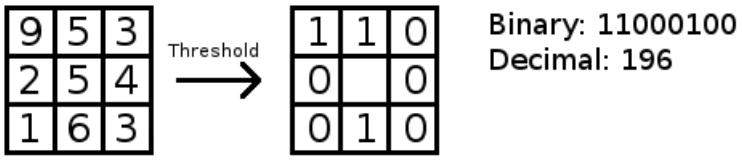


Figure A.1: Example of the LBP operator

The LBP operator is applied to each pixel within a frame. This is done by taking the pixel and looking at its neighbors. The value of the middle pixel is taken as the threshold, and each of its neighbors is set to a new value, which is 1 if the neighbor's value is higher than the threshold, and 0 if the neighbor has a lower value. Starting at the top-left pixel and moving in a clockwise manner, a binary number is constructed, which is then converted to decimal. This decimal number is the result of the LBP operator on the given pixel. This can be expressed in the following formula:

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

where the function s is defined as:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

By doing this for each pixel in a frame, a large set of decimal numbers is generated. The histogram of this set represents the feature vector of the frame in question. In the OpenCV implementation, nearest neighbors is used to classify these feature vectors.

A.2. OpenCV's Cascade Classification

We use OpenCV's cascade classification for face detection, which is based on Haar feature-based cascade classification by Viola and Jones[26]. This kind of classifier is especially useful for us because it is very efficient, enabling us to detect faces in real time.

Haar features are calculated by subtracting the sum of pixel values of one area from the sum of pixel values from another area. An example of two Haar features is given in figure A.2. The sum of pixel values in the white areas are subtracted from the sum of the pixel values in the black area resulting in a Haar feature.

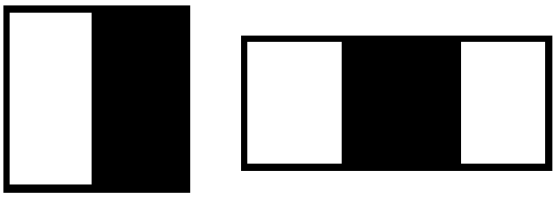


Figure A.2: Example of Haar features

In order to speed up the calculation of the sum of pixel values of an area, the integral image is used. The integral image is defined as follows:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

where $i(x', y')$ is the pixel value at pixel (x', y'). Using the integral image, the sum of pixel values of an area can be calculated using the integral image value of its four corner points.

Because Haar features can exist in any width and height within a window, a single image produces a very large amount of Haar features, which exceeds the amount of pixels in the original image. Because of this, AdaBoost is used to find the most representative features that, in our case, can distinguish faces from non-faces. By only using the most representative features, the number of features decreases significantly. AdaBoost is also used for classification of the features.

To detect faces, the algorithm scans an image window by window, applying the most representative Haar features to determine which windows contain a face. To speed up this process, cascade classifiers are used. The idea behind cascade classifiers is that the classification is split up into stages. The first stage uses the most representative feature to detect a face and each subsequent stage uses a less representative feature. If during a stage no face is detected, the subsequent stages will not be used. This ensures that windows in which there is clearly no face present are skipped at an early stage, speeding up the detection process.

B

Additional Performance Tests

B.1. OpenCV's Eigenfaces Algorithm

For the evaluation of the eigenface algorithm, the first thing we wanted to do is get an idea of a good number of components (eigenfaces) to use. We did this by running the algorithm on the AT&T dataset with 25 times repeated random sub-sampling for a number of different components (just like we did for the other face recognition algorithms), see figure B.1 for results. We decided to set the number of components at 15, because after that there is significantly less difference between the impostors and genuine matches.

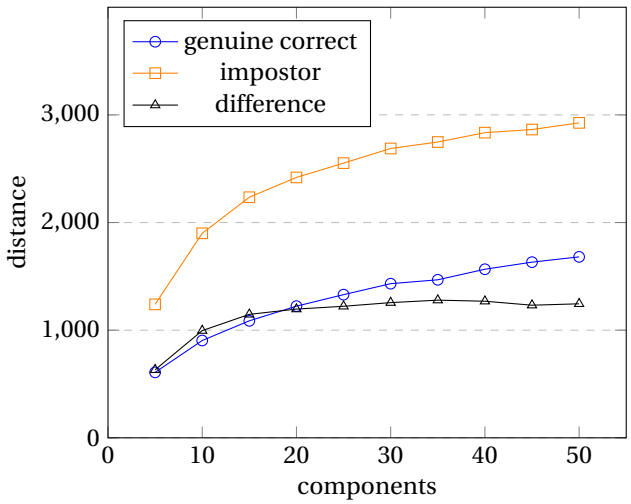
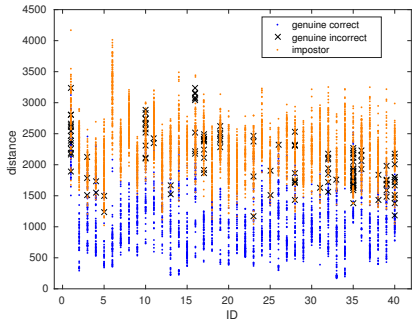


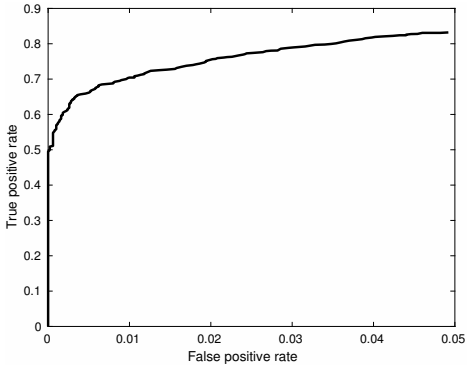
Figure B.1: Average distances of matches for different numbers of components, lower distance means higher confidence

Even though our research suggested that the eigenfaces algorithm isn't the best face recognition algorithm, it still showed itself as a usable classifier (see figure B.2), and also managed to keep a decent performance on a small training set (figure B.3).

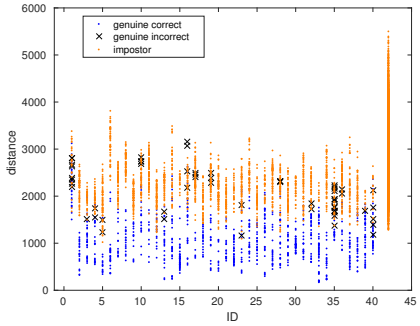
Because the AT&T dataset isn't a very large one, and we wanted a larger set of different impostors, we added frontal faces from the FERET database to the evaluation. Figure B.2c shows at what distance (normalized) frontal faces from the FERET database are valued. It is clear that none of the roughly 2400 impostor images come any closer the genuine matches than when testing solely on the AT&T set. Because this was also the case for all subsequent performance tests, the results including the FERET impostors have been omitted.



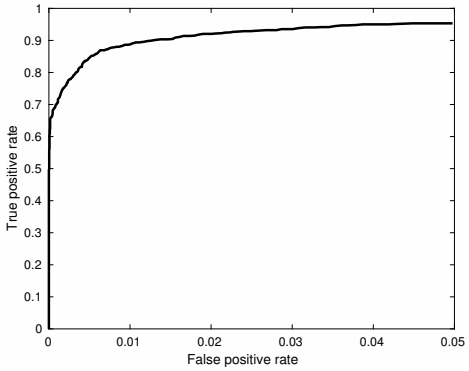
(a) Scatter plot AT&T only



(b) ROC curve AT&T only, $AUC_{0.05} = 0.742$

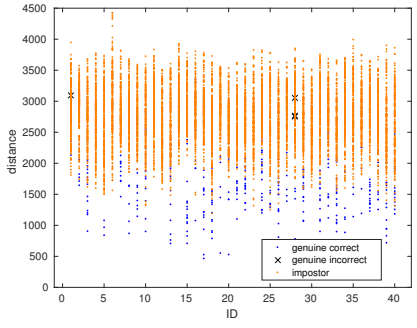


(c) Scatter plot with FERET

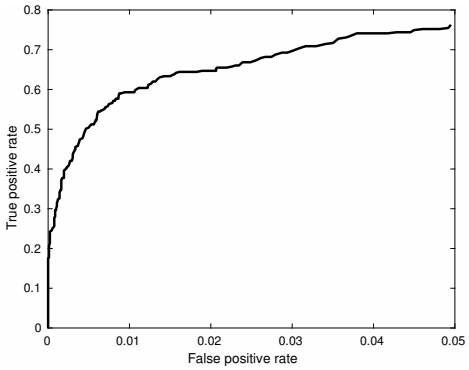


(d) ROC curve with FERET

Figure B.2: Genuine vs impostor matches on a training set of 20 people, 5 pictures each, form the AT&T dataset with OpenCV's eigenfaces algorithm.



(a) Scatter plot



(b) ROC curve, $AUC_{0.05} = 0.642$

Figure B.3: Genuine vs impostor matches on a training set of 3 people, 5 pictures each, from the AT&T dataset with OpenCV's eigenfaces algorithm.

B.2. OpenCV's Fisherfaces Algorithm

We did the same component evaluation on OpenCV's Fisherfaces algorithm as we did for the eigenfaces algorithm (see figure B.4). For the Fisherfaces there is no significant improvement in difference between impostors and genuine users when using more than 20 components, so that's the amount we use in any further evaluation.

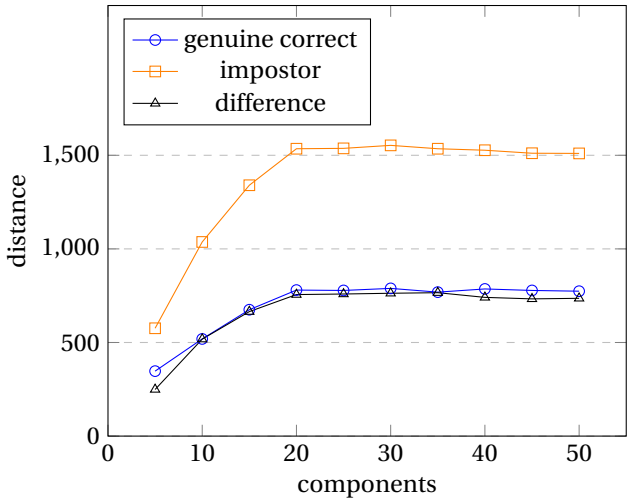
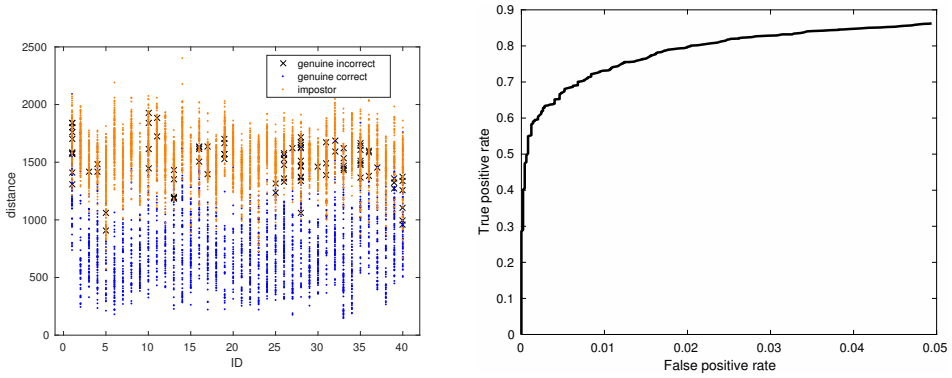


Figure B.4: Average distances of matches for different numbers of components

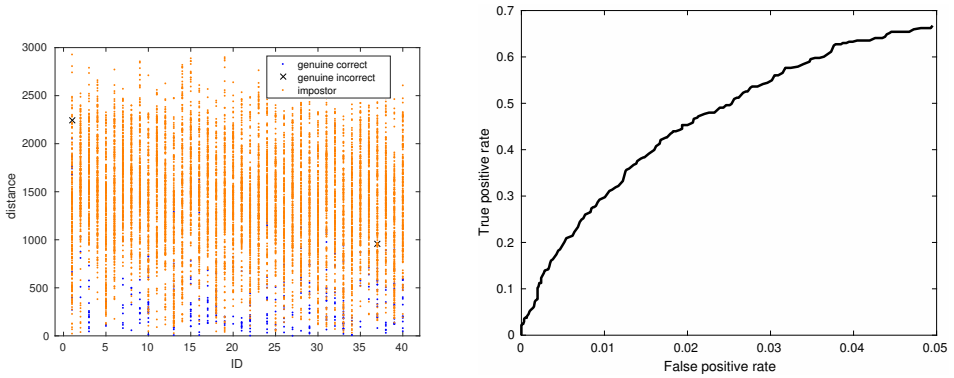
The Fisherfaces algorithm performs similarly to the eigenfaces algorithm on training sets consisting out of around 20 people (figure B.2 vs B.5). However, where the eigenfaces algorithm loses only a little performance on a smaller training set (figure B.3) the Fisherfaces algorithm becomes completely unusable (figure B.6). Because we would like to keep re-training the algorithm with the application online, and OpenCV's Fisherfaces algorithm isn't updatable, a smaller training set would be preferred. But since Fisherfaces doesn't work on a small training set it might not be the most suitable method for our purpose.



(a) Scatter plot

(b) ROC curve, $AUC_{0.05} = 0.774$

Figure B.5: Genuine vs impostor matches on a training set of 20 people, 5 pictures each, form the AT&T dataset with OpenCV's Fisherfaces algorithm.



(a) Scatter plot

(b) ROC curve, $AUC_{0.05} = 0.456$

Figure B.6: Genuine vs impostor matches on a training set of 3 people, 5 pictures each, from the AT&T dataset with OpenCV's Fisherfaces algorithm.

C

Software Improvement Group Evaluation

C.1. First Submission

Feedback

De code van het systeem scoort bijna 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Unit Complexiteit.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt.

Een voorbeeld van zo'n lange method in jullie systeem is `evaluateDataset` in `thresholddecider.cpp` of `makeScatterPlot` in `evaluation.cpp`. Er zijn hier aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld `//write results sp` zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Improvements

Based on SIG's feedback on our first submission we have made improvements to the application's source code. The two main points of feedback from SIG were that our code had a number of functions that were too long, and that our code lacked test code.

We have addressed the first point by dividing long and overly complex functions into smaller and simpler functions. This reduced the complexity of our code, and it improved

the readability of the code. We found that maintaining and extending our source code became easier after implementing this improvement as well.

Regarding the second point: the nature of our application makes it inherently difficult to automatically test. We would have to mock webcam frames and voice input. Instead of testing our application using test code, we have tested the algorithms in our application as described in section 4. Additionally, we have frequently tested our application 'by hand'. We agree with SIG that our system could benefit from adding test code, but due to time constraints we focused on improving the performance of the face and speaker verification algorithms over adding test code.

C.2. Second Submission

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Bij de punten die in de feedback op de eerste upload als verbeterpunt werden genoemd, Unit Size en Unit Complexity, zien we een duidelijke verbetering. Jullie hebben niet alleen de genoemde voorbeelden aangepakt, maar er ook voor gezorgd dat er in de nieuwe code niet meer van dergelijke constructies worden geintroduceerd. Jullie zaten met 4 sterren al vrij hoog, vandaar dat deze verbetering niet genoeg is om de totaalscore naar 5 sterrem te laten stijgen.

Net als bij de eerste upload hebben jullie geen testcode geschreven. Zoals al eerder genoemd is dat ook moeilijk met jullie technologiekeuze, maar desondanks heb je hier hnider aan, aangezien handmatig testen aanzienlijk meer tijd kost. Als jullie in de toekomst weer met C/C++ gaan werken is het aan te raden om eens naar de huidige staat van unit test frameworks te kijken.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject.

D

Original Project Description

“Kijk Mam, zonder handen!” Log in op je computer met je gezicht of stem
In plaats van met een password, of met een vingerafdruk, moet het ook kunnen om in te loggen op je laptop door je gezicht te laten zien, of je naam te noemen in de microfoon.

Om dit te kunnen doen, hebben we een systeem nodig dat uit de beeld van een kamera van de laptop het gezicht haalt, dit gezicht vergelijkt met gezichten die al in een database staan, en toegang geven tot de laptop als een bekend persoon voor de laptop zit. Natuurlijk moet je dan ingelogd worden met de juiste username. En nog belangrijker, onbekende personen mogen geen toegang krijgen!

Afhankelijk van de grootte van de groep, moet er in dit bachelor project een login applicatie gemaakt worden (voor Windows, of Linux, of MacOSX) die gebruikt maakt van gezichtsherkenning en/of stemherkenning, waarbij er gebruikers kunnen worden toegevoegd en verwijderd, waarbij verschillende fotos en stemfragmenten per gebruiker gebruikt kunnen worden, en waarbij de fotos en/of stemfragmenten kunnen worden gechecked. Het zou mooi zijn als de interface er goed uitziet, maar het is nog belangrijker dat het werkt...

E

Project Plan

In this document we outline the planning of our Bachelor thesis. We started a week later, but we plan to get on the original schedule over the course of the preparation phase of the project (weeks 2 and 3). Because of this, we shortened the research phase from two weeks to one and a half weeks, and aim to be done with the initial prototype after week 3.

Week 2-3
Research ($1^1/_2$ weeks). We will:

- Look at literature and existing methods
- Specifically we want to look at the following:
  - Face recognition
    - Libraries
    - Algorithms
    - Training methods data
    - Comparative studies
    - Face authentication/verification
    - Evaluation/validation strategy
  - Speaker recognition
    - Libraries
    - Algorithms
    - Training methods and data
    - Comparative studies
    - Evaluation/validation strategy
  - Login application
    - Find a suitable login manager to extend with our system
  - General software engineering
    - Compare programming languages and determine which is best suited for our project
- Find suitable datasets for training and evaluation
- Produce a 10 page report about our findings

Week 3

- Build prototype, get basic face recognition, speaker recognition working in simple interface
- Keep final report updated with progress
- Make a few slides + small presentation showing progress

Week 4 - 9

Iterative development, roughly weekly:

- Improve system
  - Continuously aim to improve performance
  - Add different (independent) predictors
  - Expand features of login app
  - Improve GUI look
- Evaluate and log performance
- Keep final report updated with progress
- Make a few slides + small presentation showing weekly progress

Final:

- Finalize report (30-50 pages) about design, implementation, results

Week 10

- Last tweaking/polishing of the system
- Prepare presentation + demo

F

Infosheet

Title of the project:	Login with face and speaker verification
Client:	David Tax, TU Delft
Presentation date:	June 24, 2016

Description
We have created a system, at the request of our client, which can authenticate users by their face and their voice. The final product consists out of three subsystem: a face recognition module, a speaker recognition module, and a mock login application.

At the core of the face recognition lies the local binary patterns algorithm implemented in the OpenCV library, which we found to have the best overall performance compared to a number of different algorithms we also evaluated. We also take a number of pre-processing steps to improve the performance. The final classification is done by a nearest neighbor algorithm. Additionally, we have developed and implemented a blink detection system to increase security.

For the speaker verification we start by extracting Mel-frequency cepstral coefficient feature vectors from raw audio data. These MFCC feature vectors are used to train two Gaussian mixture models: one for the speaker, and one for the background. The model can classify other MFCC vectors by matching them to either the speaker or the background, based on the log-likelihood ratio between the two models.

Both modules have been implemented and have been integrated into a mock login application. Using the application, users can register and review biometric data, and they can log in to a mock protected environment.

Members of the project team
Sebastiaan Brand
Interests: Machine learning, pattern recognition, data mining
Contributions: Face verification

Tom Catshoek
Interests: Machine learning, artificial intelligence, software engineering
Contributions: Speaker verification

Joost van Oorschot
Interests: Machine learning, software engineering, data mining
Contributions: Application, face verification

Client, Coach

Client: David Tax, TU Delft
Coach: Thomas Abeel, TU Delft

Contact

Sebastiaan Brand	sebastiaanbrand@gmail.com
Joost van Oorschot	jjevanoorschot@gmail.com
Tom Catshoek	t.w.j.catshoek@student.tudelft.nl

Report

The final report for this project can be found at http://repository.tudelft.nl/