Universitatea Politehnica din Bucuresti
Facultatea de Automatica si Calculatoare

Sistem de identificare a persoanelor bazat pe fuziunea semnalelor audio si video

Absolvent:
Vasile Bogdan Alexandru

Coordonatori:

Prof. Dr. Stefanoiu Dan

Prof. Dr. Balan Radu

Septembrie 2006

I would like to thank all the persons that helped me during the Siemens internship, by offering both moral support and precious advice.

I am very grateful for the chance offered by prof. Stefanoiu Dan, who believed in my capabilities. He was also the one who taught me the basics of signal processing, giving me the possibility to accede to an interesting and always changing world that already influences our every day life.

My special thanks and admiration go to Mr. Radu Balan, who amazed me with the vast knowledge from multiple domains of activity. He always helped me by giving me advice or by telling me where to look for a solution. I found his work and performance very inspiring.

I would also like to thank my teachers for letting me get involved in many projects and for giving me the knowledge that I appreciated so much.

Contents

Chapter 1

Introduction

The domain of identity recognition has received an increasing attention in the recent years. Most of the methods of securing the access to a resource are based on physical devices like magnetic cards or RFID. But the greatest disadvantage is that the resource is compromised if one of the access devices are lost or stolen. So being able to identify a person not based on the physical devices but on the unique characteristics of the human body would allow implementing a better security system. This is the area of competence for biometrics.

Biometrics is the study of automated methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits [10].

Biometric authentication refers to technologies that measure and analyze human physical and behavioral characteristics for authentication purposes. Examples of physical characteristics include fingerprints, eye retinas and irises, facial patterns and hand measurements, while examples of mostly behavioral characteristics include signature, gait and typing patterns. Voice is considered a mix of both physical and behavioral characteristics. However, it can be argued that all biometric traits share physical and behavioral aspects.

The first known modern system of biometric identification is the Bertillon system [11], invented by the French criminologist Alphonse Bertillon (1853-1914). The system consisted in recording the height, the length of one foot, an arm and index finger, along with the subject's photo. The system was officially adopted in France in 1888 and it was later introduced in the other European countries. It was later replaced by a fingerprint verification system.

Although the use of biometrics can provide a better security, implementing such a system poses lots of difficulties, which are determined by:

- Intra - class variability and inter - class similarity
- Segmentation of data and extracting useful signal from background
- Noisy input
- Individual performance of classification algorithms
- Scalability of the algorithms

Excepting the technical difficulties, there are other problems related to the attacks on the biometric systems, issues related to the privacy of the individuals, the ease of acceptance by users and the cost of the system.

The technologies that are less intrusive (like audio and face recognition) tend to be accepted

easier than other technologies (retina scanning). Another advantage of the audio and visual recognition is the low cost of the identification system. The identification can be performed using low cost sensors and a normal desktop or laptop can serve as a processing unit.

A problem of the recognition systems is their accuracy. Although there are systems that reported 100% recognition rate in the absence of noise [4], the performance degrades as soon as the input signal becomes corrupted by noise.

Combining multiple methods of person recognition improves the recognition rate under different noise levels. In the case of audio and video recognition, while the presence of background noise has a detrimental effect on the performance of voice biometrics, it doesn't affect the face biometrics. On the other hand, while the performance of face recognition depends heavily on illumination conditions, lighting doesn't have any effect on the voice quality.

In this paper a system that is capable of recognizing a user based on audio and video data is presented.

We use two separate expert systems for computing scores for the audio data and video data. Then the scores are integrated in a statistically principled manner to obtain the classification result. The data is first recorded then is processed in an automatic fashion by our software. The data sensor is a Logitech QuickCam Pro 4000, connected to the USB port of the computer.

After stopping the recording, the system starts to compare the test data with the training data, obtaining a set of scores for each user. Based on these scores, a posterior probability is computed. The identified user is chosen as the one having the highest computed posterior probability.

Chapter 2

Theoretical Background

2.1 Audio Recognition

Speech is one of the most powerful and a natural form of communication. Our voice also contains various cues to our identity, like the gender, emotion, attitude, health status as well as distinctive and identifiable. Everyday unknowingly, or sometimes in a determined way, we "identify" and get assurance that we are indeed speaking to the "right person".

However, what we are able to do naturally everyday, in a face-to-face conversation, or over telephone, or over some other communication device, i.e. the task of speaker identification/recognition, is still remaining an elusive target.

2.1.1 Problem Definition

Speaker recognition is the task of analyzing speech signal to automatically recognize the speaker. Speaker recognition tasks can be broadly classified into:

- Speaker Identification (SI)
- Speaker Verification (SV)

Given a sample of speech, Speaker identification (SI) is the task of deciding who amongst a set of N candidate speakers said it. The identification task is called a *closed set* task when the speaker being tested is always one of the N candidate speakers and called an *open-set task*, when the speaker being tested may not be one of the candidate speakers. The value of N has a powerful impact on performance - performance can be very good for just a few speakers but can fall drastically when N becomes quite large.

Given a sample of speech, Speaker verification (SV) is the task of deciding whether a certain specified candidate said it. This is also known as speaker detection. As opposed to the N-class SI, SV is a 2-class task, the two classes being the specified speaker also called the target speaker or true speaker) and some speaker other than the target speaker (also called an impostor). Therefore, the performance of SV is somewhat independent of the number of potential impostor speakers.

According to their operating modes , speaker recognition systems can also be divided into:

- text-dependent
- text-independent

- prompted

In text-dependent approach, the speaker recognition system "knows" what the speaker is supposed to say. In other words, the same set of key-words, phrases or sentences are used during training of the system and during actual testing/use. Text-dependent systems are understandably more accurate as the content variability is removed. Also the amount of speech data required will be less here.

The text-independent approach is the more challenging one (particularly as it operates in a manner as humans perform) which does not used any "pre-fixed" utterance, but need to recognize speakers by processing whatever they are saying. This force the system to use whatever data is available - for training as well as testing.

In the text-prompted speaker recognition method, the recognition system prompts each user with a new key sentence every time the system is used and accepts the input utterance only when it decides that it was the registered speaker who repeated the prompted sentence. The sentence can be displayed on the monitor. Because the vocabulary is unlimited, prospective impostors cannot know in advance what sentence will be requested. Not only can this method accurately recognize speakers, but it can also reject utterances whose text differs from the prompted text, even if it is spoken by the registered speaker. If the likelihood is high enough, the speaker is accepted as the claimed speaker. A recorded voice can thus be correctly rejected.

Automatic speaker recognition technology relies on the basic fact that people's voices are distinct, i.e., each person's speech possesses and exhibits distinctive characteristics indicative of the identity of the speaker. Thus, one can be tempted to assume that such "distinctiveness" of the voice of a person will be somewhat constant over time and over various set-up or conditions in which the signals are recorded.

In practical terms, such voice-prints are captured as a speech signals recorded through various microphones, or over various voice communication systems, in a quiet or noisy background. Therefore, we see variability due to the "channel" (microphone used, acoustical noise, telephone line / channel characteristics, telephone handsets, distance between speaker and microphone, etc). If there is a mismatch in any of these channel variability factors between training and testing, current systems exhibit significant drop in performance.

Apart from channel variability, there is intra-speaker variability as well. It has been observed that if the training and test sessions are conducted at different period of time, even though the speaker set is same, due to the variation of "speaker-performance", the recognition ability of the recognition system drops. The speech signal of the same user can vary over time due to different causes, which include changes in the emotional state, aging or sickness. Therefore, the core technique and design aspects of a speaker recognition system must somehow circumvent this "intra-speaker" variability.

Both intra-speaker variability and channel variability, impact quite significantly on the performance. Therefore, robustness against these variable conditions is the most important target of current researchers of speaker recognition.

2.1.2 Basic Architecture of a Speaker Recognition System

The basic recognition system comprises two distinct parts:

- front-end processing, which produces a set of features of the signal
- back-end processing, represented by the classifier

Front end processing

The front end processing includes the following stages:

- signal acquisition
- sound pre-processing
- feature extraction

The signal is acquired from a microphone and is affected by the channel transfer function. The microphone used in the A/D conversion process usually introduces undesired side effects, such as line frequency noise ("50/60 Hz hum"), loss of low and high frequency information, and nonlinear distortion [2] . The A/D converter also introduces its own distortion in the form of a less than ideal frequency response and nonlinear input/ output transfer function, and fluctuating DC bias. Because of the limited frequency response of analog telecommunications channels, and the widespread use of 8 kHz sampled speech in digital telephony, the most popular sample frequency for the speech signal in telecommunications is 8 kHz. In nontelecommunications applications, in which the speech recognition subsystem has access to high quality speech, sample frequencies of up to 16kHz have been used. These sample frequencies give better time and frequency resolution.

The main purpose of the digitization process is to produce a sampled data representation of the speech signal with as high a Signal to Noise ratio (SNR) as possible. A denoising algorithm can be used to enhance the SNR of the speech signal. Still, sometimes applying a denoising algorithm can cause the appearance of distortions in the signal, which might affect the recognition accuracy.

Another step taken is to remove the unnecessary silence from the input. A simple yet effective algorithm that isolates the speech from silence is presented in Appendix B.

Once signal conversion is complete, the last step of digital postfiltering is most often executed using a Finite Impulse Response (FIR) filter:

$$H_{pre}(z) = \sum_{k=0}^{N_{pre}} a_{pre}(k) z^{-k} \tag{2.1}$$

Normally, a one coefficient digital filter, known as a preemphasis filter, is used:

$$H_{pre}(z) = 1 + a_{pre}(k) z^{-1} \tag{2.2}$$

A typical range of values for a_{pre} is $[-1.0; -0.4]$. Values close to -1.0 that can be efficiently implemented in fixed point hardware, such as -1 or -(1-1/16), are most common in speech recognition. The preemphasis filter is intended to boost the signal spectrum approximately 20 dB per decade (an order of magnitude increment in frequency).

After acquiring a speech signal, the final stage of front end processing is the feature extraction.

Filter bank methods (implemented in analog circuits) were historically the first methods introduced. Linear prediction methods were introduced in the 1970's, and were the dominant technique through the early 1980's [2].

In the 1980s the cepstrum began to supplant the use of the LP parameters because it could easily smooth the LP-based spectrum and removes its inherent variability due to excitation. This lead to an improvement in performance [3]. Cepstral analysis is a special case of methods collectively known as "homomorphic" signal processing.

The cepstrum of a signal is computed using the transformations from figure 2.1, where *w(n-m)* is a time window of length *N* ending at frame *m*.

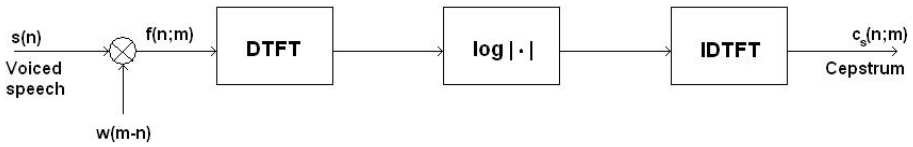

Figure 2.1: Schematic of short time Real Cepstrum computation

A second improvement over direct use of LP parameters can be obtained by use of the so-called "mel-based cepstrum" or simply "mel-cepstrum", being related to the principle of psychoacoustics, which studies human auditory perception.

A "mel" is a unit of measure of perceived pitch or frequency of a tone. It doesn't correspond linearly to the physical frequency of the tone. Stevens and Volkman (1940) arbitrarily chose the frequency 1000 Hz and designated this "1000 mels". Then, during experiments, listeners were asked to change the physical frequency of a sound until the pitch they perceived was twice the reference, then 10 times and so on; These pitches were labeled 2000 mels, 10000 mels and so on. The investigators were able to determine a mapping between the real frequency scale (Hz) and the perceived frequency scale (mels). The mapping is approximately linear below 1 kHz and logarithmic above.

In speech recognition, an approximation is usually used:

$$Mel\_frequency = 2595 \cdot log_{10}\left(1 + \frac{f}{700.0}\right) \tag{2.3}$$

Also, it has been found that the perception of a particular frequency by the auditory system is influenced by energy in a critical band of frequencies around that frequency. Further, the bandwidth of a critical band varies with frequency, beginning at about 100 Hz for frequencies below 1 kHz, and then increasing logarithmically above 1 kHz. Therefore, rather than simply using the mel-distributed log magnitude frequency components to compute the short term real cepstrum, some investigators have suggested using the log total energy in critical bands as inputs to the final IDFT. Y(i) is used to denote the log total energy in the *i*th critical band. Y(i) will be symmetrical about the Nyquist frequency. The frequency corresponding to the center of the filter is computed by dividing the frequency range on the mel scale to the number of filters, multiplying it with the index of the filter and then transforming it back to the real frequencies domain.

We can write the value of Y(i) as:

$$Y(i) = \sum_{k=0}^{N/2} log|S(k;m)| \cdot H_i\left(k \cdot \frac{2\pi}{N'}\right) \tag{2.4}$$

with H_i being the value of the *i*th filter.

If we define:

$$\tilde{Y}(k) = \begin{cases} Y(i), & k = k_i \\ 0, & \text{other } k \in [0, N'-1] \end{cases} \tag{2.5}$$

Considering that the sequence $\tilde{Y}(k)$ is symmetrical about $\frac{N'}{2}$, such that we can take the cosine transform instead of the IDTFT, the value of the cepstrum coefficient will be:

$$c(n,m) = \sum_{i=1}^{N_{filters}} \tilde{Y}(k_i) \cdot cos\left(k_i \cdot \frac{2\pi}{N} \cdot n\right) \tag{2.6}$$

The whole process is summarized in figure 2.2.

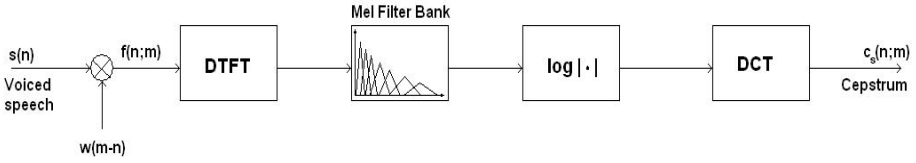

Figure 2.2: Schematic of MFCC computtation

Back end processing

The structure of the back end processing can be seen in Figure 2.3.

The feature vectors obtained in the front-end processing are compared against the model of each speaker and a score is computed for each speaker from the database. These scores are then sent to a decision block that chooses the speaker based on a probability model or a function of the scores.

Some of the procedures used to model a speaker are the following ([12]):

Template Matching In this technique, the model consists of a template that is a sequence of feature vectors from a fixed phrase.

During verification a match score is produced by using dynamic time warping (DTW) to align and measure the similarity between the test phrase and the speaker template. This approach is used almost exclusively for text-dependent applications.

Nearest Neighbor In this technique, no explicit model is used; instead all features vectors from the enrollment speech are retained to represent the speaker. During verification, the match score is computed as the cumulated distance of each test feature vector to its k nearest neighbors in the speaker's training vectors.

To limit storage and computation, feature vector pruning techniques are usually applied.

Neural Networks The particular model used in this technique can have many forms, such as multi-layer perceptions or radial basis functions. The main difference with the other approaches described is that these models are explicitly trained to discriminate between the speaker being modeled and some alternative speakers. Training can be computationally expensive and models are sometimes not generalizable.

Hidden Markov Models This technique uses HMMs, which encode the temporal evolution of the features and efficiently model statistical variation of the features, to provide a statistical representation of how a speaker produces sounds. During enrollment HMM parameters are estimated from the speech using established automatic algorithms. During verification, the likelihood of the test feature sequence is computed against the speaker's

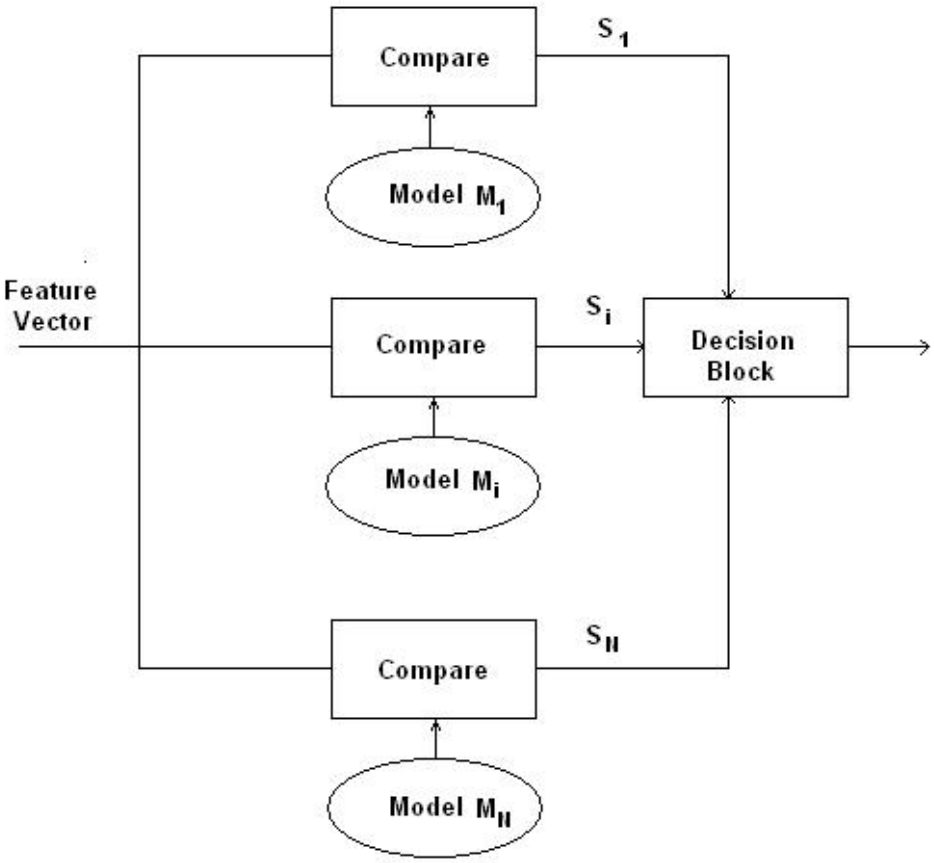

Figure 2.3: Schematic of back end audio processing for speaker identification

HMMs. For text-dependent applications, whole phrases or phonemes may be modeled using multi-state left-to right HMMs. For text-independent applications, single state HMMs, also known as Gaussian Mixture Models (GMMs), are used. From published results, HMM based systems generally produce the best performance.

The system implemented in the project is based on the dynamic time warping algorithm with multiple templates for each user. The paper is included in Appendix C.

2.2 Video Recognition

The task of video recognition is composed from two separate processes:

- face detection
- face recognition

The face detection technology consists in extracting a face or multiple faces from an image.

There are multiple scenarios for the face detection, the easiest being the ones with controlled background. Nowadays algorithms are able to extract faces from real life images, at a high speed, using normal processing equipment like laptops.

Another classification of the face detection algorithms refer to whether the detection is based on locating the components of a face, or it has a global approach.

The major difficulties of the face detection and recognition algorithms are related to ([13]):

- Viewing angle

  The face has a 3D shape. As the camera pose changes, the appearance of the face can change due to a) projective deformation, which leads to stretching and foreshortening of different part of face, and b) self occlusion and dis-occlusion of parts of the face. If we have seen faces only from one viewing angle, in general it is difficult to recognize them from disparate angles.

- Illumination

  Just as with pose variation, illumination variation is inevitable. Ambient lighting changes greatly within and between days and among indoor and outdoor environments. Due to the 3D shape of the face, direct lighting source can cast strong shadows and shading that accentuate or diminish certain facial features. The effect of the illumination change in images can be due to either of two factors, 1) the inherent amount of light reflected off of the skin and 2) the non-linear adjustment in internal camera control, such as gamma correction, contrast, and exposure settings. Both can have major effects on facial appearance. While the latter is less noticeable for humans, it can cause major problems for computer vision.

- Expression

  The face is a non-rigid object. Facial expression of emotion and paralinguistic communication along with speech acts can and do produce large variation in facial appearance. Because facial expression affects the apparent geometrical shape and position of the facial features, the influence on recognition may be greater for geometry based algorithms than for holistic algorithms.

- Occlusion

  The face may be occluded by other objects in the scene or by sunglasses or other paraphernalia. Occlusion may be unintentional or intentional. Under some conditions subjects may be motivated to thwart recognition efforts by covering portions of their face.

- Inter session variability
  Faces change over time. There are changes in hair style, makeup, muscle tension and appearance of the skin, presence or absence of facial hair, glasses, or facial jewelry, and over longer periods effects related to aging.

The algorithm used for this project ([6]) uses a component based approach. Each face is detected based on the detection of three components, namely the left eye, the right eye and the lower face. The three components must adhere to geometrical location constraints. The model of the face is parameterized by the average of the position of the components and the covariance matrices for each component. The location of each component is determined by using a modified AdaBoosting cascade.

The full article of the face detection algorithm can be found in Appendix D

Chapter 3

Project Implementation

3.1 Project overview

A diagram of the recognition process is depicted in the Figure 3.1:

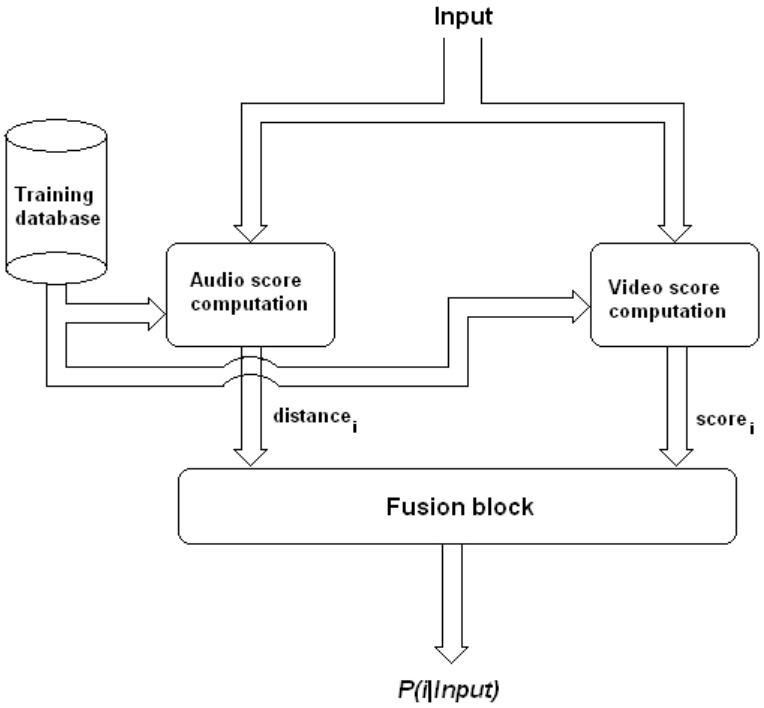

Figure 3.1: System architecture

The data is acquired automatically with a Logitech QuickCam Pro 4000 webcam, connected to the USB port of the computer.

The audio and video recognitions are performed by separate algorithms.

The output of the audio block is a set of distances, so the best score will be the lowest number, while the output of the video block is a set of scores, the higher the score, the better.

The scores are combined in a statistical manner using the late fusion principle.

A set of posterior probabilities is computed. The identified user is chosen as the one having the highest computed posterior probability.

3.2 Audio Processing

On the audio side we used a variable-text text-dependent speaker recognition system, based on the dynamic programming algorithm.

The text is a password made up of digits, and it is variable because it is randomly generated at runtime.

The training database is made of templates for each digit, recorded for each user. This is done automatically by our recording software. The user has to let the system know what digit (s)he intends to record. Then the user utters the digit and the system records the data. After stopping the recording, the program performs a pre-processing and features extraction. These two operations are presented in the following sub-sections. The features will be saved as text files, along with the processed sound file that generated them.

The steps taken for identifying a speaker are the following:

- Generation of a random password
- Recording the input from the user
- Signal pre-processing
- Feature extraction
- Audio distance generation

3.2.1 Random password generation

The user has to read the password from the screen. The password is a random number generated by the system each time identification is performed. The password length can be changed at runtime. As described later, a silence period of approximately 300ms has to be included at the beginning of the recorded password. This is naturally accomplished by showing the password after beginning the recording of the audio data. This way, the user has to first acknowledge the password and then read it.

3.2.2 Sound pre-processing

After the recording, the sound wave has to be pre-processed. The pre-processing consists in three steps:

- amplitude normalization
- denoising
- waveform trimming

The recording was performed at 16.000 Hz and the processing was made on windows of length 256 samples, which is 16 ms. We used the Hamming window and a step of 160 samples.

The waveform was normalized to 80% of the maximum value.

The denoising was performed with a Wiener filter using a noise estimation based on Voice Activity Detection.

The waveform trimming is necessary because it diminishes the computing time of the score computation algorithm and it also improves accuracy because unwanted silence doesn't have to be compared to useful signal.

The wave form trimming is based on a simple and fast algorithm presented in [1] (see Appendix B). The original algorithm is able, in certain conditions, to find the correct beginning and ending of an utterance, when the sound is recorded in noisy conditions. The algorithm uses the signal energy and the zero-crossing rate. The energy level and the zero-crossing average and dispersion of the noise are estimated from the first frames of the input signal. The first part of the algorithm determines the temporary beginning and ending of an utterance, based on the energy levels of the signals. For this, there are two thresholds used. The first threshold (ITL) is chosen based on the maximum values of the signal and noise energy. The second threshold (ITH) is a multiple of the first threshold. The beginning of the utterance is considered to be the first point that has the energy level above the first threshold, and the energy level will reach the second threshold before it goes back below the first threshold. The rational for doing this is to eliminate mouth clicks; however this processing may skip some weak fricatives. The temporary ending point is determined in a similar manner.

Based on these temporary points, the true beginning and ending of the speech are computed using the signal zero-crossing rate. This step is necessary because of its ability to include weak fricatives at the beginning and ending of the utterance (like /f/ in four or /v/ in five).

The algorithm as described above was not meant to work with multiple utterances in the same input, like a password with multiple digits. The difficulty when working with multiple digits consists in the fact that some digits have the peak energy higher than other digits. This might lead to the fact that some digits might be entirely skipped.

But by adjusting these parameters, we were able to adapt the algorithm to trim passwords made of multiple digits with a sufficient accuracy. A safety margin can be included because the algorithm that computes the scores can cope with extra silence periods. Also, by using the denoising algorithm, we were able to use the algorithm for inputs that had -6dB input SNR.

The algorithm can also set a threshold for the minimum value of the zero-cross rate computation. We encountered situations when the value of the zero-cross rate has abnormal high values for silence periods. These can be ignored if a threshold for the energy is set, such that if the energy of the time frame is lower than the threshold, the zero-cross rate will be set to zero. The threshold is given as a ratio of the ITL value.

Parameters can be adjusted at run-time by opening a configuration dialog.

A representation of the waveforms for the password '4866', at each step of the processing can be seen in Figures 3.2, 3.3 and 3.4.

3.2.3 Features extraction

The features extracted are Mel Frequency Cepstral Coefficients (MFCC).

A set of MFCCs is computed for each window (time frame) of 256 samples.

Before computing the Fourier Transform, an emphasis filter is applied on the input signal, if the respective option is selected.

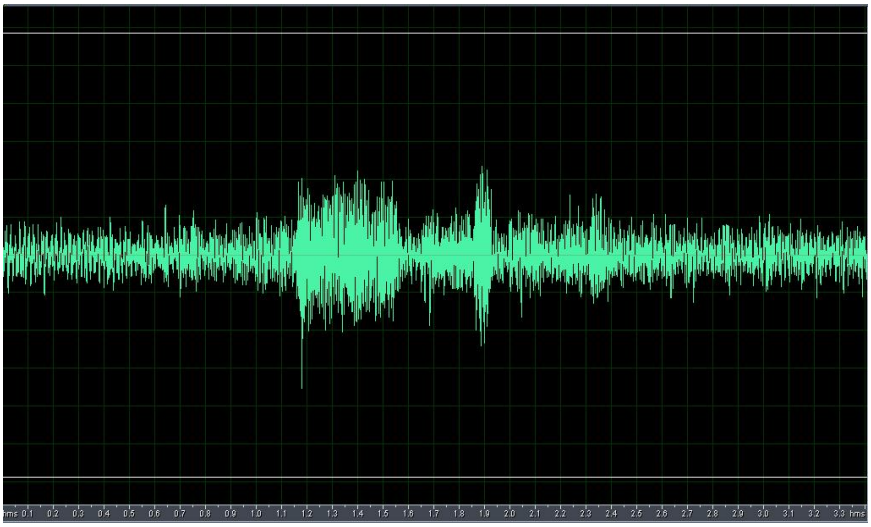

Figure 3.2: Raw wave file

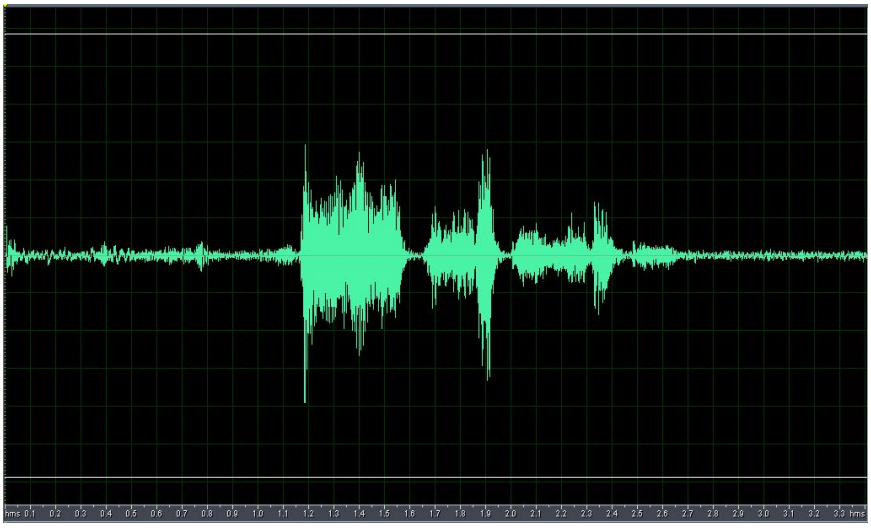

Figure 3.3: Denoised wave file

The centers of the bandwidth filters are computed based on the formula ([2]):

$$mel\_frequency(f) = 2595 \cdot \lg\left(1 + \frac{f}{700.0}\right) \tag{3.1}$$

Based on this formula, 40 filters logarithmically spaced between 200Hz and 8000Hz are computed.The filters are computed so that they have the same total log energy. Figure 3.5 shows the distribution of filters, on 128 points.

The coefficient for the window starting at sample n and ending at sample m is computed using the formula 2.6:

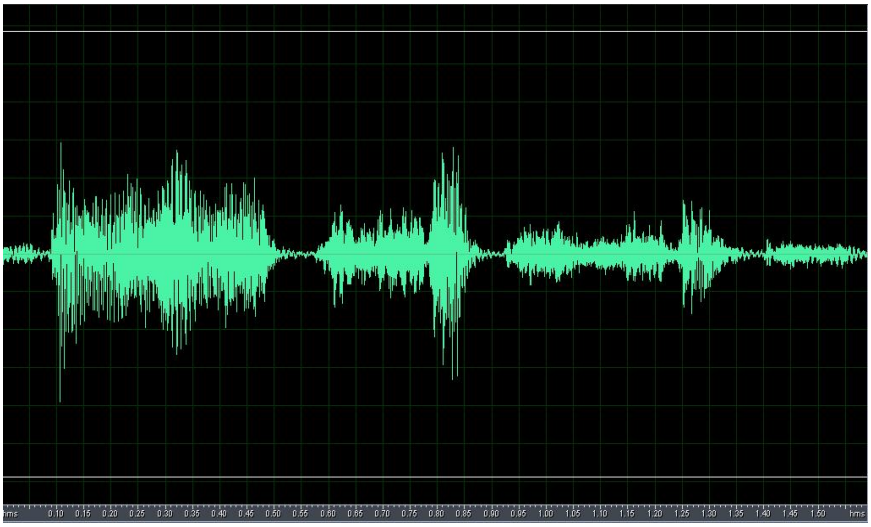

Figure 3.4: Trimmed wave file

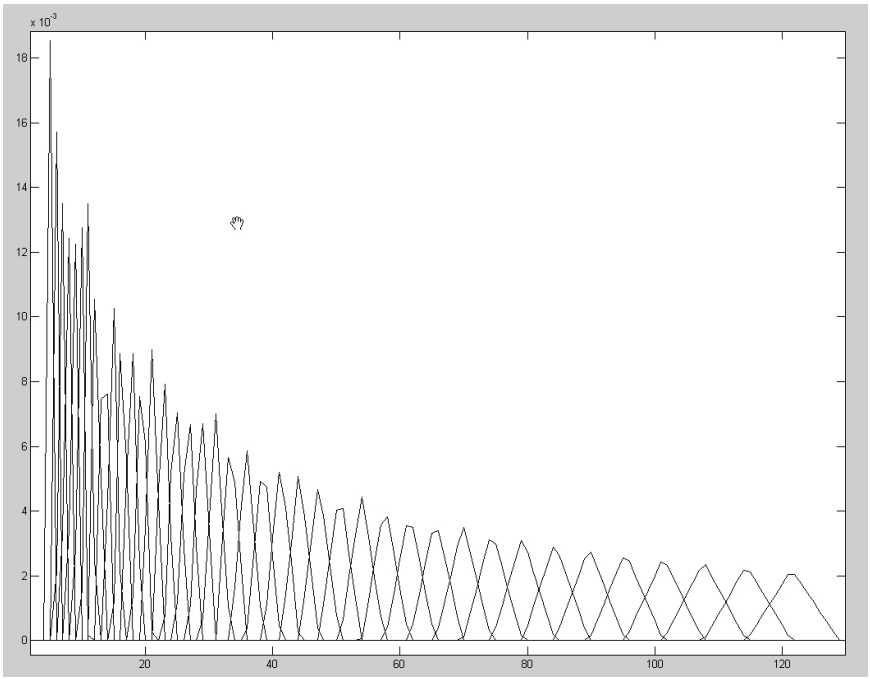

Figure 3.5: Filter amplitude vs. log scale frequency

$$c(n,m) = \sum_{i=1}^{N_{filters}} \tilde{Y}(k_i) \cdot cos\left(k_i \cdot \frac{2\pi}{N} \cdot n\right), \tag{3.2}$$

where :

k_i - index of the center frequency of filter i

N - number of points used for the computation of DCT

$$\tilde{Y}(k) = \begin{cases} \sum_{k=0}^{N/2} log|S(k;m)| \cdot H_i\left(k \cdot \frac{2\pi}{N'}\right), & k = k_i \\ 0, & \text{other } k \in [0, N'-1] \end{cases} \tag{3.3}$$

3.2.4 Audio distance generation

The distance generation is based on the one-pass dynamic-programming algorithm with multiple templates and inter-word silence templates [4].

After computing the MFCCs for the input password, the system will match these features with the features of each user from the database.

If the password is for example "4866", the system will create a set of features made of the templates of the digits 4, 8, and two times the templates of digit 6. Between all the templates of two digits, in the beginning and in the end, the system inserts a silence template. The silence template is taken from the beginning of the recorded input, in order to have a better approximation of the silence model. Because the denoising algorithm introduces noise until all the parameters are estimated, we made it possible to have a silence offset. The silence data will be extracted after the silence offset. Both the silence offset and silence data lengths are configurable from the configuration dialog.

The number of templates for each user and digit doesn't have to be the same, although having an increased number of in the database could increase the accuracy of the recognition. Especially if the templates are recorded at different moments in time, the recognition will be improved because the training will have a better approximation of the voice variability for that user. The one pass dynamic programming algorithm will compute at each time step of the input the distance between the current set of features and all the features of all the digits of the training data. The distance between each two feature sets, from the input and from the training database, is computed using a weighted Euclidean distance. The weights are read from a file when the program starts. The details and reasons for using this method can be found in section 3.5.1.

After computing the distances for all the windows of the test input, the algorithm will look at the accumulated distance in the last frame of all the templates of the last digit. The point that has the smallest accumulated distance will be the one chosen as the end of the path. If wanted, the generated path can be saved.

The accumulated distance in the last point is then divided by the number of frames of the input and this is the computed distance for a specific user.

The one-pass dynamic programming algorithm will be run for all the users that are registered in the database. To register a user, its name has to be written in a configuration file. Not all the users from the database have to be registered, such that only a few of them can be selected from the database. The list of speakers for which the algorithm will be used can be changed at runtime, so, if at some point other discrimination techniques will be used, the algorithm will be faster by not having to compute the distances for everyone in the database.

At the end of the score generation algorithm there will be a matching distance between the input and all the selected speakers. The vector that holds these values will be passed on to the score fusion block.

3.3 Video processing

The video processing is performed by the Binglong Xie's algorithm (see [6]).

Using this algorithm implies two processes:

- video database creation
- video recognition

3.3.1 Video database creation

The algorithm requires a database with the images of the faces of the users.

The user registration is performed by recording a compressed movie from which the faces will be extracted. As it will later be presented, the video sequence necessary for the identification is recorded at the same time as the audio data is recorded. Thus it is recommended that the user speaks the digits from 0 to 9 while recording the training pictures.

The movie is created automatically by a database creator program. The pictures are acquired from the USB camera at intervals of 250 ms. The pictures are recorded as 24 bit color bitmaps. After recording enough pictures, a compressed divx movie is created using the recorded pictures. The movie is passed to an external program that is able to perform a face extraction, based on the user inputs. The user has to select three points, corresponding to the left eye, the right eye and the center of the mouth. The program will save two pictures that contain the user face. One of the pictures is a normalized face that has 64x64 pixels and it is compressed such that the entire face will fit in the square. The other image is also 64 by 64 pixels, but it is not normalized. Both of the faces are saved as 8 bit gray scale bitmaps. The difference between the two images is that the one normalized will include the chin of a user that has a long face, while the normal picture will only cut the square of 64 by 64 pixels, the picture appearing as trimmed. We saved the normal face because we wanted to use the length of the face as a feature, in order to increase the accuracy of the recognition.

The faces extracted from the video are then copied in the user folder. But in order to be able to include that user in the recognition process, the entire database has to be retrained. This is done automatically by a function that searches all the picture folders of all the users in the database and makes a list with all the picture files. This list is then forwarded to another program that will output two training files for the face recognition algorithm. The two files are copied in the end in the database root folder.

The two files needed for face recognition and another training file needed for the face detection are the only files required for the initialization and functioning of the video recognition algorithm.

3.3.2 Score generation

In order to obtain a score, the user that needs to be recognized has to record a set of test pictures. These are recorded simultaneously with the audio data, so it is better if the training pictures for that user are recorded while the user speaks the digits from 0 to 9.

The test data is recorded as 8 bit gray scale pgm images. The recording resolution is 320x240 pixels. The files are saved in the pictures folder and after the capture ends, their names are passed to the recognition algorithm as a parameter.

The recognition algorithm has two phases:

- detecting an image in a face;

- performing recognition of the detected face;

The face detection is ensured by the detection module. This needs a training file for the face detection that is found in the main program folder. In the initialization of the detection module there can be specified different configuration options. Since we are interested in only detecting a person, the limit of the detected faces was set to one. The detection accuracy parameter was set to 10000.

Another option is to set the preparation level of the image. We used the moderate image preparation, which generates 9 pictures instead of only one, by moving the frame of the detected image one pixel at the time in 8 directions (up, down, left, right and diagonals).

The video recognition module receives the 9 images generated by the face detection algorithm and adds them to an internal buffer, on which it performs the recognition. The algorithm only works on a stream of data, and not on separate still images. So we had to configure the algorithm so it receives a number of pictures before outputting a score. A score is outputted every N valid frames (that have a face detected) have been received. The stream size was set to 10, so we would receive a score every ten valid frames. The user with the highest score is considered the recognized user.

The output from the recognition module did not include all the users from the database. So if we sent enough frames to have the algorithm give three sets of scores, we would not get every time a score for each user, or even the same users. So the algorithm had to look at the users recognized each time a score was outputted. If the recognition module receives more sets of pictures, the outputs of the consecutive recognitions are added to the previous obtained scores. In order to obtain a normalized score, we set the video capture function to be able to record only a certain number of pictures before stopping.

The stream detection method has the disadvantage that if we send a number of pictures to the algorithm we cannot make sure that all the pictures sent will have a face detected. This lead sometimes to the situation of not receiving an output from the recognition module. In order to avoid that, we reconfigured the video data capture module so it would save only the pictures that had a face recognized by the recognition module.

In the end, the video recognition module receives 10 frames that have a face detected and outputs a score every 10 frames, so we get one set of scores. If a user was recognized in at least one picture, he will have a non-zero score. Before passing the results to the sensor fusion module, the scores are divided by the number of pictures that had a face detected.

3.4 Audio and Video Score Fusion

Based on the audio and video scores obtained from the two classifiers we tried to obtain a posterior probability for each registered user.

We start by considering the Bayes relation between the probabilities:

$$P(i|Audio, Video) \propto P(Audio, Video|i) \cdot P(i), \tag{3.4}$$

where $P(i|Audio, Video)$ is the posterior probability, $P(Audio, Video|i)$ is the likelihood and $P(i)$ is the prior probability for user i.

Our first hypothesis is that likelihoods of audio and video for an user are independent:

$$P(Audio, Video|i) = P(Audio|i) \cdot P(Video|i) \tag{3.5}$$

The rational for this hypothesis is that, given the user, the distribution of voice signal is independent upon a particular realization of the user image, and vice versa. This allows us to write:

$$P(i|Audio, Video) \propto P(Audio|i) \cdot P(Video|i) \cdot P(i) \tag{3.6}$$

Further:

$$P(i|Audio, Video) = C \cdot P(Audio|i) \cdot P(Video|i) \cdot P(i) = \frac{C}{P(i)} \cdot P(Audio, i) \cdot P(Video, i) \tag{3.7}$$

$$P(i|Audio, Video) = \frac{C}{P(i)} \cdot P(i|Audio) \cdot P(i|Video) \cdot P(Audio) \cdot P(Video) \tag{3.8}$$

where C is given by:

$$C = \frac{1}{P(Audio, Video)} \tag{3.9}$$

So the previous relation becomes:

$$P(i|Audio, Video) = \frac{P(i|Audio) \cdot P(i|Video)}{P(i)} \cdot \frac{P(Audio) \cdot P(Video)}{P(Audio, Video)} \tag{3.10}$$

The second fraction of this equation is a constant that we denote by D, that can be determined from the normalization of the posterior probability:

$$1 = \sum_i P(i|Audio, Video) = D \cdot \sum_i \frac{P(i|Audio) \cdot P(i|Video)}{P(i)} \tag{3.11}$$

We obtain:

$$D = \frac{1}{\sum_j \frac{P(j|Audio) \cdot P(j|Video)}{P(j)}} \tag{3.12}$$

which makes the posterior probability to take the following form:

$$P(i|Audio, Video) = \frac{P(i|Audio) \cdot P(i|Video)/P(i)}{\sum_j P(j|Audio) \cdot P(j|Video)/P(j)} \tag{3.13}$$

So in order to obtain the posterior probability of user i given audio and video data, we must compute both the posterior probability given audio data, and the posterior probability given audio data, and then multiply them with the prior probability and then normalize the result so that the sum of the posterior probabilities equals 1.

Our second hypothesis concerns the sufficient statistics of the distances and scores as computed by the two expert systems (classifiers):

$$P(i|Audio) = P(i|dist_1, dist_2, \ldots, dist_N) \tag{3.14}$$

and

$$P(i|Video) = P(i|score_1, score_2, \ldots, score_N) \tag{3.15}$$

where N is the number of registered users.

In other words, our hypothesis is that the posterior distribution depends only on the outputs of the two classifiers.

Our third hypothesis concerns the prior distribution. We assume that the priors P(i) are equal for the registered users. This hypothesis can be easily relaxed when more information is available.

Different models used for shaping the posterior probabilities will be described in the following sub-sections.

3.4.1 An Exponential Model with Common Rate

The first model we tried for the likelihood was an exponential model:

$$P(dist_1, dist_2, \ldots, dist_N|i) = exp(-\alpha \cdot dist_i), \tag{3.16}$$

The audio posterior probability becomes:

$$P(i|dist_1, dist_2, \ldots, dist_N) = C \cdot exp(-\alpha \cdot dist_i), \tag{3.17}$$

where α is a common coefficient for all the users and C is a constant.

The constant C can be computed from the condition of normalization of the posterior probability and we obtain:

$$C \cdot \sum_j exp(-\alpha \cdot dist_j) = 1, \tag{3.18}$$

and

$$C = \frac{1}{\sum_j exp(-\alpha \cdot dist_j)} \tag{3.19}$$

So the audio posterior probability is:

$$P(i|dist_1, dist_2, \ldots, dist_N) = \frac{exp(-\alpha \cdot dist_i)}{\sum_j exp(-\alpha \cdot dist_j)} \tag{3.20}$$

In a similar way, we write the video likelihood and posterior probability respectively:

$$P(score_1, score_2, \ldots, score_N|i) = exp(\beta \cdot score_i), \tag{3.21}$$

$$P(i|score_1, score_2, \ldots, score_N) = \frac{exp(\beta \cdot score_i)}{\sum_j exp(\beta \cdot score_j)} \tag{3.22}$$

This model was used by taking into consideration that the output of the audio classifier is a set of distances, and the higher the distance is for a user, the smaller should the posterior probability be.

Another thing we took into consideration was that if we have a set of distances and we keep the distance constant for an user k and increase the distances of the other users, the user k should have a higher posterior probability.

The value of the combined audio and video likelihood is:

$$P(A, V|i) = P(A|i) \cdot P(V|i) = \exp(\beta \cdot score_i) \cdot \exp(-\alpha \cdot dist_i) \tag{3.23}$$

$$P(A, V|i) = \exp(\beta \cdot score_i - \alpha \cdot dist_i) \tag{3.24}$$

The values of α and β are given by an optimization criterion. We implemented an algorithm that maximized the ratio between the likelihood of the correct user (k) and the likelihood of the "other best user" (s), which is the user with the highest probability, excepting the correct user. This is expressed in the following way:

$$\frac{P(A, V|k)}{P(A, V|s)} = exp[\beta \cdot (score_k - score_s) - \alpha \cdot (dist_k - dist_s)] \tag{3.25}$$

The optimization criterion can be expressed as maximizing the product of the previous ratios over the entire training set, which includes the training data from all the users:

$$\prod_t \frac{P(A, V|k)}{P(A, V|s)} = exp\left\{\sum_t [\beta \cdot (score_k(t) - score_s(t)) - \alpha \cdot (dist_k(t) - dist_s(t))]\right\} \tag{3.26}$$

where t indexes the training set.

If we take the logarithm of the previous expression, we obtain the following maximization criterion:

$$\max_{\alpha,\beta} \sum_t [\beta \cdot (score_k(t) - score_s(t)) - \alpha \cdot (dist_k(t) - dist_s(t))] \tag{3.27}$$

If we factor α, we obtain:

$$\max_{\alpha,\beta} \sum_t \alpha \cdot \left[\frac{\beta}{\alpha} \cdot (score_k(t) - score_s(t)) - (dist_k(t) - dist_s(t))\right] \tag{3.28}$$

Let γ denote the ratio $\frac{\beta}{\alpha}$. Then α turns into merely a scaling factor. Given these, we obtain the following optimization problem:

$$\max_{\gamma} \sum_t [\gamma \cdot (score_k(t) - score_s(t)) - (dist_k(t) - dist_s(t))] \tag{3.29}$$

In order to find a solution, we needed to know for each t which is the next best user. A simple approach would had been to use a brute force approach and then at each step, find the index of the needed user.

Instead of this approach, we used a more efficient method. This method consists in finding, for each test, the intervals for which only one user dominates the others.

For example, let us suppose we have the following results:

$$testResults = \begin{bmatrix} 50 & 30 \\ 40 & 10 \\ 45 & 15 \\ 45 & 5 \\ 35 & 20 \end{bmatrix} \tag{3.30}$$

The first column represents the video scores, and the second column represents the audio distances. If the training user is the one with index number 4, we obtain the following intervals:

$$\begin{bmatrix} -Inf & -2.0000 & 5.0000 \\ -2.0000 & 1.0000 & 2.0000 \\ 1.0000 & 3.0000 & 3.0000 \\ 3.0000 & Inf & 1.0000 \end{bmatrix} \tag{3.31}$$

This means for example that user with index number 3 has the highest probability if γ is in the $[1; 3]$ interval.

When this method is applied to all training sets, we obtain a number of intervals for which we know the index of the user with the highest probability, for each training set.

Given these intervals, the problem reduces to a number of linear problems, equal with the number of distinct intervals for the γ parameter across all tests. These intervals are sorted in ascending order. For each one of them, the value of the function is computed, and the one that maximizes it is chosen as the optimal value.

The algorithm can apply a penalty for misclassifying the correct user. If for a given test, the true user doesn't have the highest probability, the difference between the two considered users is multiplied with a factor. The higher the value of the penalty, the lower the value of γ.

This approach has the advantage of training only one parameter, so the number of tests needed for the training algorithm does not have to be very large. Another advantage is that the training algorithm adjusts the parameter such that the final posterior probability will be maximized for the correct user.

This method has the disadvantage that the final probabilities were not that well separated. Even the correct user has a low probability. This happened because the parameter for audio probability was not modified.

These finding determined us to look for an alternate approach where both α and β are optimized.

The algorithm alternates between two steps. In the first step the optimal value for α for a fixed β is found. In the second step, for fixed α the optimal value of β is found. In both steps we used a brute force search, given the fact that both the number of training sets and the range of parameters was not very large.

The α parameter was chosen as the value that maximizes the criterion:

$$\hat{\alpha} = \max_{\alpha} \sum_{t} \left(-\frac{P(A|s)}{P(A|k)} \right) = \max_{\alpha} \sum_{t} \left(-\frac{exp(-\alpha \cdot d_s)}{exp(-\alpha \cdot d_k)} \right) \tag{3.32}$$

where t runs over the training set, k is the index of the true speaker and s is the index of the user with the highest probability, other than the true speaker.

The ratio is multiplied with a constant value if the true speaker doesn't have the highest probability. If the value of the penalty is higher, the algorithm will have a cautious behavior and it will output a low value for α. If α is close to 0, all the users will tend to have the same probability. A low value for α is obtained if a relatively high portion of the training set doesn't have the smallest distance associated to the true speaker. If the training set only contains good classifications or a small number of misclassifications (lowest audio distance for the true user), the algorithm will output a high value for α. A very high value for α has the same effect as choosing the minimum value from the result set.

After finding the optimal value for α, the next step was to optimize β. The optimization criterion used was:

$$\hat{\beta} = \max_{\beta} \sum_{t} \left(-\frac{P(A,V|s)}{P(A,V|k)} \right) = \max_{\beta} \sum_{t} \left(-\frac{exp(\beta \cdot score_s - \hat{\alpha} \cdot dist_s)}{exp(\beta \cdot score_k - \hat{\alpha} \cdot dist_k)} \right) \tag{3.33}$$

The results of the second method were better than the one using only one parameter. Also, even though for some tests neither of the classifiers would give the best scores for the true user, the final result can be the right one.

3.4.2 Gaussian Model

In the previous subsection we used both the audio and video results to train the values of the fusion parameters. Another approach we tried was to optimize independently the audio and video probability parameters for all the users. With the resulted parameters, we would then compute the combined audio and video posterior probabilities and select the user with the highest probability.

Another approach was to optimize the parameters independently for each speaker. So, instead of using the training data that belonged to all the users to obtain only one common parameter, we used for each user his specific set of parameters.

In Figure 3.6 we can see the distribution of audio distances obtained by applying the one pass dynamic programming algorithm to the audio data that belongs to one of the speak-

ers. We tried to model each speaker probability based on the distribution of the distances obtained by the entire speaker set on his training data.

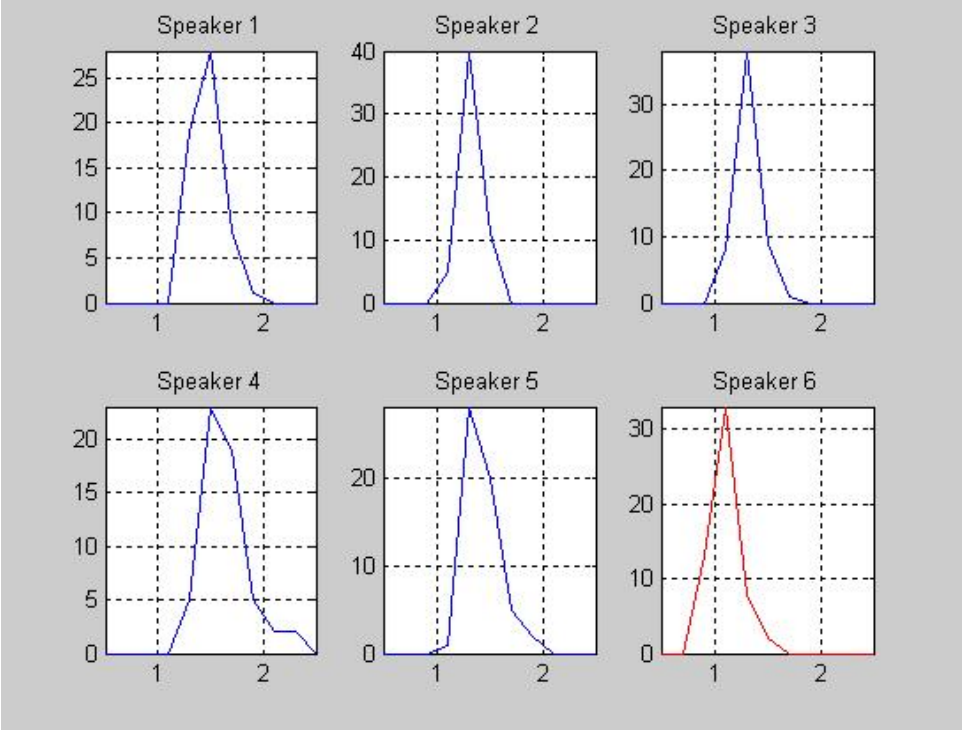

Figure 3.6: Distance distribution for training speaker number 6

We used a constrained multi Gaussian distribution to model a speaker's probability. The function we use is:

$$P(\underline{d}|i) = \begin{cases} C(\underline{m}_{(i)}, \underline{R}_{(i)}) \cdot exp\left(-\frac{1}{2} \cdot (\underline{d} - \underline{m}_{(i)})^T \cdot R_{(i)}^{-1} \cdot (\underline{d} - \underline{m}_{(i)})\right) & \text{when all } d_k > 0 \\ 0 \quad otherwise \end{cases} \tag{3.34}$$

We imposed $R_i = diag(r_1, r_2, \ldots, r_N)$, with N being the number of users. $C(\underline{m}_{(i)}, \underline{R}_{(i)})$ is the normalization constant.

For an unconstrained Gaussian distribution:

$$P(\underline{d}|i) = \frac{1}{\left(\sqrt{2\pi}\right)^N \cdot \sqrt{det(R_{(i)})}} \cdot exp\left(-\frac{1}{2} \cdot (\underline{d} - \underline{m}_{(i)})^T \cdot R_{(i)}^{-1} \cdot (\underline{d} - \underline{m}_{(i)})\right) \tag{3.35}$$

the Maximum Likelihood estimates of parameters r_i, m_i are given by:

$$\hat{m}_j^{(i)} = \frac{1}{T} \cdot \sum_{t=1}^{T} d_j(t) \tag{3.36}$$

$$\hat{r}_j^{(i)} = \frac{1}{T} \sum_{t=1}^{T} (d_j(t))^2 - \left(\frac{1}{T} \cdot \sum_{t=1}^{T} d_j(t)\right)^2 = var(d_j^{(i)}) \tag{3.37}$$

The diagonal matrix $R_i = diag(r_1, r_2, \ldots, r_N)$, has entries that each represents the variance of the data for user i. The vector $\underline{m}_{(i)}$ contains the averages of the training data for every user when the training user is the one with index i.

Using these constraints, the likelihood function for user i becomes:

$$P(\underline{d}|i) = \frac{1}{\left(\sqrt{2\pi}\right)^N \cdot \sqrt{r_1^{(i)} \cdot r_2^{(i)} \ldots \cdot r_N^{(i)}}} \cdot exp\left(-\frac{1}{2} \cdot \sum_{j=1}^{N} \frac{(d_j - m_j^{(i)})^2}{r_j^{(i)}}\right) \tag{3.38}$$

where N is the number of users.

However, when we consider the positivity constraint, the normalization constant $C^{(i)}(\underline{m}, \underline{r})$ no longer equals $(2\pi)^{-N/2}(det\, R)^{-1/2}$. Instead:

$$P(d_j^{(i)}|i) = \begin{cases} c_j^{(i)} \cdot exp\left(-\frac{1}{2 \cdot r_j^{(i)}} \cdot (d_j - m_j^{(i)})^2\right) & d_j^{(i)} \geq 0 \\ 0, & d_j^i < 0 \end{cases} \tag{3.39}$$

The value of c_j^i can be computed from the normalization condition:

$$c_j^{(i)} \cdot \int_0^\infty exp\left[-\frac{1}{2r_j^{(i)}} \cdot \left(x - m_j^{(i)}\right)^2\right] dx = 1 \tag{3.40}$$

$$c_j^{(i)} \cdot \sqrt{r_j^{(i)}} \cdot \int_0^\infty exp\left[-\frac{1}{2} \cdot \left(\frac{x}{\sqrt{r_j^{(i)}}} - \frac{m_j^{(i)}}{\sqrt{r_j^{(i)}}}\right)^2\right] \frac{dx}{\sqrt{r_j^{(i)}}} = 1 \tag{3.41}$$

If we denote:

$$a := \frac{m_i}{\sqrt{r_j^{(i)}}} \tag{3.42}$$

and change the integration variable, we obtain:

$$\int_0^\infty exp\left[-\frac{1}{2} \cdot (u - a)^2\right] du = \frac{1}{c_j^{(i)} \cdot \sqrt{r_j^{(i)}}} \tag{3.43}$$

$$c_j^{(i)} = \frac{1}{\sqrt{r_j^{(i)}}} \cdot \frac{1}{\int_0^\infty exp\left[-\frac{1}{2} \cdot (u - a)^2\right] du} = \frac{1}{\sqrt{r_j^{(i)}}} \cdot \frac{1}{\frac{\sqrt{\pi}}{2} + \sqrt{\frac{\pi}{2}} \cdot erf\left(\frac{a}{\sqrt{2}}\right)} \tag{3.44}$$

where

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z exp(-t^2) dt \tag{3.45}$$

and it is the "error function".

We note the result:

$$f(a) := \int_0^\infty exp\left[-\frac{1}{2} \cdot (u - a)^2\right] du = \frac{\sqrt{\pi}}{2} + \sqrt{\frac{\pi}{2}} \cdot erf\left(\frac{a}{\sqrt{2}}\right) \tag{3.46}$$

The values of r_j^i and m_j^i are obtained from maximizing the likelihood function over the entire training set of user i:

$$\hat{r}, \hat{m} = \underset{r,m}{\operatorname{argmax}} \left\{ \prod_{t=1}^{T} \frac{1}{\sqrt{r}} \cdot \frac{1}{\int_0^{\infty} exp\left[-\frac{1}{2} \cdot \left(x - \frac{m}{\sqrt{r}} \right)^2 \right] dx} \cdot exp\left[-\frac{1}{2r} \cdot (d_t - m)^2 \right] \right\} \tag{3.47}$$

Here, for simplicity of notation we dropped i and j indexing.

This is equivalent to:

$$\hat{r}, \hat{m} = \underset{r,m}{\operatorname{argmin}} \left\{ \frac{1}{r} \cdot \frac{1}{T} \cdot \sum_{t=1}^{T} (d_t - m)^2 + \ln r + 2 \cdot \ln \left(\int_0^{\infty} exp\left[-\frac{1}{2} \cdot \left(x - \frac{m}{\sqrt{r}} \right)^2 \right] dx \right) \right\} \tag{3.48}$$

If we denote:

$$D_1 := \frac{1}{T} \cdot \sum_{t=1}^{T} d_t \tag{3.49}$$

$$D_2 := \frac{1}{T} \cdot \sum_{t=1}^{T} d_t^2 \tag{3.50}$$

we obtain:

$$\hat{r}, \hat{m} = \underset{r,m}{\operatorname{argmin}} \left\{ \underbrace{\frac{D_2}{r} - \frac{2m}{r} \cdot D_1 + \frac{m^2}{r} + \ln r}_{A} + \underbrace{2 \cdot \ln \left(\int_0^{\infty} exp\left[-\frac{1}{2} \cdot \left(x - \frac{m}{\sqrt{r}} \right)^2 \right] dx \right)}_{\Phi(\frac{m}{\sqrt{r}}) = \Phi(a)} \right\} \tag{3.51}$$

The part denoted with A of the minimized function was optimized for $C(\underline{m}, \underline{r}) = \frac{1}{(\sqrt{2\pi})^N \cdot \sqrt{det(R_{(i)})}}$. So we can consider that the function we want to minimize has the same solutions, but slightly perturbed by $\Phi(a)$.

If we make the following notation:

$$J(m, r) = \frac{D_2}{r} - \frac{2m}{r} \cdot D_1 + \frac{m^2}{r} + \ln r_A + \Phi\left(\frac{m}{\sqrt{r}} \right) \tag{3.52}$$

we can apply the Newton method of optimization, using one correction step. We initialize the solution with:

$$m = D_1 \tag{3.53}$$

$$r = D_2 - D_1^2 \tag{3.54}$$

We consider γ and δ small variations around the initial solution.

$$J(\hat{m} + \gamma, \hat{r} + \delta) = J(\hat{m}, \hat{r}) + \begin{bmatrix} \frac{\partial J}{\partial m} & \frac{\partial J}{\partial r} \end{bmatrix} \cdot \begin{bmatrix} \gamma \\ \delta \end{bmatrix} + \tag{3.55}$$

$$+ \frac{1}{2} \cdot \begin{bmatrix} \gamma & \delta \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial^2 J}{\partial m^2} & \frac{\partial^2 J}{\partial r \partial m} \\ \frac{\partial^2 J}{\partial m \partial r} & \frac{\partial^2 J}{\partial r^2} \end{bmatrix} \cdot \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \tag{3.56}$$

The values of the partial derivates are the following:

$$\frac{\partial J}{\partial m} = \frac{1}{\sqrt{r}} \cdot \Phi'\left(\frac{m}{\sqrt{r}} \right) \tag{3.57}$$

$$\frac{\partial J}{\partial r} = -\frac{m}{2r\sqrt{r}} \cdot \Phi' \left(\frac{m}{\sqrt{r}} \right) \tag{3.58}$$

$$\frac{\partial^2 J}{\partial m^2} = \frac{2}{r} + \frac{1}{r} \cdot \Phi'' \left(\frac{m}{\sqrt{r}} \right) \tag{3.59}$$

$$\frac{\partial^2 J}{\partial r^2} = \frac{2}{r^3} \cdot \left(D_2 - 2\hat{m} D_1 + m^2 \right) - \frac{1}{r^2} + \frac{3m}{4r^2\sqrt{r}} \cdot \Phi' \left(\frac{m}{\sqrt{r}} \right) + \frac{m^2}{4r^3} \cdot \Phi'' \left(\frac{m}{\sqrt{r}} \right) \tag{3.60}$$

$$\frac{\partial^2 J}{\partial r \partial m} = \frac{\partial^2 J}{\partial m \partial r} = \frac{2D_1}{r^2} - \frac{2m}{r^2} - \frac{1}{2r\sqrt{r}} \cdot \Phi' \left(\frac{m}{\sqrt{r}} \right) - \frac{m}{2r^2} \cdot \Phi'' \left(\frac{m}{\sqrt{r}} \right) \tag{3.61}$$

The values of $\Phi' \left(\frac{m}{\sqrt{r}} \right)$ and $\Phi'' \left(\frac{m}{\sqrt{r}} \right)$ can be computed by observing that

$$\Phi(a) = 2 \cdot \ln(f(a)) = 2 \cdot \ln \left[\frac{\sqrt{\pi}}{2} + \sqrt{\frac{\pi}{2}} \cdot erf \left(\frac{a}{\sqrt{2}} \right) \right] \tag{3.62}$$

We obtain the functions:

$$\Phi'(a) = \frac{2}{f(a)} \cdot f'(a) = \frac{2}{f(a)} \cdot \sqrt{\frac{\pi}{2}} \cdot \sqrt{\frac{2}{\pi}} \cdot erf' \left(\frac{a}{\sqrt{2}} \right) = \frac{1}{\frac{\sqrt{\pi}}{2} + \sqrt{\frac{\pi}{2}} \cdot erf \left(\frac{a}{\sqrt{2}} \right)} \cdot exp \left(-\frac{a^2}{2} \right) \tag{3.63}$$

$$\Phi''(a) = -\frac{2}{f^2(a)} \cdot exp \left(-\frac{a^2}{2} \right) - \frac{2a}{f(a)} \cdot exp \left(-\frac{a^2}{2} \right) \tag{3.64}$$

We denote:

$$M := \begin{bmatrix} \frac{\partial^2 J}{\partial m^2} & \frac{\partial^2 J}{\partial r \partial m} \\ \frac{\partial^2 J}{\partial m \partial r} & \frac{\partial^2 J}{\partial r^2} \end{bmatrix} \tag{3.65}$$

$$V^T := \begin{bmatrix} \frac{\partial J}{\partial m} & \frac{\partial J}{\partial r} \end{bmatrix} \tag{3.66}$$

Using these notations, we obtain:

$$[\hat{\gamma}, \hat{\delta}] = \underset{\gamma, \delta}{\operatorname{argmin}} \left\{ J(m, r) + V^T \cdot \begin{bmatrix} \gamma \\ \delta \end{bmatrix} + \frac{1}{2} \cdot \begin{bmatrix} \gamma & \delta \end{bmatrix} \cdot M \cdot \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \right\} \tag{3.67}$$

When we take the derivative of the criterion to find the stationary point, $J(m, r) = 0$ because we assumed that the starting point was a solution.

The minimum of the criterion is reached for:

$$V + M \cdot \begin{bmatrix} \hat{\gamma} \\ \hat{\delta} \end{bmatrix} = 0 \tag{3.68}$$

So the values of the adjustments needed because of introducing the normalization constant $c_j^{(i)}$ is:

$$\begin{bmatrix} \hat{\gamma} \\ \hat{\delta} \end{bmatrix} = -M^{-1} \cdot V \tag{3.69}$$

with M and V defined above.

Experimentally, we obtained values for the adjustments ranging from 10^{-7} to 10^{-22}, which means that introducing the scaling factor $C(m, r)$ we do not obtain a significant change in the values of the trained parameters.

This method was only tested on the audio probability computation.

The values of the probabilities obtained for one user were very close to unity. Compared to the method of taking the minimum of the distances, there were good parts and also bad parts. The good thing is that this method was able to classify correctly some of the cases when the minimum distance did not belong to the true speaker. The disadvantages of the method were the sensibility to outliers and that even if the result was incorrect, the probability associated to the incorrect user was very close to unity. We would have wanted the values of the probabilities obtained in the case of a misclassification to be similar between the true speaker and the one classified as the recognized speaker.

3.4.3 An Exponential Model With Individual Rate

One disadvantage of the Gaussian model is that results are not always similar to the ones obtained by taking the minimum of the audio distances. By this, we mean that the users that had the first three smallest distances were not always among the users with the 3 highest probabilities.

So we wanted to include a higher dependency of the probabilities on the order of distances obtained from the one pass dynamic programming algorithm. This can be accomplished by modeling the probabilities of the difference between each distance in the set and their minimum instead of individual distances.

The distribution of the difference of distances can be seen in Figure 3.7.

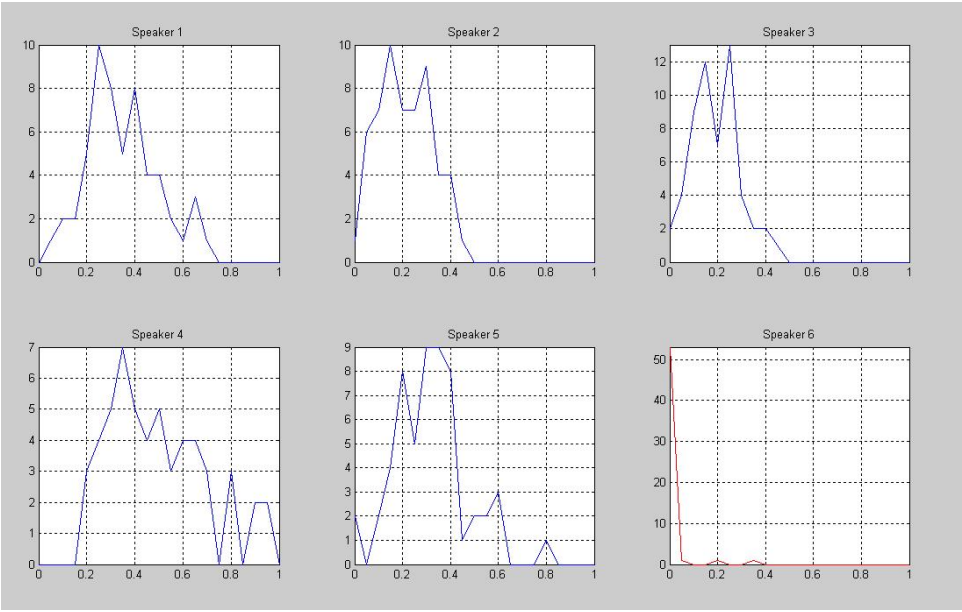

Figure 3.7: Distribution of $(d_i(t) - \min_j d_j(t))$ for training speaker number 6

We will begin by working on the audio likelihood, and then we will write down in a similar

way the likelihood for video data.

The function we chose to model the audio data likelihood is:

$$P(d_1, \ldots, d_N|i) = \lambda_i \cdot exp\left[-\lambda_i \left(d_i - \min_j(d_j)\right)\right] \tag{3.70}$$

The associated posterior probability is:

$$P(i|d_1, \ldots, d_N) = \frac{\lambda_i \cdot exp\left[-\lambda_i \left(d_i - \min_j(d_j)\right)\right]}{\sum_{k=1}^{N} \lambda_k \cdot exp\left[-\lambda_k \left(d_k - \min_j(d_j)\right)\right]} \tag{3.71}$$

Each user will have his own trained coefficient, based on the training data provided.

If the distance obtained by the user i, the true speaker, is the minimum distance in the set, then the likelihood will have the maximum value, which is λ_i.

If the distance is not the minimum, the value of the likelihood will decrease, depending on the value of λ_i and the value of the difference between the two distances.

The training of the parameters is performed using Maximum Likelihood Estimation. The training data consists in the sets of scores obtained by applying the one pass dynamic programming algorithm on the passwords spoken by the tested user. The number of tests will be denoted with T.

The MLE function is:

$$\hat{\lambda_i} = \underset{\lambda_i}{\operatorname{argmax}} \left\{ \prod_{t=1}^{T} P(d_1, \ldots, d_N|i) \right\} \tag{3.72}$$

$$\hat{\lambda_i} = \underset{\lambda_i}{\operatorname{argmax}} \left\{ \lambda_i^T \cdot exp\left[-\lambda_i \cdot \sum_{t=1}^{T} \left(d_i(t) - \min_j d_j(t)\right)\right] \right\} \tag{3.73}$$

If we take the log-likelihood of the previous function, we obtain:

$$\Phi(\lambda_i) = T \cdot \ln \lambda_i - \lambda_i \cdot \sum_{t=1}^{T} \left[d_i(t) - \min_j \left(d_j(t)\right)\right] \tag{3.74}$$

By taking the derivative of the $\Phi(\lambda_i)$ function, we obtain:

$$\Phi'(\lambda_i) = \frac{T}{\lambda_i} - \sum_{t=1}^{T} \left[d_i(t) - \min_j \left(d_j(t)\right)\right] \tag{3.75}$$

So the optimal value of λ_i is:

$$\lambda_i = \frac{T}{\sum_{t=1}^{T} \left[d_i(t) - \min_j \left(d_j(t)\right)\right]} \tag{3.76}$$

The results of the tests were characterized by the fact that the highest probability obtained for each test was very close to unity. The problem was that this was true also for the cases when the classification was incorrect. Also, compared to the method of taking the minimum value, the new method had a higher number of misclassification. For the tests we used, there was no case when this method would give the true speaker if he didn't have the lowest score.

Analyzing the data, we observed that the values obtained for the parameters were very high (between 100 and 250). As we discussed in section 3.4.1, having high values makes the classifier behave like choosing the true speaker the one with the smallest distance from the set. The values were "over saturated".

We then tried to lower the values of the parameters, by dividing them with a constant. Although the number of misclassification was still higher than taking the minimum value, there were cases when the algorithm was finding the true speaker even if he didn't have the lowest distance in the set. Also, by lowering the values, the probability difference between the correct and the recognized users was lower in the case of a misclassification.

Because of these disadvantages, we had to change the optimization method. The new method is based on optimizing the probability of the true speaker relative to the user that has the highest probability in that test, excepting the training speaker.

The differences from the algorithm presented in 3.4.1 are the following:

- Each user has its own parameter that is trained, based on the training data provided to the algorithm;
- The likelihood is not only an exponential – the λ_i parameter appears in front of the exponential too;
- The optimization is performed on the differences between the distances and the minimum distance in the set.

In order to find the values of the parameter for user *i*, we try to maximize the function:

$$\Phi(\lambda_i) = \sum_{t=1}^{T} -\frac{P(A|s)}{P(A|k)} \cdot \psi(d_i, \min_j (d_j)) \tag{3.77}$$

where $\psi(d_i, \min_j (d_j))$ is a function that penalizes the cases when the true speaker doesn't have the highest probability.

If we replace the likelihoods with their formulas we obtain:

$$\Phi(\lambda_i) = \sum_{t=1}^{T} -\frac{exp\left(\lambda_s \cdot [d_s(t) - \min_j (d_j(t))] + \ln \delta \cdot \mathbf{1}_{d_i(t) > \min_j (d_j(t))}\right)}{exp\left(\lambda_i \cdot [d_i(t) - \min_j (d_j(t))]\right)} \tag{3.78}$$

where δ is a penalty factor that penalizes the cases when the true speaker doesn't have the highest probability.

The function can be rewritten as:

$$\Phi(\lambda_i) = \begin{cases} \sum_{t=1}^{T} -\delta \cdot \frac{exp(\lambda_s \cdot [d_s(t) - \min_j (d_j(t))])}{exp(\lambda_i \cdot [d_i(t) - \min_j (d_j(t))])} & , d_i(t) \neq \min_j d_j(t) \\ \sum_{t=1}^{T} -\frac{exp(\lambda_i \cdot [d_i(t) - \min_j (d_j(t))])}{exp(\lambda_s \cdot [d_s(t) - \min_j (d_j(t))])} & d_i(t) = \min_j d_j(t) \end{cases} \tag{3.79}$$

If $\delta = 1$, then the algorithm will treat in the same way the correct and the incorrect classifications. If $\delta < 1$, the algorithm will emphasize the good results and will output a higher value for λ_i. Finally, if $\delta > 1$, the algorithm will tend to emphasize the misclassifications, outputting a lower value for λ_i.

The optimal value for λ_i is:

$$\hat{\lambda_i} = \underset{\lambda_i}{\operatorname{argmax}} \{\Phi(\lambda_i)\} \tag{3.80}$$

The approach used for finding the optimal value for λ_i was the brute force. The range used for the search was between 1 and 30.

During the tests there were some users that were correctly classified in most of their training sets and the algorithm was giving a high value for λ. The value was most of the times the upper limit of the search interval, because the cost function was not decreasing fast enough because of the penalties. On the other extreme, there were users that didn't have so many tests with them having the lowest distance in the set. In their case, the algorithm was outputting a low value for λ. So on one side there were users that had very high values for λ, and others that had very low values for λ. In a situation where the correct user would be the one with a small λ, but another user with a large λ would have a similar score, most of the times the algorithm would give an incorrect result.

To avoid this unpleasant fact, we restricted the search interval for the optimal value of λ. In order to get the lower limit, we chose a higher value (2) for the penalty, and the training was performed on the entire data set. Sending the entire data set ensured an averaging of the bad results over the entire set of users. For the upper limit of the search interval the value of the penalty was less than unity (0.85).

After finding $[\lambda_{min}, \lambda_{max}]$, we applied the optimization algorithm for each user with their specific train data.

The same optimization algorithm can be applied for the video probability if we model the video likelihood with the function:

$$P(score_1, \ldots, score_N|i) = \gamma_i \cdot exp\left[-\gamma_i \left(\max_j(score_j) - score_i\right)\right] \tag{3.81}$$

The algorithm was able in some cases to classify correctly the users that didn't have the lowest distance in their set. Also, in the case when the true speaker didn't have the highest probability, the difference between his probability and the highest probability had relatively low values.

3.5 Audio Recognition Algorithm Improvements

During the time the program was developed there were some attempts to optimize the audio processing part.

These optimizations refer to:

- Using different weights for the MFCC coefficients when computing the Euclidean distance in the one pass dynamic programming algorithm;
- Finding the best range of MFCC coefficients to use in the one pass dynamic programming algorithm;
- Retrieving the accumulated distance along each coefficient used in the one pass dynamic programming algorithm and trying to get a set of projections that would maximize the distance between the different users.

3.5.1 Using the Weighted Euclidean Distance

When the audio processing algorithm was first developed, the distance computation used between two feature sets was the Euclidean distance. But this method has a disadvantage due to the values and the variations of the different coefficients. For example, the variance of the first order coefficient can be at least 100 times greater than the variance of the fifth order coefficient. Normally, this would be good, because it would mean that the zero order coefficient is a good discriminator between the users. The problem is that the average of the coefficient is a lot higher than the average of the other coefficients, but the trajectories

remain similar. So, in the end, the trajectory will only be dictated by a few coefficients, in particular the ones that have a low order.

In order to show that, we have generated a test program that computes coefficient wise the distances between the templates in the database. This is done by running the one pass dynamic programming algorithm on all the different combinations of two templates of the same digit from the database, for all users. The warping path is generated using all the coefficients when computing the Euclidean distance. But along with the value of the distance between the two feature sets, the algorithm will also keep track of the accumulated distances between the individual coefficients. The accumulated distances represent the sum of the squared differences between the coefficients, along the alignment path.

In the end we obtained two files, one for the inter-speaker variability, and another one for intra-speaker variability. The inter-speaker variability output contains all the combinations of comparisons between two templates of the same digit that belong to different users. The intra-speaker variability file contains all the different combinations of comparisons between two templates of the same digit that belong to the same user. With these results, we have computed the average distance of each coefficient. The results are synthesized in the graphics from Figures 3.8 and 3.9.

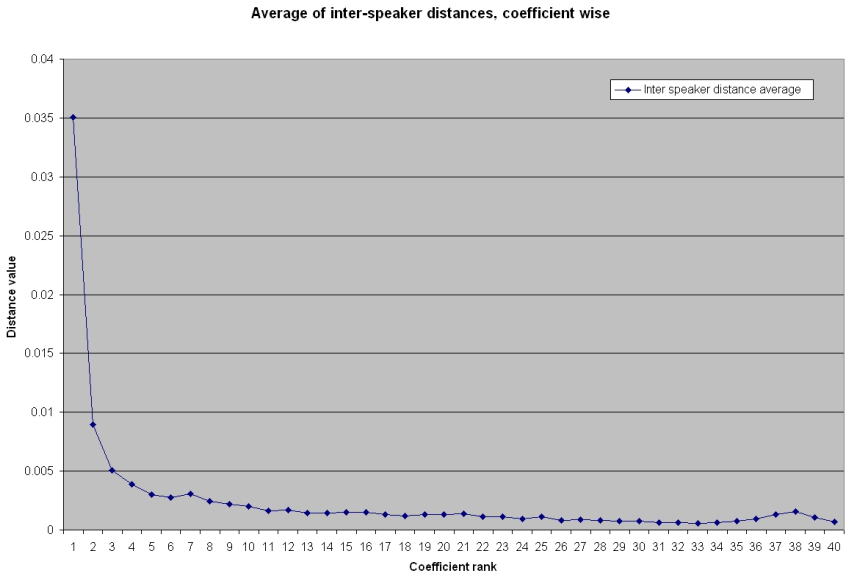

Figure 3.8: Inter speaker distances

It can be seen from the graphics that even if the values are different for the two data sets, the form of the two curves are almost the same.

In order to alleviate the problem of one small set of coefficients dictating the path, we introduced the possibility to use different weights for the coefficients. This way, instead of computing the simple Euclidean distance, each coefficient will have a different weight which will be multiplied with the difference between the two feature vectors.

In the tests we used a simple measure for grading the performance of the algorithm. Because we wanted to obtain a low distance between the true speaker and its templates, the measure was set to be the difference of distances between the true speaker and the minimum of the distances obtained by all the other users, divided by the lowest distance in the

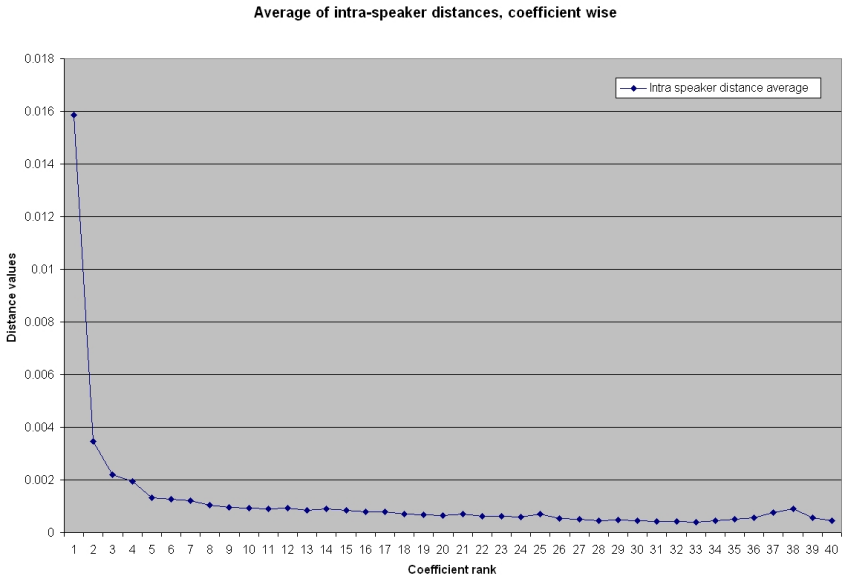

Figure 3.9: Intra speaker distances

set. This can be expressed as:

$$\Delta = \frac{|d_k - min_{i \neq k}(d_i)|}{min_j(d_j)} \tag{3.82}$$

If the true speaker gets the lowest distance in the set, then it is considered that the recognition was correct and a variable that holds a positive margin is increased with δ.

If the difference between the distance of the true speaker and the minimum of the other distances is positive, it means that the true speaker did not have the minimum distance and it is considered that the classifier gave a wrong result. In this case, another variable that holds the negative margin is increased with Δ.

The reason for using the two variables is that we want to know how far were the other speakers from the true speakers in the case of a positive result, or how wrong was the classification when the wrong speaker was identified. We wanted to increase the positive margin and decrease the negative margin.

If we combine the two margins by subtracting the negative margin from the positive margin we obtain a total margin.

One of the weights set we used was the inverse of the variance of each coefficient, computed on the templates of all digits and all users.

The results were good for clean speech data, the total margin obtained being more than 60% higher than the one obtained using equal weights.

The problem with this method is that with lower SNRs like 6dB or 0dB the results were worse than using equal weights.

For 6dB the percent of accurate recognition was lower than the one obtained with equal weights, but the total margin was still better. For 0 dB both of the indicators had worse values.

Another method we tried was based on the use of the intra-speaker variance that we men-

tioned earlier. Considerring that the database contains N_u users, each user has N_d digits and for each digit a user has $N_t(d,u)$ templates (which depends on the digit and the user), we used the following formula to compute an average of the distances obtained for each coefficient:

$$\sigma_p = \frac{1}{N_u} \cdot \sum_{u=1}^{N_u} \frac{1}{N_d} \cdot \sum_{d=1}^{N_d} \frac{1}{N_t(d,u)} \sum_{t=1}^{N_t(d,u)} dist_p, \qquad p \in [0,39] \tag{3.83}$$

Based on the averages of the coefficients, we used the following set of weights:

$$w_p = \frac{1}{\sigma_p}, \qquad p \in [0,39] \tag{3.84}$$

Although the method seemed promising, the results were worse than using equal weights.

3.5.2 Finding the best range of MFCC coefficients

Another thing we wanted to optimize was the range of coefficients to use in the one pass dynamic programming algorithm.

We wanted to find the set of coefficients that would give a set of distances with the true speaker having the lowest distance, and the rest of the speakers having the distances as high as possible. In order to characterize this, the margin from equation 3.82 was used. Another goal of the testing was to be able to find a range that would give good results across multiple databases, and not only one.

For this, an automated test program was created. The program applies the one pass dynamic programming algorithm on a file that contains the MFCC coefficients for a password and outputs a set of distances, corresponding to the registered users in the database. But instead of performing this on only one range of MFCC coefficients, the test is repeated for multiple ranges, which are specified in a configuration file.

We selected 400 ranges of coefficients, with lengths from 5 to 40, 40 being the total number of coefficients.

The audio database on which the testing was performed was TIDIGITS. This contained the templates from 18 speakers. The distribution on sex and age was the following:

- 6 female adult
- 2 female adolescent
- 1 female child
- 5 male adult
- 1 male adolescent
- 3 male child

Because we were looking for a set of coefficients that would give good results not only for a database, we created eight sets of users. Each set contained eight speakers. Each set had a different number of males and females. The eight sets contained the following distribution of speakers, on age and sex:

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8
Female adult	4	2	4	3	2	6	2	3
Female adolescent	-	1	-	-	-	-	-	1
Female child	-	1	-	-	-	-	1	-
Male adult	3	2	4	5	5	2	2	3
Male adolescent	1	1	-	-	1	-	-	1
Male child	-	1	-	-	-	-	3	-

Each user had two training templates for each digit, and for the digit '0' we had separate templates for 'zero' and 'oh'. For the results presented here, the templates were cut manually. For other tests, the templates were cut automatically, based only on the level of energy. The manual cutting of the templates was used because the sound files were presenting some abnormal levels in the zero-crossing rate. But this was not a restrictive condition because when a real database is built, the user can control the quality of the trimming of each template, and adjust the parameters if needed. The test files were cut automatically using the word isolator algorithm. Each user had about 50 test files and it contained passwords with lengths between 2 and 7 digits. Each file was matched against the users of the respective set, and it was tested for all the test ranges.

The database containing the templates for the digits had only clean speech, recorded in the studio. The test passwords had different signal to noise ratios, and were put in different folders, according to the SNR.

We used 6 SNRs for testing:

- clean speech
- 18 dB
- 12 dB
- 6 dB
- 0 dB
- -6 dB

Based on the sound files, on which denoising was performed, we created the MFCC files. These contained all the MFCC sets corresponding to the audio file. Because the test files were already trimmed, we needed to include a silence template as an external file. The silence template was different for each database corresponding to a SNR and it was taken from a denoised test file from the respective database. The length of the silence template was 200 ms.

We used an equal set of weights for the one pass dynamic programming algorithm.

The tests were run in parallel on two servers. The output of the tests consisted in one file for each set of speakers, and for each SNR. So there were 48 files with the computed distances for all the test ranges.

We used a function to interpret the results of the tests and to get quality indicators. The quality indicators used were:

- The number of tests with the correct user identified;
- The number of tests with the wrong user identified;
- A positive margin representing the distance from the true speaker to the closest match, when he was correctly identified.
- A negative margin representing the distance from the true speaker to the identified user, when he was not recognized as the true speaker;

- A global obtained by substracting the negative margin from the positive margin.

The margins were computed using equation 3.82.

The results were imported into Excel and then analyzed. We note a few observations:

- The performance of the algorithm degraded as the SNR was dropping;
- For all the SNRs, the ranges that included the first three coefficients gave bad results. This happened because we used equal weights and, as we have seen in section 3.5.1, this tends to ignore the differences between the higher order coefficients. The first three coefficients were supposed to help the recognition task, because the first three formants of the vowels are usually found in that range [7]. In [8] we can see that the first three formants help differentiate between a male speaker and a woman speaker.
- There was no test case in which the use of all the coefficients would give good results.
- For the high SNR, a small range of coefficients, usually starting from the fourth coefficient and not passing the 15^{th}, gave the highest scores. As the SNR was dropping, the starting coefficient remained the third, but the length of the optimal range was increasing. For the tests performed on the database with the SNR equal to -6dB, the last coefficient included in the optimal range was the 33^{rd}.
- The length of the password influenced the recognition rate, longer passwords giving better results. The results were less accurate if the length of the password was two or three digits.

The second goal was to see which range of coefficients is best suited if we cannot perform the extensive tests. So we grouped the data according to the SNR and tried to find an optimal range of coefficients. From all the possibilities of choosing a criterion to get the best range, we considered only two:

- MaxMin
- MaxAverage

In order to get a range according to the MaxMin criterion, for each SNR and for each coefficient we kept only the minimum value from all the speaker sets. If we can estimate the SNR of the environment where the recognition will take place, we can use these results and take the range of coefficients that has the maximum global margin. If we apply further the MaxMin criterion for all the SNRs, the result will be the range that maximizes the global distance for the lowest SNR. But if the SNR in the testing environment is higher than the lowest limit we imposed, the performance would only get better than what we would expect. This happens because, for each range, the global margin increases as the SNR increases. Still, optimal performance would not be reached. In conclusion, the MaxMin criterion would be useful if we are not interested in obtaining the best scores from the tests, but we are interested in obtaining a minimum level of performance.

The second criterion is not that pessimistic. So if a set has a lower global margin for a range of coefficients, it will possibly be ameliorated by the results obtained in the other databases. The criterion can also be applied only on the data obtained from tests performed at the same SNR.

It was interesting to see that, for the same SNR, the ranges indicated by MaxMin and MaxAverage were the same or the difference between the global margins indicated by the two methods was very small.

Applying the MaxAverage over the entire database tended to recommend the ranges closer to the optimal range of the tests performed on data with a high SNR. So if the SNR of the

medium is actually lower than what we were hoping, the performance of the algorithm will possibly be worse than we would expect.

As a comparison, the recommended range by the MaxMin criterion is $[3; 33]$ (starting from 0 and not including the last coefficient) if we don't include the data that has -6dB SNR and the range recommended by the MaxAverage is $[3; 15]$ if we don't include the tests that were performed on clean data.

One final observation would be that the results of the tests were very sensitive to the trimming of the templates. If the training database is not cut the proper way, the recognition accuracy drops with a few percents.

3.5.3 Finding the optimal projections

A weak point of the speech recognition algorithm was that it was not taking in consideration the actual structure of the database. There was no training performed on the data belonging to the other speakers. For example, the algorithm could not take into account the fact that two speakers have a similar voice, such that there was no possibility to find a way to emphasize the difference between the two of them.

This, combined with the fact that the scores were really close one to another, made us look for a new way of discriminating between the speakers.

The algorithm we will present was only implemented to work on passwords of one digit.

The idea is based on computing the distances coefficient wise. Instead of getting only one distance representing the distance to the origin of the feature space, we modified the one pass dynamic programming to output the accumulated distance along the alignment path for each coefficient. So instead of obtaining one distance, we were able to obtain a point in a 40-dimensional space. The distance obtained with the normal version of the algorithm can be thaught of like a projection of a vector on the line that passes trough the point $(1, 1, ..., 1) \in \Re^D$, where D is the dimension of the feature space. But that direction might not be the best direction to project the data. So we used an approach somehow similar to the Fisher Linear Discriminant (FLD). The FLD method, in the case of having two classes that need to be separated, tries to find a line in the space such that, when the data points would be projected on it, the projections will be clustered for the points that belong to the same class. At the same time, the classes would be ideally well separated.

In our method, we tried to find a line that would project the point belonging to the true speaker in the origin of the feature space, and the points belonging to the other users as far as possible, This way we would ideally obtain the distance equal with zero for the correct user, and higher distances for the other users.

One possible disadvantage of using the one pass dynamic programming algorithm with multiple templates is that the we don't know too many things about the 'geometry' of the space. We only know that we obtained by using a time warping algorithm a point situated at a certain distance from another reference *point*. The dimension of the reference point is equal to the number of frames of the template multiplied with the number of features (MFCC coefficients) that correspond to a frame. So if we have a template that has 60 frames, and for each frame we use 12 coefficients, we could think that we have a point in a 720 dimensional space. If we have multiple templates, each one of them having a different number of frames, then we would have to represent each template in a different space. So if we present a test template to the algorithm, we would eventually know which template was the closest to the input and what was the distance to that template. Even if all the templates had the same length, so we would be able to represent them in the same space, we would still not know where that point is situated.

So we needed a *fixed* point that we could use as a reference. The idea that was at the basis of obtaining that point is based on the algorithm presented in [9]. The algorithm was trying to find an average template based on multiple templates. It was shown that this method would increase the speech recognition accuracy by including in the same template the features of more templates, obtaining this way a better robustness.

In the following lines, the term 'templates' will refer to the training templates that belong to the same digit, and the same user.

In order to obtain the average template, first there was a template selected that had the length closest to the average distance of the templates for that digit. That was called the reference template. Then, the one pass dynamic algorithm was applied on the rest of the remaining templates, matching them against the reference template. A warping path was obtained for each of these templates. Based on these paths, a normalized template was computed, that had the same length as the reference template.

The original algorithm was only dealing with slopes equal to 0.5, 1 and 2. Based on these slopes, the following rules were used to obtain the normalized templates:

- Slope is 1: Nothing is needed to be changed
- Slope is 2: The frame of the speech signal is replicated (expand), i.e., w(i-1) gets the identical frame to w(i)
- Slope is 0.5: An average frame is calculated from the two consecutive frames (compress), i.e., w(i) is merged

We had to modify this algorithm because we didn't impose a limit on the lower value of the slope, but we limited the upper limit of the slope to 2. When the path goes for *t* time frames on the horizontal (slope is zero), we deal with a sub-unitary slope. The value of this slope is equal to the first non-zero slope divided by *t+1*. For example, in Figure 3.10 the slope is equal to $\frac{2}{5}$. The average was not computed only on the last two frames, but for all the

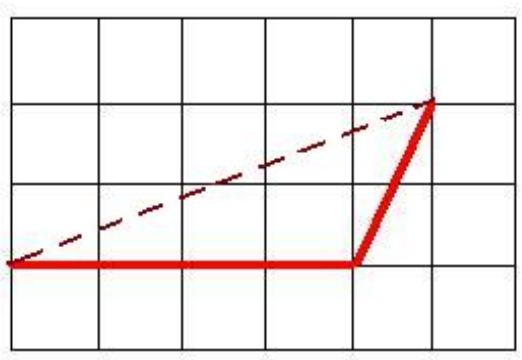

Figure 3.10: Example of sub-unitary slope

frames that made the sub-unitary slope. This average was multiplied as many times as the value of the last (non-zero) slope in the sequence. Multiplying the same value two times gives the algorithm the behavior of a zero order holder. An example can be seen in figure 3.11.

But we wanted to keep as many characteristics as possible from the original template. Repeating the same value twice made the normalized and the original time series to not

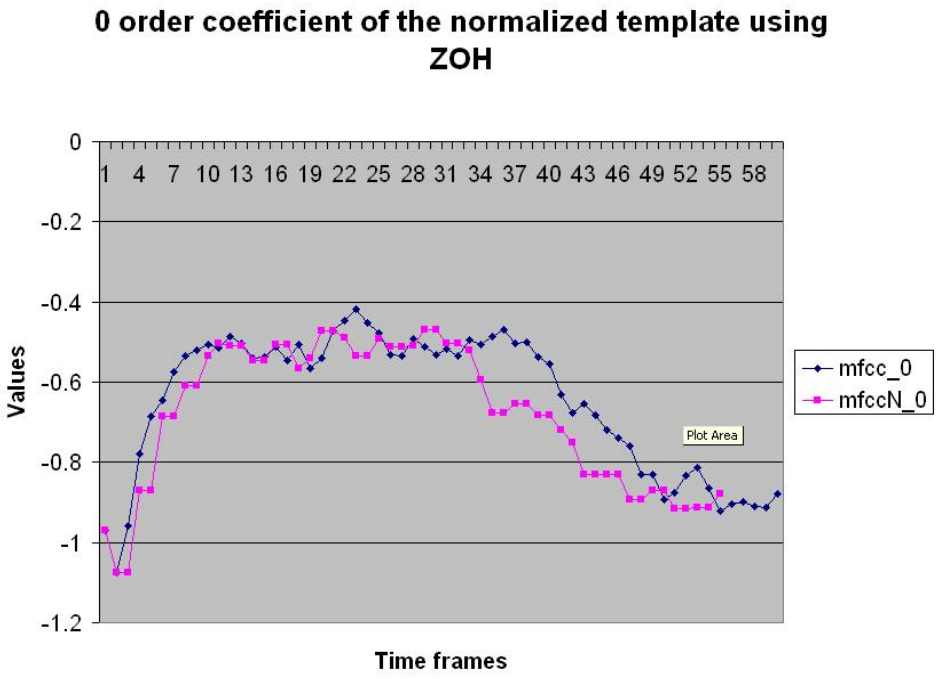

Figure 3.11: Example of time warped template - the variation in time of the first coefficient; the blue dots belong to the original template, while the pink dots to the normalized template.

look alike. So instead of using the ZOH behavior, we implemented a first order holder behavior: the value that was supposed to be copied the second time will be the average of the two adjacent points. By doing this, we hoped that the intra-speaker variability would not be affected too much. Using this algorithm, the same template from figure 3.11 will now look like the one in Figure 3.12. As it can be seen, the normalized template has a smoother aspect and tends to look more like the original template.

After all the normalized templates have been determined, an average template can be computed. Because all the normalized templates have the same length, each frame of the average template will represent the average of the corresponding frames from the other templates.

So now we have the fixed point in our feature space, against which we can do our matchings. But we wanted a measure in a lower dimension space, where we would have to move our fixed point.

The answer was again given by the dynamic time warping algorithm. We can reduce the dimension of the space by matching all our test inputs against the average template. By getting the distances accumulated along each coefficient, we would obtain a point in a relatively low dimension space, in our case at most 40-dimensional. We will call this the distance space.

In order to move the average template in this low dimension space, we applied the dynamic time warping algorithm on all the *original* templates, having the average template as a reference template. We obtained for each one of them a set of accumulated distances on each coefficient, which represent a point in the distance space. The points obtained from all the templates will form a cluster, which models the intra-speaker variability.

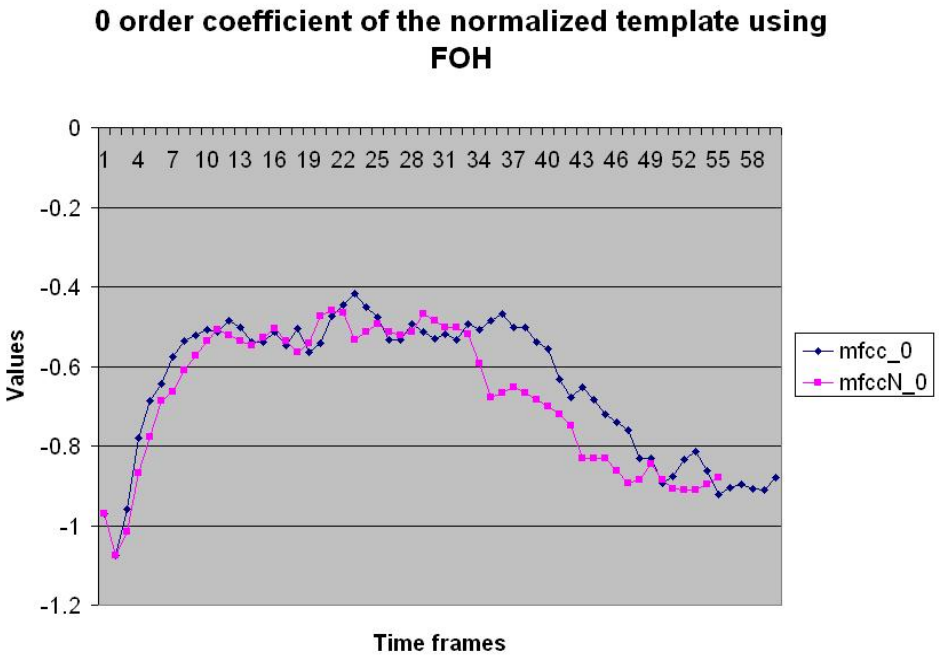

Figure 3.12: Example of time warped template - the variation in time of the first coefficient; the blue dots belong to the original template, while the pink dots to the normalized template.

If the intra-speaker variability of that user is low, then the cluster will tend to be very compact. If the variability is high, then the points will be far apart one from another.

Ideally, the points should be very close one to another.

If we take the average of all the distance-points, we will get the center of that cluster. So, instead of working with all the distances, we can only use the average point.

A schematic of the process of obtaining the average distance is represented in Figure 3.13, where T stands for 'template' and N for 'normalized template'.

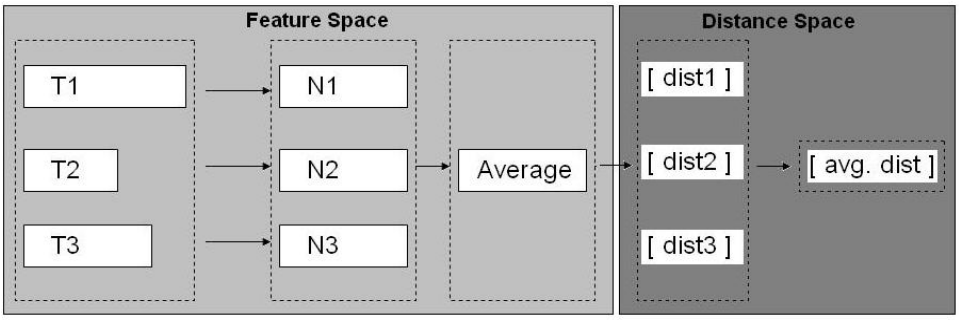

Figure 3.13: Obtaining the average distance for one digit of a user

We will have an average distance point associated for each digit of the user.

After obtaining the average distance point and the average templates for all the users, the first step of the algorithm is finished.

The second step of the algorithm is to perform the training of the database. In Figure 3.14 is presented a very simple case of finding an optimal projection.

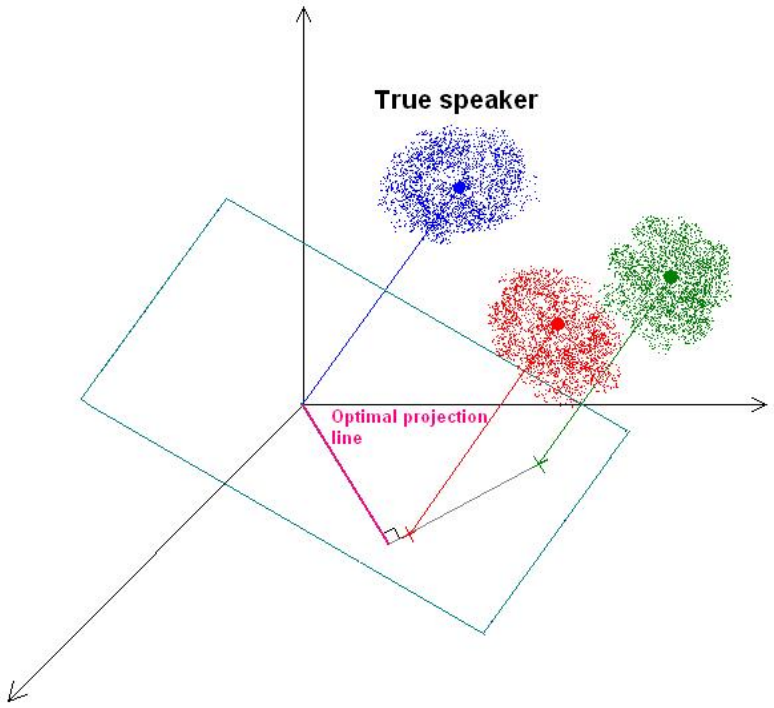

Figure 3.14: Finding the optimal projection line

We defined optimal the vector for which the smallest projection of all the other averages is the maximum possible. This can be stated as:

$$\underline{\hat{w}} = \max_{w} \left[min_j \left(< \underline{w}, \underline{d}_j > \right) \right] \tag{3.85}$$

In the simple case from figure 3.14, the optimal distance is the vector perpendicular on the difference of the projections of the two other points. It could have also been the vector parallel with the difference vector. The vector which is parallel with the distance is actually the perpendicular on the combination of points with one point negated.

In the general case, finding the optimal projection for a set of given points is more complicated. All the combinations of signs for the set of points have to be taken into consideration. For each combination, the optimal distance will be the vector perpendicular on the plane generated by the current configuration of points.

In order to find the optimal projection line for one digit of a speaker, the algorithm first projects all the points that belong to the other users in a subspace perpendicular on the vector represented by the average distance of the true speaker. Finding a vector in this subspace will ensure the fact that the true speaker will have its vector projected in the origin, and others will have a non-zero distance. Then the closest point to the origin of the projected subspace is checked for optimality. If all the lengths of the projections of the other points on this directions are greater than its norm than this direction is valid. At the

same time, if it is valid, this point gives the optimal direction. This is the best projection we can find. Any other direction we would try in the subspace will not give us a better margin.

If the closest point didn't give the optimal projection, we increase the number of points taken into consideration until we can find a valid direction. The number of points taken into consideration cannot be greater than the dimension of the space. If the number of points taken in considerations is k and the total number of points is N, then all the combinations of k points have to be taken into consideration. For each combination of points, all the combinations of signs have to be checked. If a valid direction is found, than the algorithm will stop. If more directions are valid, we will keep the one that gives the maximum minimal projection. If no valid directions are found, we increase the number of points and start over again.

Applying this algorithm for all the digits of all the users will give us a set of projection lines.

For the testing part, the user speaks the prompted digit. After preprocessing and feature extraction, the input data will be matched against the average templates of each user. For each user, the distances accumulated along the warping path are stored in a vector. This vector will be then projected on the direction obtained in the training phase for the respective digit. So each user will obtain a projection distance. The identified user will be the one with the distance closest to 0.

This method had some potential strong points:

- The method could be extended to using longer passwords, by extracting from the warping path the distances accumulated on each digit.
- The one pass dynamic programming algorithm was a lot faster because the input was not matched against all the templates for that digit, but only against the average template.
- By recording the value of the maximum minimum distance (which is like a safety margin) obtained during training, we could possibly estimate the performance of the algorithm.

During the tests we obtained some very good results. For example, the ratio between the correctly identified user and the next user was 10^4 or even more. Unfortunately, the method didn't give stable results, meaning that on some of the tests the true speaker was not correctly identified, and the one identified had the resulted distance very close to zero. This, and the time constraints, made us not use the method in the project.

Chapter 4

Project Documentation

4.1 General Description of Programs

The work included the creation of these programs:

1. Database Creator
2. AVPersonRecognition
3. Database Utilities
4. Audio Tester
5. Video Tester
6. Additional programs

Database Creator has the purpose of getting train data for the users, both audio and video. This part acquires the speech data and the video data, and transforms them into streams of data required by the main program.

AVPersonRecognition is the main program for identification, using both audio and video. In order to function properly, some functions from the Database Creator and Database Utilities need to be called.

Database Utilities contains functions that parse parts or the entire database, giving as output different parameters or data for other algorithms. For example, this program is able to recompute the MFCC files for a user or for the entire database, or compute particular sets of weights based on the audio data.

Audio tester had the main purpose of testing the audio recognition part of the program. The tests are done on a database containing digits from the TIDIGITS database. The database with the format needed is created using the Database Utilities.

The Video tester has the purpose of finding the video test files in the user test folders from the database and applies the video processing algorithm in order to generate the scores for a video training algorithm.

The additional programs are the ones written in Matlab or in Perl. The Matlab programs are mainly created to compute optimization parameters or for visualization. There is also a Perl script that creates a list of the audio files in a database that need to be tested.

4.2 Database Creator

The database creator is a set of functions that will create the structure of the database, registers new users and gathers data for training. It has the necessary functions for the audio and video data recording.

A screen capture of the dialog can be seen in Figure 4.1

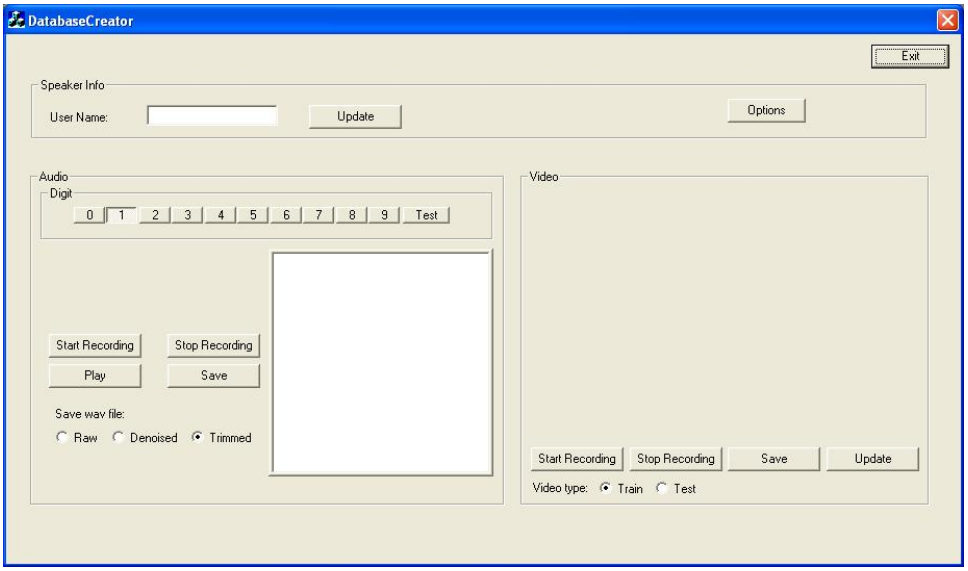

Figure 4.1: Database creator dialog

In order to be able to use any of the functions, the user name has to be entered in the corresponding text box and the Update button has to be pressed.

In the function called by the update button the program looks in the database folders to see if there is already a folder with the same name as the text inserted. If the program cannot find the user folder, it will ask the user if he wants to create a new user. If the answer is No, nothing will happen. If the answer is Yes, a new folder with the right structure will be created in the database. This dialog will appear for each user inserted if the database path is not correctly stored in the AVPR_Parameters.ini file.

The user folder structure is identified by the following:

- the root folder is always the user name;
- there is a folder created for each digit, from 0 to 9. This is where the audio and the corresponding MFCC training templates are stored;
- there is a folder called 'Pictures' where the video training faces are stored;
- there is a 'Test' folder where both the audio and video test data is stored. These files can be used as training data for fusion parameters;
- A folder called 'Dump' is created to store the temporary files for the audio processing (denoised and trimmed files).

If the user already exists in the database, the Tree Control is updated with the MFCC files found in the user folder. If there are folders without templates, the program will show a warning.

After updating the user name, both video and audio data can be recorded.

4.2.1 Audio Recording

The audio recording part takes care of recording train data for the respective user. The function called to record audio data is AudioDataCapture.

In order for the program to work correctly, the user has to select the right option from the Radio buttons (e.g. If the user wants to record a template for the digit 2, the radio button ”2” has to be selected).

If one of the radio buttons containing a digit or ”Test” is selected, the tree control will be updated with the names of the MFCC templates found in that digit folder.

To actually record a template, the user has to press the ”Start Recording” button. This will start the recording of the speech in the WaveNew.wav file. It is recommended that the user starts to speak 300 ms after the Start button was pushed. This is necessary for the trimming algorithm. If the break has a smaller length, the word isolator parameter related to silence length should have a smaller value. The silence offset should not be changed, because the noise is still introduced by the denoising algorithm in the first 100 milliseconds of the audio data.

Pressing the ”Stop Recording” button after the digit was spoken will stop the recording and start processing the data. The performed processing is the denoising and trimming of the wave file.

The user can listen to the files obtained after processing of the signal is done. If an external editor that accepts command line parameters (like Cool Edit Pro 2) was provided to the program, the user can see and edit the results. The default file that is edited is the trimmed file, but this can be changed by pressing one of the three radio buttons that are situated under the ”Start Recording” and ”Stop Recording” (Raw, Denoised and Trimmed). If unsatisfactory results are obtained with the trimming process, the user can change the parameters in the Options dialog and then press the ”Stop Recording” button again.

After obtaining a satisfactory result, the final step is to save the file in the selected digit folder. This process will perform the feature extraction on the trimmed file. The MFCC file saved will be put in the right folder, along with the selected file. It is recommended that the trimmed file is saved in the digit folder. The name of the template saved has the following template: ”UserName_Digit_OrderNumber.mfcc”. The order number is a two or three digit string that starts from zero and differentiates the different templates saved. The order number is generated with a function (getNextAvailableIndex) that looks in the destination folder for the files with a specific extension and returns the first number that is available. E.g. : If a folder contains the two templates: User_1_00.mfcc and User_0_02.mfcc, the next index returned is 1; if the templates are User_0_00.mfcc and User_0_01.mfcc, the returned index will be 2.

If the ”Test” radio button is selected, instead of a digit, an entire password will be saved in the digits folder. The password length can be set in the text box that appears after pressing the button.

After pressing the ”Start Recording” button, a random password will be generated and displayed on the screen. The user will have to read this password.

The rest of the process is identical to the recording of a digit template. The filename will contain in the Digit section the spoken password.

4.2.2 Video Recording

The capture of the video data from the camera is implemented in the VideoDataCapture.cpp file and the control structure sent to the function is defined in the VideoDataCapture.h file.

Trough the video control structure the following functions can be performed:

- record 24 bit color bitmaps or 8 bit grayscale pgm files;
- record a compressed video if the pictures are saved as 24 bit bitmaps;
- set a limit on the number of recorded pictures;
- record only the pictures that have a face detected, if the saving format is pgm file.

The function accepts a prefix for the saved pictures, to which an index will be added, a path to where the pictures will be saved and a name and a path for the movie in case the option of recording a movie is specified.

The function will put in the CString parameter ”recordedPictures” from the control structure the files saved during the recording process. The files are separated by a new line character.

There are two options for the video recording:

- training data
- test data

The training data is required to get a number of instances of the user face in order to train the face recognition algorithm.

The test data is recorded to perform a number of tests on the video recognition module. These tests will generate a set of scores that will be used to train the parameters of the probability function.

The switching between the two modes is made by clicking on the appropriate radio button situated under the control buttons. If there are files that were not saved after recording, switching between these two modes will erase them.

All the pictures are recorded in a 320x240 format.

Recording Training Data

The user starts recording data by pushing the ”Start Recording” button.

The pictures will be recorded at a 250 ms interval and will be displayed in the video group box.

The pictures will be saved in the ”Pictures” sub-folder as 24 bit bitmaps. This is necessary because a compressed video file has to be saved.

The number of acquired images is not limited. The recording will stop when the user will push the ”Stop Recording” button.

A movie will be created based on the recorded pictures. The user has to select the compression format Intel Indeo Video 4.5.

After saving the movie, the control will be given to an external program (play.exe) that is able to extract 64 by 64 bitmaps containing the user’s face. Because the program doesn’t accept command line parameters, the user will have to open manually the movie saved in the same folder.

A face can be extracted out of every frame by clicking on the points corresponding to the centers of the left eye, the right eye and the mouth. The needed images will be saved in the folder C:\lobby_fdr\face2. This folder cannot be changed because the face extracting program was given ”as it is”.

After extracting a number of faces, the face extractor program has to be closed and the control will go back to the Database creator program.

If the user is satisfied with the images, he can push the ”Save” button. This will copy the acquired faces from the ”C:\lobby_fdr\face2” into the ”Pictures” folder of the selected user.

In order for the face recognition algorithm to take into account the newly recorded pictures, the video database training file has to be updated. This is done by pushing the ”Update” button. The function behind this button will start looking for training images in all the user folders from the database. The names of the found images will be put in a text file having as a header the database path, after which each line will contain the path of the training file relative to the database.

This newly created file will be passed to another external program, found in the ”fr_train” subfolder. This is also the place where the text file is saved. The program that will create the training files for the face recognition is ”fp.bat”. The parameters for this batch file are:

- the text file that has the images of the users;
- the name of a temporary file;
- the name of a trained data file;
- the name of a gallery file.

The last two files contain the necessary data to perform video recognition for the speakers from the database. After computing these two files, they are copied in the root of the database folder. If the program finds previous versions of these files, the user is asked whether to update the recognition files or not. If the answer is YES, the files will be overwritten.

Recording Test Data

The process of recording the files also begins by pushing the ”Start Recording” button.

The pictures are saved as 8 bit pgm files in the ”Pictures” subfolder at intervals of 250 ms or more. The reason for not saving at precise intervals is that the setting for detecting a face in the files to be saved is enabled. So, if 250 ms have passed, but there wasn’t a face detected a face in the image, the data will be discarded and the next picture will be checked. The process goes on until a picture has finally been saved, after which the timer will be reset.

The number of pictures acquired by the function is limited to 30, but the number can be changed, to more or less.

The pictures are recorded as 8 bit pgm files because the face detector and the face recognizer need 8 bit grayscale images as an input.

After 30 pictures will be recorded, the images will not update on the screen anymore and the ”Stop Recording” button has to be pushed.

If the user wants to save the files, the ”Save” button has to be pressed. This will copy the files in ”Test” folder. The files will be renamed in ”UserName_0_Index.pgm”, where ”Index” is the next available number in the folder.

The files will be later used by the Video Tester project.

4.2.3 Required Files, Folders and Programs

Files

The only required configuration file (.ini) is AVPR_Parameters.ini.

The program also needs the file ”FD_CompFace4Scales.UB.N” for the face detection algorithm (if face detection is used in the recording of test files).

Folders

The program needs the following subfolders:

- ”Pictures” - to store the temporary pictures
- ”fr_train” - it also contains the video database training programs

A folder called ”lobby_fdr” will be automatically created by the face extractor program (play.exe).

Programs

The following programs are needed:

- play.exe - extracts the faces from a compressed movie file;
- all the executables from the ”fr_train” folder;
- an external audio editor that can open a wave file received as a command line parameter (optional, but recommended).

4.3 AVPersonRecognition

This is the main project, which performs the audio and video person recognition.

It relies on the functions performed by Database Creator and it needs in certain situations the functions offered by the Database Utilities.

The Graphical User Interface is implemented as a dialog, and a screenshot is presented in Figure 4.2.

The parameters can be changed by pushing the ”Change Param.” button. This pops up a configuration dialog like the one in Figure 4.3.

A description of the parameters can be found in section A.1.

The interface of the program is very simple, the user having to press only two buttons.

The audio and video data is recorded simultaneously. The functions used to record the data are the same used in Database Creator project, namely AudioDataCapture and VideoDataCapture.

To begin capturing data, the user has to push the ”Start” button. This starts the two threads for audio and video data capture.

4.3.1 Audio Recognition

When the user pushes the Start button, the program generates a random password with digits from 0 to 9, having the length specified in the text box next to the ”Number of digits” label.

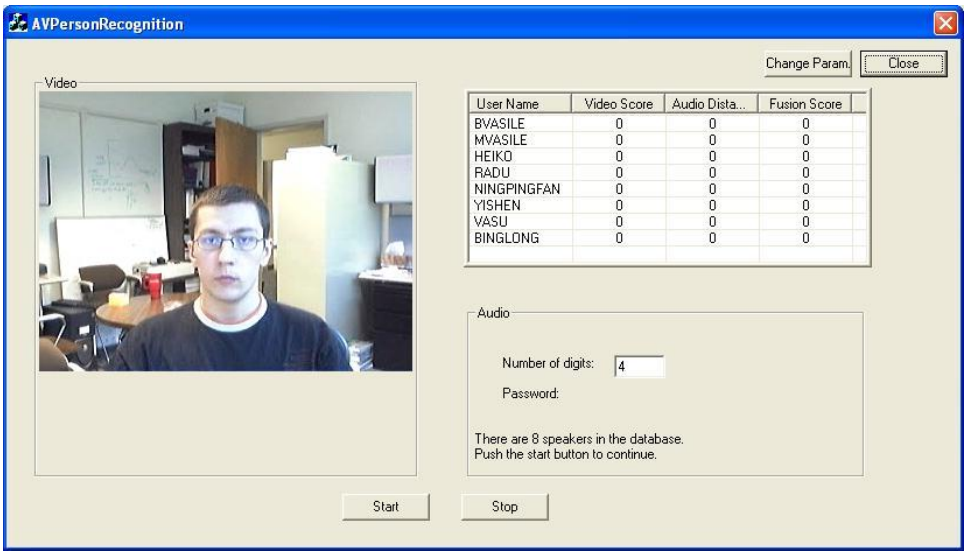

Figure 4.2: Audio Video Person Recognition dialog

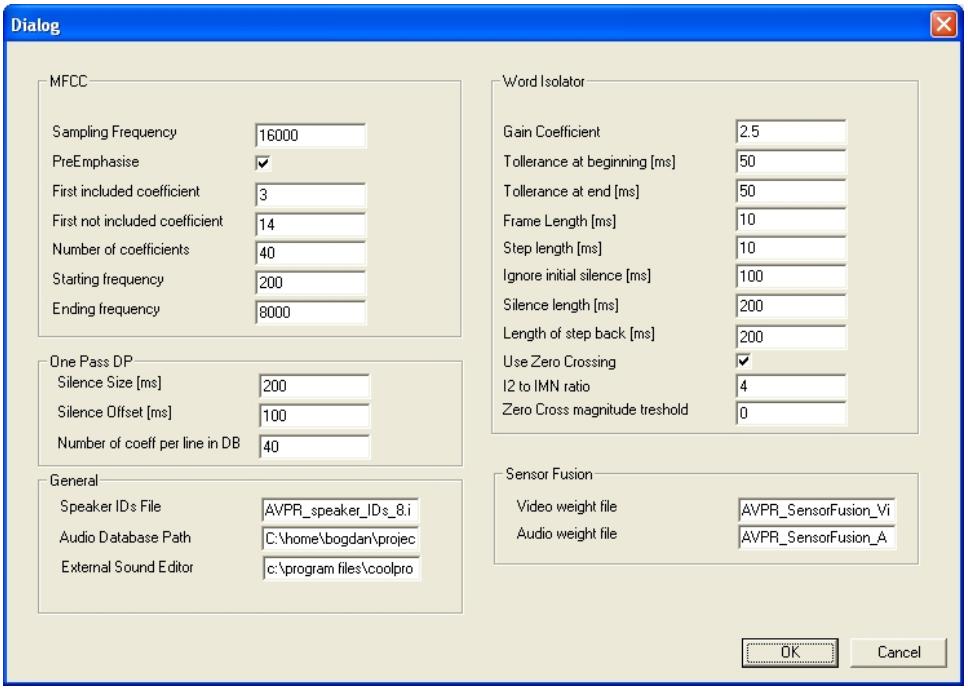

Figure 4.3: Configuration dialog

The user has to read the password. After the password was spoken, the ”Stop” button has to be pressed for the audio recording to stop. The data is saved in the ”WaveRaw.wav” file in the same folder as the executable.

The program will also wait for the video data capture to stop.

After both the recordings stop, the audio processing begins.

First, the raw wave file is processed. This consists in denoising and trimming. The two resulted files are ”WaveDenoised.wav” and ”WaveTrimmed.wav”. during this process the data for the silence templates is also extracted, from the beginning of the WaveDenoised.wav file.

Based on the data from the WaveTrimmed.wav file, the program then extracts a set of features, namely a set of Mel Frequency Cepstral Coefficients for each time window in the input. The MFCCs are computed by filtering the signal with 40 log equally spaced filters, and the applying the Discrete Cosine Transform. So the test input will be made of a series of concatenated 40-dimension vectors, its length being equal to the number of windows (frames) that enter in the test signal. The window size used had a length of 256 samples and two consecutive windows were separated by 160 samples (the step length).

The next step is to get the training data for each user registered in the ”AVPR_speaker_IDs.ini” or equivalent. This is done using the function ”GetTrainMFCCs” defined in the AudioScoreCompute.cpp file. This will get a ”long” vector of features that contains all the templates found in the digit folder of that user for all the digits of the password. In between the sets of different digits, in the beginning and in the end of the vector, a silence template is inserted. This will help match the silence found in the test file in the beginning, in the end or between the spoken digits. In order to keep track of the coefficients for the different digits, three additional structures are used.

The first is called ”Nt” and it holds the number of templates found for each user. The first template is always the silence template, and the number of templates for the rest of the digits is increased by 1, representing the silence template added at the end of the templates for the respective digit. This is an integer vector of length ”number of digits” of the password plus one.

The next auxiliary structure is called ”Nf”, coming from Number of frames, and it is a pointer to vectors of different lengths. The number of vectors is also equal to the ”number of digits” of the password plus one. Each pointed vector holds the number of frames of each template inserted in the train vector, including the silence templates.

The last auxiliary structure is called ”Linear” and it has the same characteristics of ”Nf”. The difference is that it holds the index of the first coefficient that belongs to the corresponding digit in the MFCCs vector. It is actually derived from ”Nf” to allow a faster access to the MFCCs.

The advantage of using these structures is that the number of templates doesn’t have to be the same for different digits, or for digits that belong to different users.

After getting the templates for all the digits of the password, the one-pass dynamic programming algorithm is applied on the test and training data. The accumulated distance on the warping path and divided be the input length represents the score obtained by the respective user.

This process is repeated for every registered user, until all the users have an associated score for the given test input. So the results will practically represent the distance of the tested speaker to the registered users.

The set of distances will be passed further to the calling function, where they will be trans-

formed into probabilities.

4.3.2 Video Recognition

The video recording also starts when the ”Start” button is pushed.

Because the audio and video recording take place in the same time, the tested user pictures will be taken while he or she speaks the password.

The pictures are saved as 8 bit pgm files in the ”Pictures” subfolder.

In order to get a normalized score between the different users at different moments of time, there were two measures taken. The first is to only save pictures that had a face recognized. So the video control structure parameter ”detectFaces” is set to true. This measure had to be taken because only the pictures that have a face detected go into the face recognizer, and, based on the confidence threshold for face detection, not all the pictures had a face detected.

The second measure was to limit the number of saved pictures to a certain amount, a typical setting being 10 or 20.

The two settings reduced greatly the variability of scores obtained for the same user in different tests.

Because the limit of pictures that have to be recorded can be relatively high and because some frames will be dropped, the time interval needed to record the pictures can be a lot longer than the time required to record the audio data. So, after the user pushes the ”Stop” button, the program will have to wait for the video recording to stop before the audio and video processing continue.

After the video recording thread exits, the list of recorded pictures is passed as a parameter to the video processing module. The processing function reads the face detection and the face recognition training files and initializes the detection and recognition algorithms.

After all the initializations are finished, the algorithm will take one picture at a time from the list and will try to detect a face. If a face is not detected, the algorithm will read the next picture.

Because the face detection algorithm is not perfect, the detected face image will be ”prepared” by also sending to the face recognition module the shifted versions of the images, with one pixel in each edge and corner direction, totalizing 9 pictures.

The face recognition algorithm only works on streams of data. The number of pictures in the stream is set to 10 or 20, depending on the number of pictures captured. The pictures will be accumulated in an internal buffer and a score will be outputted only after the stream is full. After receiving a score the stream is reset and the process starts over again.

The scores outputted by the face recognition module are accumulated in an array. Before the video recognition algorithm ends, the scores are divided by the number of detected faces, to obtain a normalized value.

The scores are passed in the end to the calling function to be transformed into probabilities.

4.3.3 Probability computation

The posterior probabilities for the separate audio and video are computed using the following formulas:

$$P(i|Audio) = \frac{\lambda_i \cdot exp\left[-\lambda_i \cdot (dist_i - \min_j(dist_j))\right]}{\sum_{k=1}^{N} \lambda_k \cdot exp\left[-\lambda_k \cdot (dist_k - \min_j(dist_j))\right]}$$

$$P(i|Video) = \frac{\gamma_i \cdot exp\left[-\gamma_i \cdot (\max_j(score_j) - score_i)\right]}{\sum_{k=1}^{N} \gamma_k \cdot exp\left[-\gamma_k \cdot (\max_j(score_j) - score_k)\right]}$$

The lambda and gamma parameters are read from two files that have to be in the same folder as the executable. Their names are specified in the AVPR_Parameters.ini file, the parameters names being AVPR_SENSORFUSION_AUDIOWEIGHT_FILE and AVPR_SENSORFUSION_VIDEOWEIGHT_FILE. The files are read when the application initializes. For more details on these files please review section A.3.

The posterior probability of user *i* based on both audio and video data is computed using the formula:

$$P(i|A,V) = \frac{\lambda_i\gamma_i \cdot exp[-\lambda_i \cdot (dist_i - \min_j(dist_j)) - \gamma_i \cdot (\max_j(score_j) - score_i)]}{\sum_{k}^{nUsers} \lambda_k\gamma_k \cdot exp[-\lambda_k \cdot (dist_k - \min_j(dist_j)) - \gamma_k \cdot (\max_j(score_j) - score_k)]}$$

The recognized user is considered to be the one with the maximum posterior probability. The video scores, the audio distances and the posterior probabilities are displayed in the Tree Control from the upper right side of the window.

4.3.4 Required files, folders and programs

Files

The configuration files needed are the following:

- AVPR_Parameters.ini
- AVPR_speaker_IDs.ini or equivalent
- AVPR_SensorFusion_AudioWeights.ini or equivalent
- AVPR_SensorFusion_VideoWeights.ini or equivalent
- AVPR_Weights.ini

For more details on these configuration files please consult section A.

The video recognition module needs the following files:

- FD_CompFace4Scales.UB.N in the program folder, for the face detection
- FR_Gallery.fr and FR_Trained.fr files in the root of the database folder. For more information on these files consult the Database Creator section.

Folders

The program needs the subfolder ”Pictures” to store the recorded pictures.

Programs

This project needs no additional programs.

4.4 Database Utilities

This is an auxiliary project that offers additional functions for the audio recognition.

It is implemented as a dialog and a screenshot can be seen in Figure 4.4.

The parameters can be changed by pushing the ”Options” button, which opens the configuration dialog.

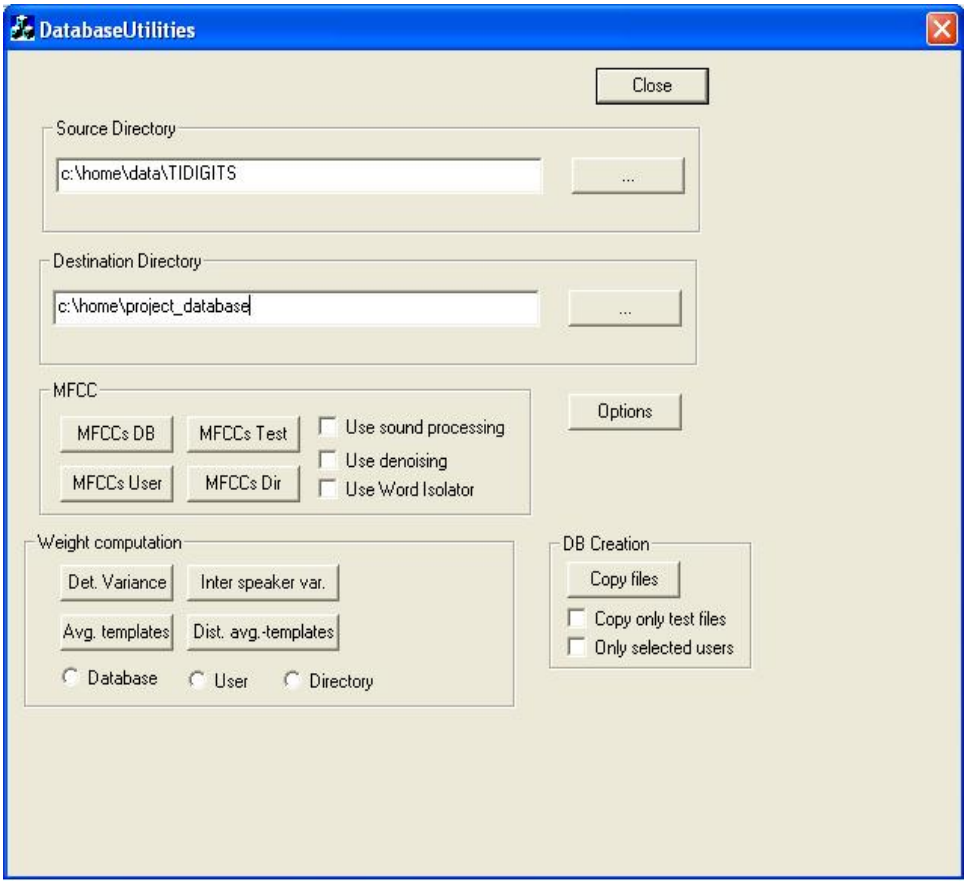

Figure 4.4: Database Utilities dialog

There are three main functions performed by the Database Utilities program:

- Audio Database creation
- MFCC computation
- Weight computation

4.4.1 Audio Database Creation

This function was used to create a database format required for the AVPersonRecognition project based on the TIDIGITS database.

The TIDIGITS database was given as a folder that contained the entire user's recorded audio data, namely the digit templates and test files.

The digit templates had the naming format UserName_DigitX_comment.wav, where X is a character denoting the template index (e.g. AC_1A_car.wav and AC_1B_car.wav). The naming convention used in the project is "UserName_Digit_Index.wav". For more information on this, check the section "Database Creator".

The digit "0" had two different templates, one for "Zero" and one for "Oh".

The test files had a similar notation, but the digit portion contained all the digits from the password.

We needed to put all the files in a new database, with the folder structure used in the other programs. The user has to select the source and the destination folders, using the associated browse buttons. If the destination folder doesn't exist, the user has to write a name for the new database.

After the user pushes the "Copy Files" button, the program will check to see whether the source and destination folders exist or not. If the source destination doesn't exist, the function will exit. If the destination folder doesn't exist, the user will be asked if he wants to create a folder or not. If the answer is YES, the program will create the folder and start the process of creating the database.

The program looks in the source folder and searches for "wav" files. For each file it splits the filename to see the user and the digits. If the user already exists, it will copy the file in the right folder. If the user doesn't exist, it will create the structure and then copy the file in the right folder. For more information related to the user folder structure, check the "Database Creator" section.

The process will end after copying all the files from the source folder.

The user has the possibility to only copy the templates of specific users, namely the ones that are registered in the "AVPR_speaker_IDs.ini" or equivalent file. This is done by checking the checkbox "Copy only selected users". If the program finds templates that don't belong to the selected user, it will ignore them.

The other option is to copy only the test files, not the templates. This function was very useful during the testing phase of the algorithm, when we wanted to test the performance of the algorithm for different levels of the SNRs. In that case, the digits templates were the ones that had a higher SNR, and the tests were performed on the lower SNRs, not needing to create a train database with the lower SNRs.

The result is a database that contains only the wave files of the training digits or test passwords. In order to use it, the user will have to compute the MFCC files corresponding to the audio data.

4.4.2 MFCC Computation

There are four functions that have the purpose of computing or recomputing the MFCC files corresponding to the audio files from a location. They are very useful when the user needs to change the MFCC related parameters, because all the MFCC files have to be recompiled to reflect the new changes.

The functions are the following:

- computing the MFCCs for the files in a folder
- computing the MFCCs for the training files found in a user folder
- computing the MFCCs for the training files found in a database
- computing the MFCCs for the test files found in a database

The last three functions rely on using the first function recursively, until all the folders were updated.

The user can specify the processing amount of the audio data by checking the ”Use denoising” and/or ”Use Word Isolator” options. If no option is selected, the MFCC files will be computed for the files found in the folders, without any further processing. This can be useful when the files from the database are already denoised and trimmed.

Because the trimming is the last processing performed on a file, if you check this option, the MFCCs will be computed based on the trimmed file. If this option is not checked, the MFCCs will be computed based on the denoised audio data, if the option is checked.

When computing the MFCCs for a user, for the database or for the test files in the database, and signal pre-processing is required, the temporary files will be saved in the ”Dump” folder of each user, by adding the prefixes ”trimmed_” and ”denoised_” in front of the wave file names.

When the button for computing the MFCCs in a folder is selected, and sound processing is required, the temporary data will be saved in the same folder. In order to prevent the recursive processing of the wave files, a list of the existing files is created and only the initial files will be processed.

4.4.3 Weight Computation

This set of functions has the purpose of determining a set of weights based on the training digit templates from the database.

There are three categories of methods for computing a set of weights:

- based on the variance of the coefficient
- based on the intra speaker and inter speaker variability
- based on the distance to the average templates

The method based on the coefficient variance will output a weights initialization file. For more information on this file, please review section A.4.

The other two methods will output some text files based on which a set of weights can be computed. The database is specified in the ”Destination Directory” text box. If the entered path is not valid, the program will not start the computation.

Computing weights based on the coefficients variance

Pushing the ”Det variance” button will begin the computation of a set of weights based on the coefficients variance.

The method looks in all the template folders of all the users and in the first step computes the average value of each coefficient.

Then the entire database is parsed again to effectively compute the variance.

The output is the inverse of the variance of each coefficient. This is dumped in the ”AVPR_Weigths_var.ini”. The name of the file has to be changed in ”AVPR_Weights.ini” if you want to use these values.

This function doesn’t need the existence of a weights file.

Computing weights based on intra-speaker and inter-speaker variability

The characteristics of the human vocal tract are variable in time. The variation appears because of many factors, including stress, fatigue, emotions, health state, etc. So if a user records two templates for the same digit at different moments in time, indubitably a variation of speech will appear. We call this the intra-speaker variance. The smaller this is, the better the recognition task performs.

Given two different speakers, that record a template for the same digit, we want to see how different the two sounds are. If they are very different, the recognition task will be quite easy (consider the same digit spoken by a man and a woman). If the sounds are very similar, the speaker recognition task can be very difficult and with a high error rate (consider the same digit spoken by two children, of the same sex and age). We call this difference the inter-speaker variability. The higher it is between the users in a database, the better the recognition task will perform.

Knowing these, we wanted to compute the intra-speaker and inter-speaker variability in the database. The function that computes the two measures of variability is called by pushing the ”Inter Speaker Var.” button.

The algorithm is applied on an entire database.

The first step of the algorithm is to gather the names of all the MFCC files in the database, from all the users, which belong to the same digit training templates.

Then the program applies the one-pass dynamic programming algorithm on all the different combinations of two templates from the list. For a combination, the algorithm is applied two times, one time with template A as reference, and then with template B as reference. The results include the accumulated distance (the normal output of the one-pass dynamic programming algorithm), but also the accumulated distance for each component. The value of the accumulated distance is the sum of the squared difference between the two sets of coefficients, taken as time series. Out of the two sets of distances, the one corresponding to the smallest accumulated Euclidean distance will be kept.

If the compared templates belong to the same digit of the same user, the result belongs to the intra-speaker variability. If the templates of the same digit belong to different users, then the result belongs to the inter-speaker variability.

The results will be put in two text files, one for the inter-speaker and one for the intra-speaker variability.

Each line of the files contains the names of the two compared outputs and the accumulated distances for each coefficient.

The results can be used and processed in a different program, like Excel, to extract a set of weights.

The results depend on the initial weights used in the one pass dynamic programming algorithm.

Computing weights based on the distance to the average templates

Obtaining this set of weights implies three steps:

- Computing the average templates for all the digits belonging to the users from the database.
- Computing the average distances between the previously computed average templates and the original templates.
- Using the output of the previous steps as an input to a Matlab script to obtain the set of weights.

The average templates are computed by pushing the 'Avg. Templates' button. The user has to push one of the radio buttons, to select the computation for a database, for a user or for the templates in a folder.

The function called will recursively compute the average templates for all the needed folders. For the templates of one folder (or one digit), the algorithm first looks at the number of frames of each template. The average length of these templates is then computed. The length of the average template will be the length of the template that has the number of frames closest to the average length. The template with the selected length will be the reference template.

Then the algorithm will get a set of normalized templates. These are obtained by applying the one pass dynamic programming on the original templates, having as training template the reference template. A warping path is obtained for each of these templates. Based on it, a normalized template will be computed for each of these templates, that will have the same length as the reference template. The normalized templates will have the same structure as a MFCC file, but the file will be saved with a "mfccn" extension.

The average template for the respective digit will be computed by taking the average of the corresponding frames from all the normalized templates. The average file will have the same structure as a MFCC file, but will have the extension "mfcca". The naming convention for the average files is "UserName_Digit_average.mfcca"

In order to find the average distances to the average templates from the second step, all the digits have to have an average template. Also, all the users must have at least two sets of templates for each digit.

The second step of getting the set of weights consists in obtaining the distances between the MFCC sets of the original templates and the average template for each digit. The function that is performing this action is called by pushing the "Dist. avg-templates" button. This function works on the entire database and it searches every user folder in the database.

The function is able to compute the distances for multiple ranges of coefficients. These ranges are specified in the "MFCCtest.ini" or equivalent file. For more informations on this file, please review section A.5. For each of the test cases, the algorithm will be applied on the database, and the results corresponding to that range will be appended to the result files.

For all the MFCC files found in a user folder, the function will apply the one pass dynamic programming algorithm having as reference the average template of that digit. For each

template, a vector containing the distances accumulated on each coefficient will be stored. After computing the distances for all the templates in the folder, the average is taken. The vector containing the average will be appended to a text file corresponding to that digit, along with the name of the user.

In the end, the program will output as many text files as the symbols found, that have the naming convention "Distances_X.txt". The "X" in the name stands for the digit. The different test cases will have the range of MFCCs used added before all the average distances are outputted.

The third step of the algorithm will find the optimal projections for each digit of each user. The algorithm is described in sections 3.5.3 and 4.7.5.

The tests were performed in the AVPersonRecognition project by setting the password length to one digit modifying the Audio Score Computation (AudioScoreComputeOnWav) function. The algorithm was applying the one pass dynamic programming algorithm on the input test data having as reference the average template of the corresponding digit of each registered user. After computing the accumulated distances for each coefficient, it was reading the projection corresponding to the tested digit of the verified user. The score was obtained by computing the dot product of the projection vector and the distances vector. The identified user was the one having the closest score to 0.

4.4.4 Required files, folders and programs

Files

The configuration files needed are the following:

- AVPR_Parameters.ini
- AVPR_speaker_IDs.ini or equivalent for the audio database creation
- AVPR_Weights.ini - for computing the distances to average templates and intra-speaker and inter-speaker variability
- MFCCTest.ini for the distance to average templates

For more details on these configuration files please consult Appendix A.

Folders

This project doesn't need any folders to be created.

Programs

This project needs no additional programs.

4.5 Audio Tester

The user interface of this program is the command line interface.

There are three functions that are performed by this program:

- testing MFCC files
- interpreting test results
- extracting results from the test output file

The three functions can be used by sending different arguments to the command line.

4.5.1 Testing MFCC files

Initially, the audio tester program was created to test different recognition algorithms on the audio databases created from TIDIGITS database.

The role of this function is to test passwords belonging to different users on different ranges of coefficients, e.g. to test a password using the coefficients from 0 to 13, then from 1 to 13 and see which range gives better classification results.

The test cases are stored in the MFCCTest.ini file or equivalent. For the testing, we used 400 contiguous intervals, with the length varying from 5 to 40 coefficients. Each file was tested on each of these intervals.

This function tests one file at a time for all the intervals specified in the MFCCTest.ini file. In order to be able to test it, the following arguments have to be sent to the command line:

- argv[1] - ”-t” in order to perform the testing
- argv[2] - the path to the tested file
- argv[3] - the test file name
- argv[4] - the path to the silence template
- argv[5] - the name of the silence template
- argv[6] - the name of the file that contains the test cases
- argv[7] - the digits of the password
- argv[8] - the length of the password (number of digits)
- argv [9] - the name of the output file, to which the results will be appended
- argv[10] - the name of the true speaker

The first argument has to be ”-t” to be able to use this function.

The second and the third arguments specify the file to be tested. Because the tests can take a long time, we wanted to optimize the process and we tested only the MFCC files. So the indicated file has to be an mfcc file.

Due to the fact that the file is already processed and maybe trimmed, we cannot extract the silence template from it. This is why an external silence template has to be provided to the system. This is made possible by using the next two arguments, argv[4] and argv[5], which indicate the path to the silence template. The silence template can be anywhere, but for the tests we saved it in the root of the database folder. The template was copied from the denoised file representing a password spoken by a user from the database. The length of the silence template can vary.

The file that contains the test cases is sent in argv[6]. For more details on this file, please review section A.5.

The information related to the spoken password are sent in argv[7] and argv[8].

The next parameter, argv[9], contains the name of the file to which the results will be appended. This was a safer way to save the results, because the data could be written after each test, without the risk of losing data if the programmed was interrupted.

The name of the true speaker is sent in argv[10] because we wanted to use the results for training purposes also.

After receiving the parameters and if no error was found, the program reads the AVPR_Parameters.ini file to get the location of the training digits database and the registered users (AVPR_speaker_IDs.ini or equivalent). Then, the test cases are read from the file.

For each test case the one-pass dynamic programming will be run. The result for each of them is a vector having the computed distance for every registered user. This vector along with the user names and the tested range are appended to a CString and deposited until all the tests are finished.

After the scores were generated for all the test cases, the CString is appended to the output file. There is also a header appended, which contains the name of the true speaker, the name of the test file and the number of frames of the input.

An example of output is the following (only 3 users are shown):

True Speaker: AC				
File name: AC_13A_car.mfcc				
noOfFrames_input: 91				
Results:				
0	5			
		AC 2.084033	AE 2.299069	AG 2.127659
1	6			
		AC 2.379874	AE 2.434002	AG 2.388726
2	7			
		AC 2.317206	AE 3.158648	AG 2.415888
3	8			
		AC 2.449011	AE 3.117939	AG 2.675370

Due to the fact that the number of test files can grow very fast and the user might loose track of the test files, a Perl script is provided to look for the test mfcc files in the ”Test” folder of the users from a database. This script, based on the input arguments, will automatically create a batch file to test all the found files. A description of the script is given in section 4.7.1.

4.5.2 Interpreting test results

The function reads a test result file and interprets the results of the tests. The first argument has to be ”-i” in order for the function to run.

The parameters received from the command line are the following:

- argv[1]= ”-i” - interpret results
- argv[2]= sourceFile
- argv[3]= destinationFile
- argv[4]= ”-threshold”
- argv[5]= threshold value
- optional
  - argv[6]= ”-u” only those users
  - argv[7]= user1,user2,user3, ... ,userN
- optional
  - argv[8]= ”-r” - a specific range of coefficients

  - argv[9]= N1,N2 - indexes of the coefficients (e.g. 1,13 will include all the coefficients from 0 to 12)
- optional
  - argv[10]= ”-l” - a specific length of password
  - argv[11]= N1,N2 - minimum length and maximum length, both included

The source file (argv[2]) is the output of the test using the ”-t” parameter.

The output file (argv[3]) will be written with the values of the following indicators:

- First, the function will output the binary result of a classification, i.e. will count the number of times when a good decision was taken, and when a bad decision was taken. A decision is considered the correct one if the lowest distance belongs to the true speaker.
- Another quality indicator is the ”safety margin” of a decision. If the true speaker was identified, we want him to have the score as far as possible from the next one. If the user was not correctly identified, we want the score to be as close as possible to the lowest score. To model this request, we introduced a quantitative measure:

$$\Delta = \frac{|d_k - \min_{i \neq k}(d_i)|}{min(d_k - \min_{i \neq k}(d_i))}$$

  If the true speaker (k) obtains the lowest distance in the set, then it is considered that the recognition was correct and a variable that holds a positive margin is increased with Δ.
- The last indicator computed by the function is a confusion matrix. If the function doesn't receive a range of test parameters, the matrix will contain the results of all the test cases.

During the test maybe some outliers are encountered. To eliminate them from the results, the threshold (argv[5]) has to be set to the appropriate value. It will be compared against the value of the current safety margin, Δ.

The next parameters are optional and give the possibility to reflect in the quality indicators only the results of specific users and/or the tests performed on a specific range of coefficients and/or the results obtained for passwords of a specific length.

It is assumed that the set of users and the test cases are consistent across the tested files found in the source file.

4.5.3 Extracting test results

This function extracts from a test result file the scores corresponding to a given range of coefficients. It is called by setting the first argument to ”-e”.

The function has to receive the following parameters from the command line:

- argv[1]= ”-e” - extract results
- argv[2]= sourceFile
- argv[3]= destinationFile
- argv[4]= ”-r” - a specific range of coefficients
- argv[5]= N1,N2 - indexes of the coefficients (e.g. 1,13 will include all the coefficients from 0 to 12)
- optional

  - argv[6]= ”-u” only those users
  - argv[7]= user1,user2,user3, ... ,userN
- optional
  - argv[8]= ”-l” - a specific length of password
  - argv[9]= N1,N2 - minimum length and maximum length, both included

The first line of the output file contains the names of the users.

The following lines will contain the scores obtained by each user from the first column, separated by a ”tab” character. The last element of the line is the index of the true speaker. The function doesn't look for outliers.

The user can also specify the option to extract only a subset of all the users from the test file and/or only the test files that contain a password from a given range of lengths.

The output is intended to be sent to the Matlab functions that perform the optimizations of the probability model's parameters.

4.5.4 Required files, folders and programs

Files

The configuration files needed are the following:

- AVPR_Parameters.ini
- AVPR_speaker_IDs.ini or equivalent - to get the speakers for which the scores will be computed in the test function
- AVPR_Weights.ini - for the testing function
- MFCCTest.ini - for the testing function

For more details on these configuration files please consult section A.

Folders

This project doesn't need any folders to be created.

Programs

This project needs no additional programs.

4.6 Video Tester

The user interface of this program is the command line interface.

The goal of this program is to run the face recognition algorithm on a set of test files and obtain a number of results that can be used for the training of the video recognition probability model.

In order for the program to work, the users from the database have to record a number of test images. The test images have to be stored as 8 bit grayscale pgm files. For more details on this process, please review section 4.2.2.

The program reads the AVPR_Parameters.ini configuration file and gets the database location and the file that contains the registered users. The testing algorithm will be run only for this set of users from the database.

After reading the names of the users, the program reads from the ”Test” folder of each user a number of pictures and creates a list as a String. The number of test pictures is controlled by the ”nPicturesForTest” parameter. This number should have the same value as the one in the AVPersonRecognition project in order to obtain a relevant set of scores for the training algorithm. For more details on how the video recognition algorithm works, please consult subsection ”Video Recognition” of the ”AVPersonRecognition” description.

The list of pictures is then sent to the video recognition algorithm and a set of scores is obtained. After running the algorithm for all the test files found in the database, the output will be written in the ”VideoScores.txt” file. The first line contains the names of the speakers from the database. Each of the following lines contain the scores obtained by each user, followed by the index (starting from 1) of the true user.

The output file can be used by a Matlab script that computes the parameters of the probability model for face recognition for each user.

4.6.1 Required files, folders and programs

Files

The configuration files needed are the following:

- AVPR_Parameters.ini
- AVPR_speaker_IDs.ini or equivalent - to get the speakers for which the scores will be computed

For more details on these configuration files please consult section A.

Folders

This project doesn’t need any folders to be created.

Programs

This project needs no additional programs.

4.7 Additional Programs

Under this title we describe a set of programs written in Matlab or Perl that perform auxiliary functions for the recognition program.

The list of programs is the following:

- Perl:
  - Db_parse.pl
- Matlab:
  - Score fusion algorithms
  - Endpoint detector
  - One pass dynamic programming path viewer
  - Projection algorithms

Some of the Matlab scripts’outputs are needed for the AVPersonRecognition project and others are only used to obtain a graphical representation of an algorithm (“Endpoint detector” and “One pass dynamic programming path viewer”).

4.7.1 Perl scripts - Db_parse.pl

This script is used conjunction with the Audio Tester. Its role is to find the MFCC test files in the ”Test” folder of the users from a database. In the end, a batch execution file is created to automatically test the found files.

The script needs to receive the following parameters:

- 0 - Destination directory to be parsed
- 1 - output file name
- 2 - silence file folder
- 3 - silence file name
- 4 - test cases file name
- 5 - output file name

Based on these parameters, a batch execution file is created. Each test file will have a line associated in the batch file, representing the command to test that file on a database. More information about the output format can be reviewed in section 4.5.1.

The Perl interpreter has to be installed on the host system if this script needs to be used.

4.7.2 Matlab scripts - Score fusion algorithms

During the project there were several scripts developed to find the parameters for the audio and video fusion, depending on the model used. The three model types used are:

1. Exponential Model With Common Rate
2. Gaussian Model
3. Exponential Model With Individual Rate

These models are described in section 3.4.

A.Exponential Model With Common Rate

The optimizations for this model are performed using both audio and video data. The values of the test scores have to be written manually in the file because it was only implemented to test the method.

There are two types of optimizations available for this model:

- Optimization of video parameter only
- Optimization of both parameters using brute force

Optimization of video parameter only The main script that performs this optimization is Optimize_database_only_beta.m. This function maximizes the probability of the training user compared to the probability of the user that has the highest probability in the same scores set.

The algorithm relies on finding the intervals for the video parameter for which only one user has the highest probability compared to all the other users.

For each interval, based on the derivative of the optimization criterion, the best value of the video parameter is computed. The value of the criterion in the optimal point in this interval

is saved in a vector. After all the intervals are verified, the value of the video parameter that maximizes the optimization criterion is chosen as the optimal value.

This function uses two other scripts:

- getIndexForValue.m, which retrieves the index of the user that has the highest probability and is not the training user;
- sortDistinct.m, which sorts the limits of the intervals for which only one user has the highest probability.

Optimization of both parameters using brute force This optimization is performed in the script "ScoreFusion_bruteForce.m". The audio and video scores have to be written manually in the script file. This function optimizes the sum of the ratios between the probabilities of the training user and the probability of the user with the highest probability.

The first step taken is to find the optimal value of the audio parameter. This is also performed by an exhaustive search of the value that maximizes the criterion from equation 3.32, using only the audio scores. The scipt that performs this optimization is called "OptimizeAudio_bruteForce.m".

With this value of the audio parameter, the algorithm computes using an exhaustive search the value of the video parameter that maximizes the value of the optimization criterion from equation 3.33.

B.Gaussian Model

This set of scripts implement the model described in section 3.4.2.

The model was only implemented for the estimation of the audio likelihood.

The four scripts used are:

- OptimizeAudioDatabaseGaussian.m
- EstimateGaussianParametersForUser.m
- EstimateGaussianParameters.m
- errorFunction.m

The "OptimizeAudioDatabaseGaussian.m" script reads all the training data belonging to all the users from a text file, by using the function "readResultsFile.m".

The text file read by "readResultsFile.m" has to have the following structure:

- The first line must contain the names of the users.
- Each of the following line must represent a set of scores, separated by a "tab" character. the scores must have the same order as the users from the first line.
- The last number on each line must represent the index of the training user from the set.
- All the tests belonging to the same user must appear in a contiguous set.

Then the script separates the data belonging to each user and sends it to the second script, "EstimateGaussianParametersForUser.m". This script computes all the gaussian parameters for that user only, based on the training data received. Each subset of the parameters corresponding to the multi-gaussian of the current training user is computed by using the third script, "EstimateGaussianParameters.m". The formulas used for computing the parameters can be found in equations 3.43, 3.53 and 3.54. The corrections for the average and

the variance parameters can be found in equation 3.69. The last script uses a polynomial approximation of the error function, which is implemented in the script "errorFunction.m"

After all the parameters have been computed, the user has the possibility to check the value of the probabilities obtained for the test data and that set of parameters.

C.Exponential Model With Individual Rate

The model of the audio and video data likelihoods are found in equations 3.70 and 3.81.

For this model there were two ways of computing the optimal values of the individual parameters:

- using the MLE from the equation 3.73:

$$\hat{\lambda_i} = \underset{\lambda_i}{\operatorname{argmax}} \left\{ \lambda_i^T \cdot exp\left[-\lambda_i \cdot \sum_{t=1}^{T} \left(d_i(t) - \min_j d_j(t) \right) \right] \right\}$$

- using the optimization criterion from equation 3.79:

$$\Phi(\lambda_i) = \begin{cases} \sum_{t=1}^{T} -\delta \cdot \frac{exp(\lambda_s \cdot [d_s(t) - \min_j (d_j(t))])}{exp(\lambda_i \cdot [d_i(t) - \min_j (d_j(t))])} & , d_i(t) \neq \min_j d_j(t) \\ \sum_{t=1}^{T} -\frac{exp(\lambda_i \cdot [d_i(t) - \min_j (d_j(t))])}{exp(\lambda_s \cdot [d_s(t) - \min_j (d_j(t))])} & d_i(t) = \min_j d_j(t) \end{cases}$$

The first method of parameter estimation was only used in tests and only for the audio data. It is implemented in the "OptimizeAudioDatabaseGaussianDifferenceMin.m" script. The training data is read using the function "readResultsFile.m" and the structure of the text file that contains the data was described earlier. After extracting the minimum distance in the set from the rest of the distances, the function obtains the value of the parameter by dividing the number of tests to the sum of the errors. An error is considered to appear when the minimum value doesn't belong to the true speaker, so the difference between its distance and the minimum distance is non zero.

The second optimization criterion is the one implemented in the AVPersonRecognition project. The optimization for the audio parameters is performed using the script "OptimizeAudioDatabaseNextBestUser.m" and the optimal value for the video parameters are found by calling the script "OptimizeVideoDatabaseNextBestUser.m".

These two functions read the training data and then call the same function, "OptimizeAudio_bruteForce_forUser.m" for each user, to get the optimal value of their parameters. We were able to use the same optimization function because of the expressions of the likelihoods.

For the audio data, if the difference between the distance obtained by the training user and the minimum distance in the set is zero, then the classification is correct. The user probability decreases as the difference between its distance and the minimum distance increases.

For the video data we obtained a behavior similar to the one of the audio model by extracting the user's score from the maximum obtained in the set.

So the difference between the two scripts consists in the operations performed on the training data before being sent to the actual optimization algorithm.

The optimization algorithm, implemented in "OptimizeAudio_bruteForce_forUser.m", takes as an input the data sets belonging to only one training user and implements the optimization criterion from equation 3.79. This function returns the value of the optimal parameter to the calling function.

After all the parameters are computed, the scripts write the obtained values in a file using the function “writeLambdas.m”. For the audio model, the name of the output file is “AVPR_SensorFusion_AudioWeights.ini”. For the video model, the name of the file is “AVPR_SensorFusion_VideoWeights.ini”. These two files contain as a header the number of trained users and on the next lines the name of the users and the value of their corresponding parameter.

These files have to be copied in the AVPersonRecognition project after they are written.

4.7.3 Matlab scripts - Endpoint detector

This script was created after the algorithm presented in [1]. It was implemented in order to be able to visualize the different values and positions of the computed parameters.

The algorithm computes in the beginning the magnitude level of the signal - the sum of the absolute values of the signal for a given time window, and a given time step.

Then, the noise magnitude level is estimated based on an interval at the beginning of the sound file, which is considered to be silence. Based on the noise magnitude level and the signal magnitude level, two thresholds are computed in order to find the beginning and ending of the speech, based on the magnitude level. The two thresholds are ITL and ITU.

Based on these two thresholds, the beginning of the speech (N1 - energy) is found as the first point that has the magnitude level higher than ITL and the signal will reach ITU before it reaches ITL again. The end of the speech period is computed the same way, from the end of the file.

In figure 4.5 we can see the two thresholds and the magnitude of the signal for a sound file which contains the password “4866”.

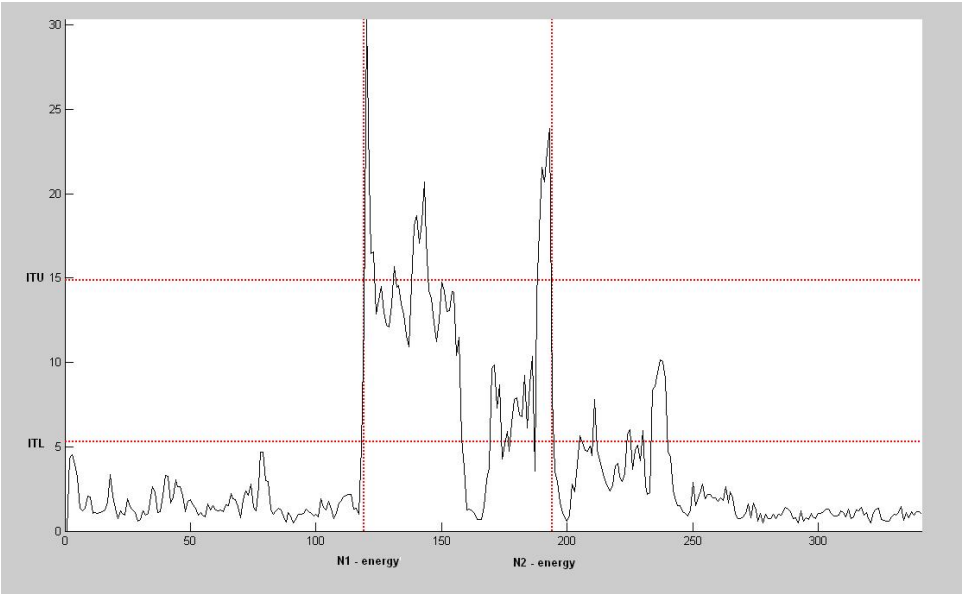

Figure 4.5: Determining the beginning and ending of a speech signal based on magnitude levels

In the implementation we made some threshold and ratios variable so that we could adjust the algorithm to different environments and noise levels.

We changed the starting point of the silence interval in order to avoid a faulty behavior due to the noise present in the beginning of the sound file. This noise could appear because of the time it took the denoising algorithm to adjust the parameters.

We also changed the ratio between the lower energy threshold and the high energy threshold (ITL and ITU) to be able to adjust to different noise levels.

The next step is to find the value of the beginning and ending of the speech signal based on zero cross rate.

This is an iterative process that starts from the previously found endpoints and goes back a number of time frames (which can be set by the user) to see if the signal has a higher zero cross rate than the computed threshold (IZCT). The zero cross threshold is computed based as the minimum value between a fixed number and the noise mean zero cross rate plus two times the noise zero cross rate variance. If on the current search interval there are found at least three time frames that have a zero cross rate higher than the threshold, the last value that is found is remembered and a new search will start from that point. The process ends when the algorithm can't find three time frames with the energy level higher than the threshold, or the end of the file is reached. This iterative algorithm is applied in a similar way to find the end of the speech signal.

In figure 4.6 we show an example of how the estimated endpoints change by using the zero cross rate of the signal.

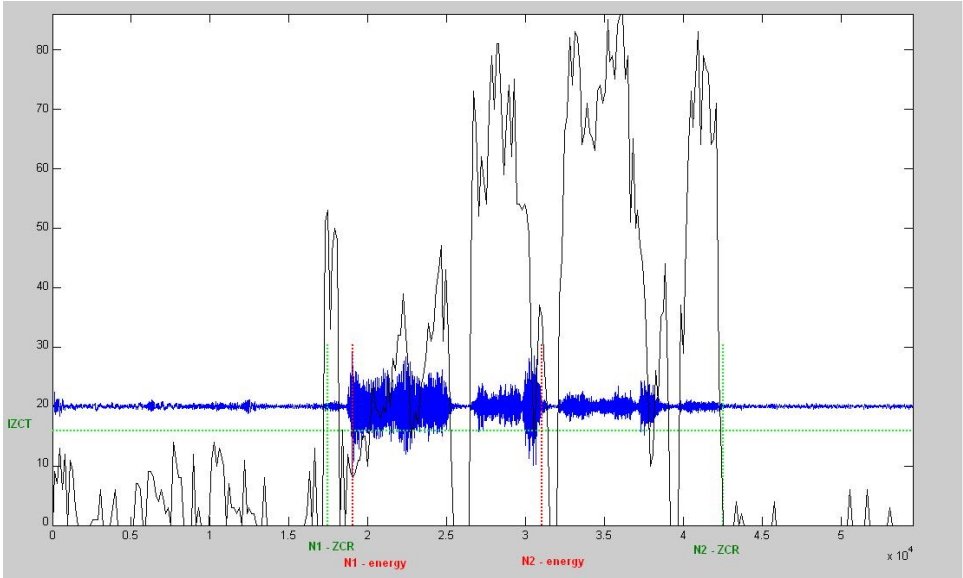

Figure 4.6: Determining the beginning and ending of a speech signal based on zero cross rate

This script also saves the speech signal delimited by the two determined endpoints. A safety interval can also be included.

The implementation in C++ only returns the recommended endpoints, that also include the safety margin, without saving any files on the hard disk.

4.7.4 Matlab scripts - One pass dynamic programming path viewer

This is a tool used for the visualization of the warping path computed by the one pass dynamic programming.

The script is called “PaintOnePassDP.m” and it needs three text files that contain the necessary information to plot the path.

These files are:

- A text file that contains the warping path computed by the one pass dynamic programming algorithm.
- A text file that contains the “Nf” (number of frames) structure to know what is the length of each template from each digit. The values are integers.
- A text file that contains the “Linear” structure, which holds the index of the frames where a template begins. The values are also integers.

This script was used only for debugging purposes, so the code that writes the three files was commented. In order to be able to write the files again, the respective code has to be uncommented.

Based on these structures the script plots the warping path, having on the horizontal the indices of the input file, and on the vertical the indices of the synthesized training data (all the training templates of the digits plus the silence template added in the beginning, in the end and between the templates of the different digits).

In Figure 4.7 there is an example of a warping path having the true speaker as the reference and four templates for each digit. The password spoken is “2118”.

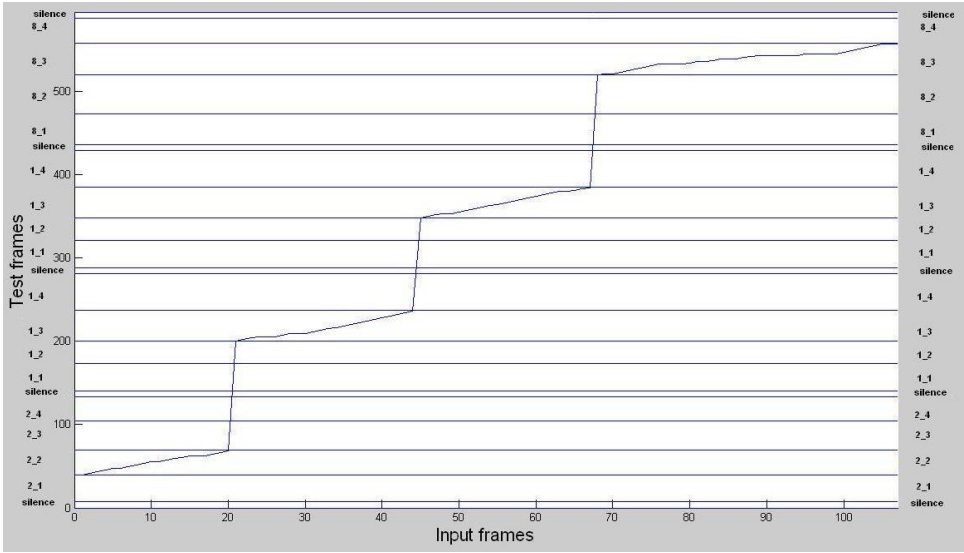

Figure 4.7: Warping path obtained with one pass dynamic programming having as reference the true speaker

As a comparison, the warping path obtained with the one pass dynamic programming having another speaker as reference is presented in Figure 4.8. This speaker has only two templates for each digit.

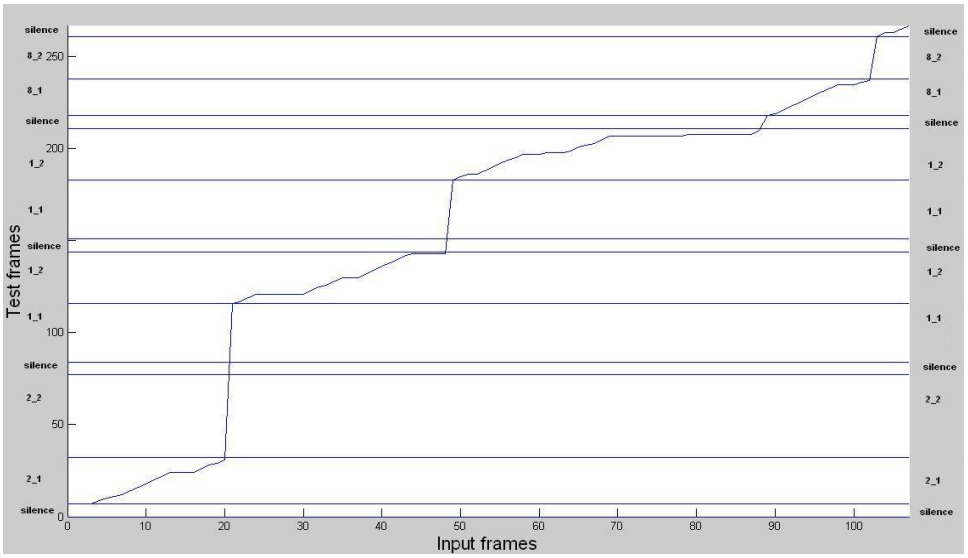

Figure 4.8: Warping path obtained with one pass dynamic programming having as reference another speaker

4.7.5 Matlab scripts - Projection algorithms

This script is used to find the optimal projections described in section 3.5.3.

The Matlab scripts used to implement the algorithm are:

- callGetOptimalProjection.m
- getOptimalProjection.m
- MaxMin_optimized.m
- getNextNuple.m

The algorithm needs as input the average of the distances from the original templates to the average template for the digit of each user. This information has to be found in text files, one file for each symbol. Each file has to have this structure:

- The first line has to contain the range of MFCC coefficients used for the computation of the distances.
- The next lines must contain in the first position the name of the user and then, separated by "tab" characters, the accumulated distances for each coefficient in the range.

The input files are obtained from the output of the algorithm presented in section 4.4.3. Although that algorithm is able to include in the same file multiple ranges of coefficients, the Matlab script only accepts a single range for each symbol.

The entry point of the algorithm is "callGetOptimalProjection.m". This script reads the distances for each digit and sends the data to the second script, "getOptimalProjection.m".

"getOptimalProjection.m" receives the vector that represents the average of the distances obtained for the current trained speaker and another matrix containing the vectors of the other users. All the vectors are normalized by dividing them with the norm of the training

vector. After the normalization, all the vectors of the matrix are projected on the subspace perpendicular on the training user's vector. Only the vectors projected on that subspace (N-1, where N is the number of users) are sent as parameters to the third script, "MaxMin_optimized.m". This is the script that tries to find a projection line (or vector) for which the minimum length of the projections of the points is the maximum possible. So we want the vector that maximizes the length of the minimum projection length.

The best possible solution is when the optimal line is along the input vector that has the smallest norm. This case is valid if the projections of all the other points on this direction have the length greater than the length of the smallest vector. If this is not the optimal vector, none of the directions along the other vectors will be valid. If the case is valid, the algorithm returns this direction and the norm of the vector.

The next step is to increase the number of points taken in consideration. This step will be performed while there is no solution found with the current number of points. The maximum number of points is limited by the dimension of the space or the number of users.

The algorithm will try to find valid directions for all the combinations of points, which represent support points. For each combination of support points, we have to analyze all the possible arrangement of signs of the points. The combinations of support points are generated using the function "getNextNuple.m".

For one of all these possible combinations of points, we give a short description of the algorithm that finds the direction that maximizes the length of projections of these points.

Let's assume that the dimension of the space is *n=4* and the number of points taken in consideration is *nPoints=3*. Let's call these points p_0, p_1, p_2. These three points generate a subspace that is described by *nDiff=2* vectors, v_1 and v_2. v_1 is the vector that unites p_1 with p_0, and v_2 is the point that unites p_2 with p_0. So we can write:

$$v_1 = p_1 - p_0 \tag{4.1}$$

$$v_2 = p_2 - p_0 \tag{4.2}$$

We want to obtain the vector *x* such that two conditions are met simultaneously:

$$x \perp v_1 \; and \; x \perp v_2 \tag{4.3}$$

$$x \in S(v_1, v_2) \tag{4.4}$$

where $S(v_1, v_2)$ is the subspace generated by v_1 and v_2. The equation that describes $S(v_1, v_2)$ is:

$$S(v_1, v_2) = p_0 + s_1 v_1 + s_2 v_2 \tag{4.5}$$

where s_1 and s_2 are two real variables.

Equation 4.3 can be expanded in a system of equations:

$$\begin{cases} v_{11}x_1 + v_{12}x_2 + v_{13}x_3 + v_{14}x_4 = 0 \\ \\ v_{21}x_1 + v_{22}x_2 + v_{23}x_3 + v_{24}x_4 = 0 \end{cases} \tag{4.6}$$

Equation 4.4 can be expanded in the system of equations:

$$\begin{cases} x_1 = p_{01} + s_1 v_{11} + s_2 v_{21} \\ \\ x_2 = p_{02} + s_1 v_{12} + s_2 v_{22} \\ \\ x_3 = p_{03} + s_1 v_{13} + s_2 v_{23} \\ \\ x_4 = p_{04} + s_1 v_{14} + s_2 v_{24} \end{cases} \tag{4.7}$$

We can rewrite the previous equation in the following form:

$$\begin{cases} x_1 & & & - s_1v_{11} - s_2v_{21} = p_{01} \\ & x_2 & & - s_1v_{12} - s_2v_{22} = p_{02} \\ & & x_3 & - s_1v_{13} - s_2v_{23} = p_{03} \\ & & & x_4 - s_1v_{14} - s_2v_{24} = p_{04} \end{cases} \tag{4.8}$$

If we put equations 4.6 and 4.8 in a matriceal form we obtain the equation:

$$A \cdot y = b \tag{4.9}$$

where

- $$A = \begin{array}{|c|c|} \hline V & 0 \\ \hline I & -V^T \\ \hline \end{array} \tag{4.10}$$

  with

  $V \in \Re^{nDiff \times n}$ - the matrix formed with the differences between the support points

  $I \in \Re^{n \times n}$ - the identity matrix.

- $$y = \begin{array}{|c|} \hline x \\ \hline s \\ \hline \end{array} \tag{4.11}$$

  with

  $x \in \Re^{n \times 1}$ - the vector we want to find

  $s \in \Re^{nDiff \times n}$ - the vector containing the slack variables.

- $$b = \begin{array}{|c|} \hline 0 \\ \hline p_0 \\ \hline \end{array} \tag{4.12}$$

Based on these equations we obtain the value for *x*:

$$x = p_0 - V^T(VV^T)^{-1}Vp_0 = \left[I - V^T(VV^T)^{-1}V\right] \cdot p_0 \tag{4.13}$$

With this value of *x*, we try to see if it is a valid direction. The direction of *x* is valid if the projections of all the points that are not in the support vector set are longer than the norm of *x*. If it is valid, the algorithm will store it in a matrix, along with the norm.

After all the valid directions for all the combinations of support points are found, the algorithm chooses the direction with the maximum norm.

The value returned by the "MaxMin_optimized.m" will be returned to "callGetOptimalProjection.m" which will store it in an array. After all the projections for the current digit have been computed, the algorithm starts computing the projections for the next digit.

After all the projections have been computed, the script writes these values into a text file that has the structure:

- a header containing the number of symbols, the number of users and the number of coefficients.
- a section for each symbol, that begins with the symbol and each of the following lines contain the projection vectors, along with the user names.

Chapter 5

Conclusions and future work

An audio and video sensor fusion based person recognition system was implemented. The system has a very easy to use interface, is flexible and can be used in future researches.

During the project we tried to improve the audio recognition module by performing extensive searches of the best MFCC ranges of coefficients.

We have also tried to find a set of weights for the Euclidean distance computation during the audio matching process and we obtained better results for the tests that had a relative high SNR.

One last optimization for the audio processing module, inspired from the Fisher Linear Discriminant and Support Vector Machines, was implemented, but due to time constraints, it was not thoroughly tested.

Several models for the audio and video likelihoods have been tested and for each one of them an optimization algorithm was proposed.

The Sensor fusion algorithm was able to improve the overall accuracy of the system.

But there were also difficulties encountered during the development.

In the audio recognition algorithm, we observed a very high sensibility of the obtained set of distances relative to the trimming of the templates from the database and of the test files. A possible improvement would be to use newer approaches based on Voice Activity Detection for the trimming of the audio files.

Another improvement would be related to the audio score computation based on the weighted euclidean distances. A set of weights based on the inter-speaker variability or intra speaker variability could make it possible to obtain better classification results.

Another important thing we would need to consider is the functioning of the system on an open-set basis. The system as it was implemented was only performing recognition for a closed set of users. Changing this feature would also need finding newer models for the likelihoods, that would take into consideration the existence of a user that is not registered in the system.

Appendix A

Description of Configuration Files

The programs use a set of files that initialize different algorithms. The files and the reading/writing functions are the same for all the programs, although each program can store its own parameters. Also, not all the files are needed for all the programs.

The list of all the configuration files is the following:

- AVPR_Parameters.ini
- AVPR_speaker_IDs.ini
- AVPR_SensorFusion_AudioWeights.ini
- AVPR_SensorFusion_VideoWeights.ini
- AVPR_Weights.ini
- MFCCtest.ini

A.1 AVPR_Parameters.ini

This is the main configuration file of the program. It contains the values of the configuration parameters for audio and video processing, and the paths or file names of other configuration files or programs.

The name of this file cannot be changed. The program exits if the file is not found. Trough this configuration you can change the name of other initialization files or paths that the program will look for (user IDs file or database path).

The comments can be added after the ”%” sign.

The parameters that can be configured trough this file are the following:

- AVPR_silencesize_ms
  The length in milliseconds of the estimated silence considered in the audio recognition programs. The data of this length will be copied from the beginning of the audio file and an offset, and considered as the silence template used in the one pass dynamic programming algorithm.
- AVPR_silenceOffset_ms
  The offset (in ms) of the start of the silence template, considered from the beginning of the file.

- AVPR_numOfMFCCs
  The number of coefficients that will be read from a MFCC file. It is equal to the number of columns of data.
- AVPR_FS
  Sampling frequency (in Hz) used in the recording of the audio data.
- AVPR_MFCC_PREEMPHASIS
  Boolean variable used to enable or disable the use of the preemphasis filter, when the computation of the MFCC is performed.
- AVPR_MFCC_DIM_START
  Starts from 0.

  - For score computation: The order of the first coefficient included in the range used in computing the distance between two feature points.

  - For feature extraction: The order of the first coefficient that will be included in the range of coefficients saved to an MFCC file.
- AVPR_MFCC_DIM_END
  Starts from 0.

  - For score computation: The order of the first coefficient that will not be included in the range used in computing the distance between two feature points.

  - For feature extraction: The order of the first coefficient that will not be included in the range of coefficients saved to an MFCC file.
- AVPR_MFCC _NUMBER_COEFFICIENTS
  The total number of the coefficients or filters used in the computation of the MFCCs. The length of used frequency interval will be divided in this number of log equal spaced filters.
- AVPR_MFCC_F_START
  The starting frequency (in Hz) used in computing the MFCC filters. Default is 200Hz.
- AVPR_MFCC_F_END
  The ending frequency (in Hz) used in computing the MFCC filters. Default is 8000Hz.
- AVPR_WORDISOLATOR_GAINCOEFF
  The value of the ratio between ITU and ITL thresholds used in determining the candidate beginning or ending of speech, based on energy.
- AVPR_WORDISOLATOR_TOLLERANCE_START_MS
  Length of the safety margin that can be added to the recommended beginning point of the speech, in case the algorithm cuts too much of the sound wave.
- AVPR_WORDISOLATOR_TOLLERANCE_END_MS
  Length of the safety margin that can be added to the recommended end point of the speech, in case the algorithm cuts too much of the sound wave.
- AVPR_WORDISOLATOR_FRAMELENGTH_MS
  The length in milliseconds of the window used in computing the values of signal magnitude and zero-cross rate.
- AVPR_WORDISOLATOR_STEPLENGTH_MS
  The length in milliseconds between the beginning of two consecutive time frames used in computing the values of the signal magnitude and zero-cross rate.
- AVPR_WORDISOLATOR_IGNORE _INITIAL_SILENCE_MS
  The offset in milliseconds from the beginning of the sound wave, that marks the beginning of the silence data used in the noise estimation.

- AVPR_WORDISOLATOR_INITIAL_SILENCE_LENGTH_MS
  Te length in milliseconds of the silence data used to estimate the noise in the file that has to be trimmed.
- AVPR_WORDISOLATOR_STEPBACK_MS
  The length in ms of the interval that is tested to see if it contains weak fricatives or plosive sounds. If there are found three points in this interval that have the zero-cross rate higher than a threshold, the beginning or ending point is moved where the last point was found, and the process is continued until less than three of this points are found.
- AVPR_WORDISOLATOR_USEZEROCROSS
  Boolean variable used to enable or disable the search of the speech endpoints based on zero-cross rate.
- AVPR_WORDISOLATOR_I2TOIMNRATIO
  Ratio between I2 threshold and the maximum magnitude of the noise. It influences the value of the ITL threshold.
- AVPR_WORDISOLATOR_ZEROCROSS_TRESHOLD
  Value used in computing an auxiliary threshold for the zero-cross rate. Represents the fraction of the value of ITL. If the value of the signal magnitude is smaller than the auxiliary threshold, the zero-cross rate is set to 0.
- AVPR_SENSORFUSION_VIDEOWEIGHT
  NOT USED IN FINAL VERSION. Value of the exponential factor used in computing the posterior probability of a user based on video data.
- AVPR_SENSORFUSION_AUDIOWEIGHT
  NOT USED IN FINAL VERSION. Value of the exponential factor used in computing the posterior probability of a user based on audio data.
- speakerIDsFile
  Name of the speaker IDs file. It has to be in the same folder as the executable.
- audioDatabasePath
  CString variable that holds the path to the audio and video database.
- AVPR_EXTERNAL_SOUND_EDITOR
  Variable that holds the path to an external sound editor. The program has to accept command line parameters. If the file is not found, the program will play the file using an internal function.
- AVPR_SENSORFUSION_VIDEOWEIGHT_FILE
  Variable representing the name of the weights of the exponentials used in computing the posterior probability of a user based on video data.
- AVPR_SENSORFUSION_AUDIOWEIGHT_FILE
  Variable representing the name of the weights of the exponentials used in computing the posterior probability of a user based on audio data.

For more details on the word isolator parameters, please see [1].

The structure of the configuration file is the same for all projects; the only thing changed is the values of the parameters. The functions that read, write or display the parameters are also common for all the projects.

The strings have to be delimited by double quotes.

For the Database creator, AV Person Recognition and Database utilities the parameters can be changed trough a dialog at runtime.

A.2 AVPR_speaker_IDs.ini

The function of this file is to provide to some of the programs a list with the users from the database included in the tests. The programs will only take from the database the data that belong only to those users, even if the database contains more users.

The file has to contain a column with the names of the users. In order for the program to run properly, all the users have to be in the database folder.

This file has to be updated after the creation of a user with the Database Creator program. It is not used in the Database Creator project.

You can specify a different file name, but the name has to be reflected in the corresponding parameter in the "AVPR_Parameters.ini".

A.3 AVPR_SensorFusion_AudioWeights.ini and AVPR_SensorFusion_VideoWeights.ini

These two files are used only in the AVPersonRecognition project.

The audio and video probabilities are computed using the formula:

$$P(Audio|i) = \lambda_i \cdot exp(-\lambda_i \cdot (dist_i - \min_j(dist_j)) \tag{A.1}$$

$$P(Video|i) = \gamma_i \cdot exp(-\gamma_i \cdot (\max_j(score_j) - score_i) \tag{A.2}$$

The AVPR_SensorFusion_AudioWeights.ini file contains the values of λ_i for all the users and the AVPR_SensorFusion_VideoWeights.ini file contains the values for γ_i for the users. The files can include all the users in the database, but the program will only read the ones of the users registered in the AVPR_speaker_IDs.ini (or equivalent) file.

The files have to include a header that specifies the number of users in the file (for example: "noOfUsers 8"), and then an empty line. After the empty line each user and its weight will be written on each line, separated by a tab character.

You can change the name of the files, but you have to make the necessary changes in the "AVPR_Parameters.ini" file.

The values written in the file are obtained using a training algorithm.

A.4 AVPR_Weights.ini

This file is only needed in audio recognition, when the one pass dynamic programming algorithm is applied on two sets of data. It contains the weights of the different MFCC coefficients used in the computation of the Euclidean distance between two sets of features.

The file contains two lines. On the first line, separated by "tab" characters, there are the weights. This is the only line that is read. The second line contains the indexes of the coefficients in order for the file to be easier to read.

The name of the file cannot be changed in the "AVPR_Parameters.ini". It has to be in the same folder as the program that needs it.

It is not needed in the Database Creator and Video Tester projects.

A.5 MFCCtest.ini

This file is only needed in the Audio Tester and Database Utilities file.

From this file the programs need to read the range of coefficients for which some tests will be performed. Each line contains a test, the beginning and the end of the range being separated by a ”tab” character.

In the case of the Audio Tester it is only needed when testing an MFCC file (”-t” parameter sent in the command line), but the name can be changed and sent as a parameter in the command line.

Appendix B

An Algorithm for Determining the Endpoints of Isolated Utterances

An Algorithm for Determining the Endpoints of Isolated Utterances

By L. R. RABINER and M. R. SAMBUR

(Manuscript received June 10, 1974)

An important problem in speech processing is to detect the presence of speech in a background of noise. This problem is often referred to as the endpoint location problem. By accurately detecting the beginning and end of an utterance, the amount of processing of speech data can be kept to a minimum. The algorithm proposed for locating the endpoints of an utterance is based on two measures of the signal, zero crossing rate and energy. The algorithm is inherently capable of performing correctly in any reasonable acoustic environment in which the signal-to-noise ratio is on the order of 30 dB or better. The algorithm has been tested over a variety of recording conditions and for a large number of speakers and has been found to perform well across all tested conditions.

I. INTRODUCTION

The problem of locating the beginning and end of a speech utterance in an acoustic background of silence is important in many areas of speech processing. In particular, the problem of word recognition is inherently based on the assumption that one can locate the region of the speech utterance to be recognized. A further advantage of a good endpoint-locating algorithm is that proper location of regions of speech can substantially reduce the amount of processing required for the intended application.

The task of separating speech from background silence is not a trivial one except in the case of acoustic environments with extremely high signal-to-noise ratio, e.g., an anechoic chamber or a soundproof room in which high-quality recordings are made. For such high signal-to-noise ratio environments, the energy of the lowest-level speech sounds (e.g., weak fricatives, low-level voiced portions, etc.) exceeds the background noise energy and a simple energy measure suffices.[1] However, such ideal recording conditions are not practical for real-world applications of speech-processing systems. Thus, simple energy measures are not sufficient for separating weak fricatives (such as the /f/ in "four") from background silence. In this paper, we propose a fairly simple algorithm for locating the beginning and end of an utterance, which can be used in almost any background environment with a signal-to-noise ratio of at least 30 dB. The algorithm is based on two measures of speech: short-time energy and the zero crossing rate. The algorithm possesses the feature that is somewhat self-adapting to the background acoustic environment in that it obtains all the relevant thresholds on its decision criteria from measurements made directly on the recorded interval.

The organization of this paper is as follows. In Section II we discuss the major difficulties in locating the beginning and end of an utterance and propose various measurements for distinguishing between speech and no speech in these cases. In Section III we describe the algorithm to locate the endpoints of the utterance. In Section IV we give examples of the use of the algorithm, and give the results of both formal and informal tests on its ability to find endpoints of a corpus of words from several speakers. Finally, in Section V we discuss the general characteristics of the endpoint-location problem and propose alternative methods of solving the problem.

II. EXAMPLES OF SPEECH ENDPOINT-LOCATION PROBLEMS

To arrive at a reasonable algorithm for separating speech from nonspeech, it is necessary first to define the acoustic environment in which the recordings are made. In this paper, we consider two specific modes of recording. In the first mode, the speaker makes recordings on analog tape using a high-quality microphone in a soundproof room. This mode of recording is useful for obtaining reasonably high-quality speech. In the second mode of recording, the speaker records directly into computer memory in a noisy environment (e.g., a computer room) using a noise-reducing, close-talking microphone. This mode of recording is a reasonable approximation to a real-world environment for most man-machine interaction problems. To eliminate 60-Hz hum, as well as any dc level in the speech, it is assumed that the speech is high-pass filtered above 100 Hz; similarly, to keep the processing simple, the speech is low-pass filtered at 4 kHz, thereby allowing a 10-kHz sampling frequency.

Figure 1 shows a comparison of the waveform* of the background silence (on a greatly amplified scale) for these two modes of recording. The top two lines of this figure show the waveform for tape-recorded

* In this and subsequent illustrations, each line shows 25.6 ms of the waveform. Successive lines show successive 25.6-ms segments of the waveform.

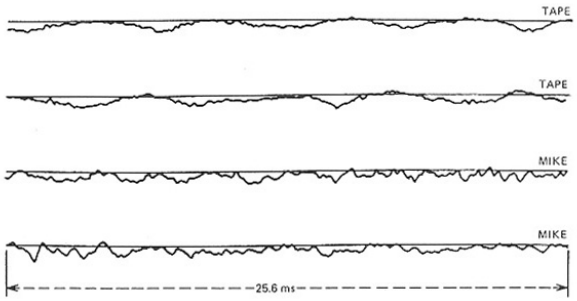

Fig. 1—Acoustic waveforms for the silences from tape and microphone.

silence from a soundproof booth, whereas the lower two lines show the waveform for the silence from the close-talking microphone. It is seen from this figure that the tape-recorded silence has a strong low-frequency component (period ≈ 8 ms) due to the recording process. The waveforms from both the close-talking microphone and the recording process appear to be quite broadband, as one would expect. Figure 2 shows typical frequency spectra of these background silences. The spectra are plotted on a log magnitude scale and are for 512-point Hamming window weighted sections. Except for the strong low-frequency-hum components for the recorded silence, the spectra of these silences are quite similar.

The problem of locating the endpoints of an utterance in these backgrounds of silence essentially is one of pattern recognition. The way one would attack the problem by eye would be to acclimate the eye (and brain) to the "typical" silence waveform and then try to spot some radical change in the pattern. In many cases this is easy to do. Figure 3 shows an example (a waveform of the word "eight") in which the silence pattern (on a reduced amplitude scale) is easily distinguished from the speech which begins just past the beginning of the third line on this figure. What one is observing in this case is a radical change in the waveform energy between the silence and the beginning of the speech.

Figure 4 shows another example (a waveform of the word "six") in which the eye can do an excellent job in locating the beginning of the speech. In this case, the frequency content of the speech is radically different from the frequency content of the background noise as manifested by the sharp increase in the zero crossing (or level crossing)

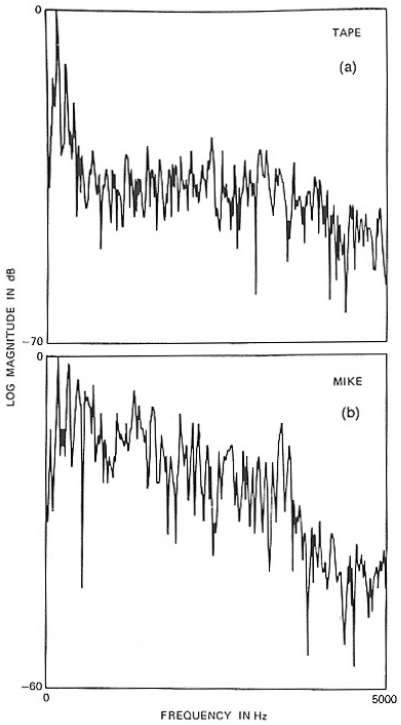

Fig. 2—Log magnitude spectra for the silences from tape and microphone.

rate of the waveform. For this example, the speech energy at the beginning of the utterance is not radically higher than the silence energy; however, other characteristics of the waveform signal the beginning of the speech.

The next set of figures illustrates some of the cases in which the eye can be greatly deceived, even with the use of expanded amplitude scales to aid in the examination of the frequency content of the speech. Figure 5 shows the waveform for the beginning of the utterance "four."

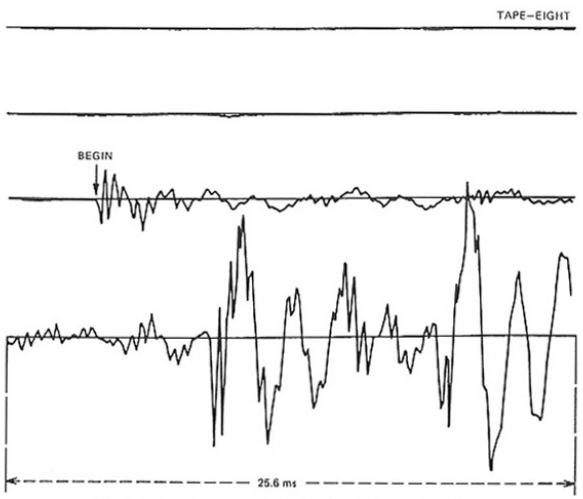

Fig. 3—Waveform for the beginning of the word "eight."

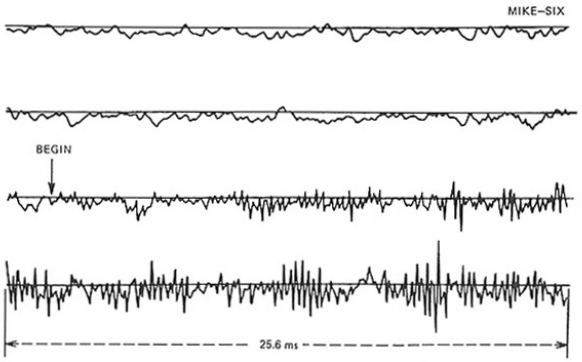

Fig. 4—Waveform for the beginning of the word "six."

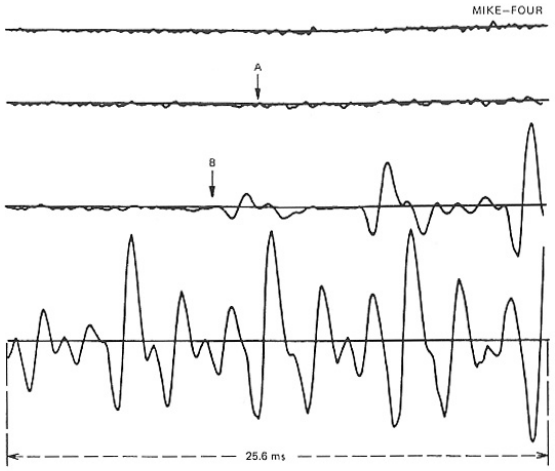

Fig. 5—Waveform for the beginning of the word "four."

This utterance begins with the weak fricative /f/. Without any *a priori* information about the utterance, the eye would select point B as the beginning of the utterance. This is incorrect, however, in that it completely misses the weak fricative /f/ at the beginning. For this example, point A is a better indication of the beginning of the speech.* Thus, one problem to be concerned with is weak fricatives at the beginning (or end) of the utterance.

Figure 6 shows another example of the difficulty in locating the endpoint of an utterance. This figure shows the waveform for the end of the word "five." Without any *a priori* information, point A might be chosen by eye as the endpoint of the utterance. However, the actual endpoint occurs approximately at point B. In this example, the final /v/ in "five" becomes devoiced and turns into an /f/, a weak fricative. Such weak fricatives are difficult to locate by eye (and sometimes even by ear).

* The criterion for deciding the actual beginning and ending points of the utterances was to use a combination of careful listening combined with precise visual examination of the waveform.

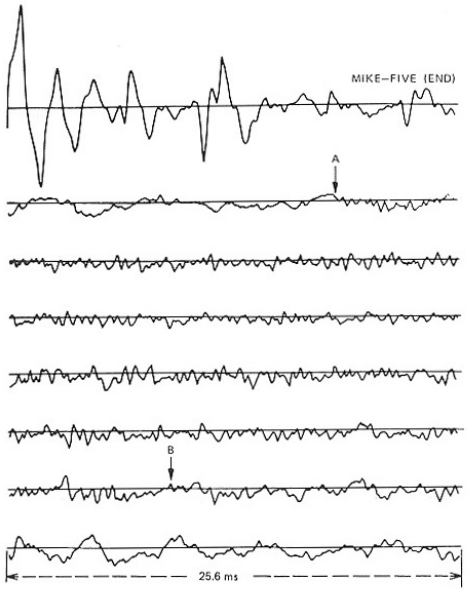

Fig. 6—Waveform for the end of the word "five."

As a final example, Fig. 7 shows the waveform for the end of the word "nine." It is quite difficult to say where the final nasal ends and where the silence begins. A reasonable location for the endpoint is the point marked END in this figure, although it is not clear how accurate this choice actually is.

Rather than give several more examples of situations in which it is difficult to locate either the beginning or the end of an utterance, we list below the broad categories of problems encountered. These include:

(*i*) Weak fricatives (/f, th, h/) at the beginning or end of an utterance.
(*ii*) Weak plosive bursts (/p, t, k/).
(*iii*) Final nasals.

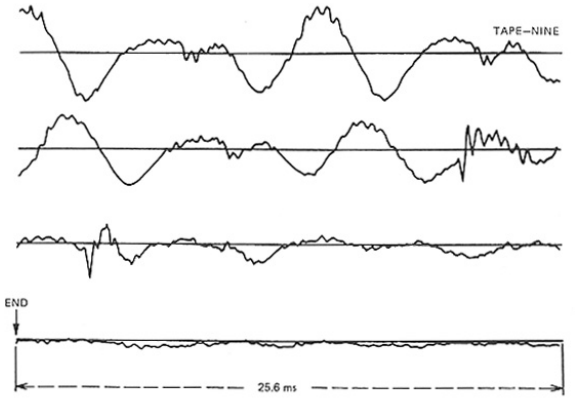

Fig. 7—Waveform for the end of the word "nine."

(*iv*) Voiced fricatives at the ends of words which become devoiced.
(*v*) Trailing off of certain voiced sounds—e.g., the final /i/ becomes unvoiced sometimes in the words "three" (/th-r-i/) or "binary" (/b-aI-n-e-r-i/).

The approach we have taken to solve these problems in an automatic endpoint-location algorithm is a pragmatic one. Our goal is to isolate enough of the word (utterance) so that a reasonable acoustic analysis of what is obtained is sufficient for accurate recognition of the word. Thus, it is not necessary to locate *exactly* the point where the word begins or ends, but instead it is important to include all significant acoustic events within the utterance. For a word like "binary," it is of little consequence if the trailing off unvoiced energy is omitted (in fact, it is probably quite helpful for a "phonetic" word-recognition strategy); however, for a word like "four" it is important to be able to reliably locate and include the initial weak fricative /f/. For this last example, the word "four," it is not necessary to include the entire initial unvoiced interval; in fact, experience has shown that 30 to 50 ms of unvoiced energy is sufficient for most word-recognition purposes. This type of knowledge is of great importance in an endpoint-finding algorithm because it enables you to set conservative values on all decision thresholds (thereby guaranteeing a very low false-alarm rate) and, for the word-recognition application, the concomitant

high miss rate will be of little practical significance. In Section III, we give the details of one practical implementation of an endpoint-location algorithm.

III. THE ENDPOINT-LOCATION ALGORITHM

Based on the preceding discussion, the goals of the endpoint algorithm are:

(*i*) Simple, efficient processing.
(*ii*) Reliable location of significant acoustic events.
(*iii*) Capability of being applied to varying background silences.

The first goal implies that only simple measurements can be made on the speech waveform as a basis for the decision. If speed and simplicity were not major issues, far more sophisticated processing could be used to give a better, more accurate result.

With the above considerations in mind, the endpoint location algorithm that was implemented is based on two simple measurements, energy and zero crossing rate, and uses simple logic in the final decision algorithm. Both energy and zero crossing rate are simple and fast to compute, and, as seen in Section II, can give fairly accurate (although conservative) indications as to the presence or absence of speech. Before proceeding to a description of the algorithm, we first define how the energy and zero crossing rate are measured. The speech "energy," $E(n)$, is defined as the sum of the magnitudes of 10 ms of speech centered on the measurement interval,[2] i.e.,

$$E(n) = \sum_{i=-50}^{50} |s(n+i)|, \tag{1}$$

where $s(n)$ are the speech samples and it is assumed that the sampling frequency is 10 kHz. The choice of a 10-ms window for computing the energy and the use of a magnitude function rather than a squared-magnitude function were dictated by the desire to perform the computations in integer arithmetic and, thus, to increase speed of computation. Further, the use of a magnitude de-emphasizes large-amplitude speech variations and produces a smoother energy function. By way of example, Fig. 8 shows typical energy functions for the words "directive" and "multiply." (The beginning and end of these words is noted on these energy plots.) For this example, the energy function is computed once every 10 ms, or 100 times per second.

The zero (level) crossing rate of the speech, $z(n)$, is defined as the number of zero (level) crossings per 10-ms interval. Although the zero crossing rate is highly susceptible to 60-Hz hum, dc offset, etc., in

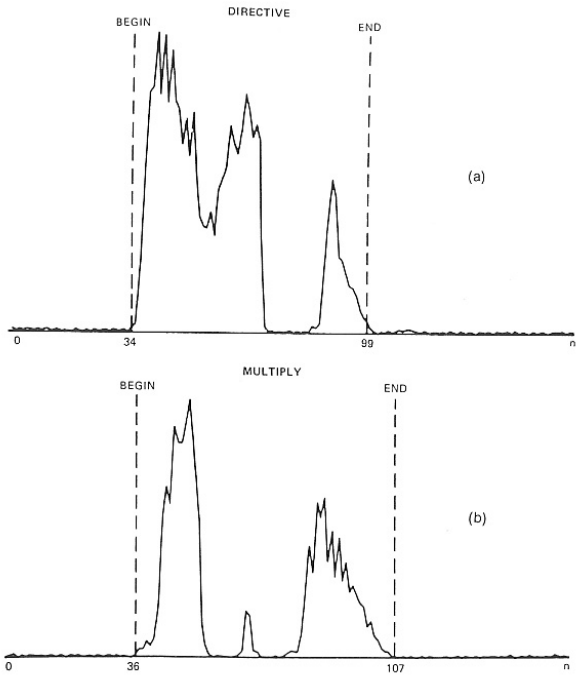

Fig. 8—Typical energy plots for the words "directive" and "multiply" with markers indicating the beginning and end of the utterance.

most cases it is a reasonably good measure of the presence or absence of unvoiced speech.

Figure 9 shows a flowchart of the endpoint-location algorithm. The speech waveform is filtered prior to sampling at 10 kHz by a bandpass filter with a 100-Hz low-frequency cutoff and a 4000-Hz high-frequency cutoff and having 48 dB per octave skirts. It is assumed that during the first 100 ms of the recording interval there is no speech present. Thus, during this interval, the statistics of the background silence are measured. These measurements include the average and

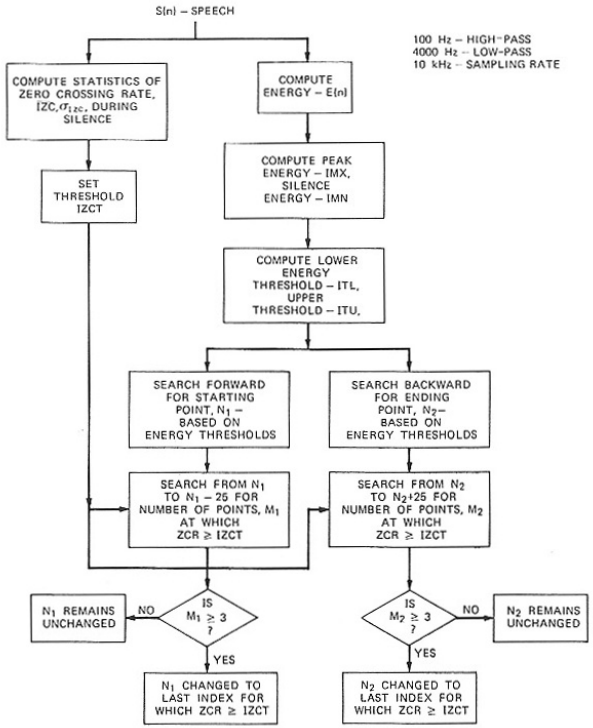

Fig. 9—Flowchart for the endpoint algorithm.

standard deviation of the zero crossing rate and the average energy. If any of these measurements are excessive, the algorithm halts and warns the user. Otherwise, a zero crossing threshold, $IZCT$, for unvoiced speech is chosen as the minimum of a fixed threshold, IF (25 crossings per 10 ms), and the sum of the mean zero crossing rate during silence, $\overline{IZC}$, plus twice the standard deviation of the zero crossing rate during silence, i.e.,

$$IZCT = \text{MIN}(IF, \overline{IZC} + 2\sigma_{IZC}). \tag{2}$$

The energy function for the entire interval, $E(n)$, is then computed. The peak energy, IMX, and the silence energy, IMN, are used to set two thresholds, ITL and ITU, according to the rule

$$I1 = 0.03*(IMX - IMN) + IMN \tag{3}$$

$$I2 = 4*IMN \tag{4}$$

$$ITL = \text{MIN}(I1, I2) \tag{5}$$

$$ITU = 5*ITL. \tag{6}$$

Equation (3) shows $I1$ to be a level which is 3 percent of the peak energy (adjusted for the silence energy), whereas (4) shows $I2$ to be a level set at four times the silence energy. The lower threshold, ITL, is the minimum of these two conservative energy thresholds, and the upper threshold, ITU, is five times the lower threshold.

The algorithm for a first guess at the endpoint locations is shown in Fig. 10. The algorithm begins by searching from the beginning of

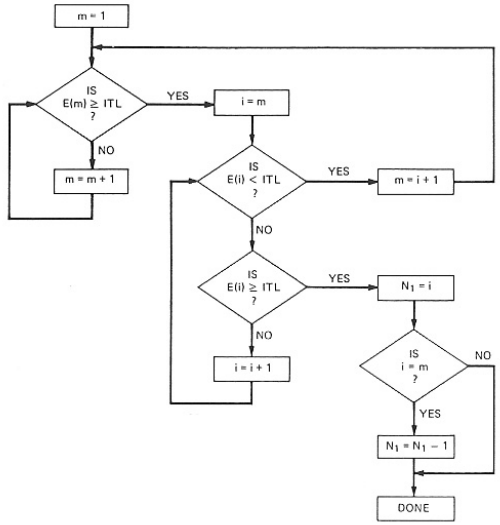

Fig. 10—Flowchart for the beginning point initial estimate based on energy considerations.

the interval until the lower threshold is exceeded. This point is preliminarily labeled the beginning of the utterance unless the energy falls below ITL before it rises above ITU. Should this occur, a new beginning point is obtained by finding the first point at which the energy exceeds ITL, and then exceeds ITU before falling below ITL; eventually such a beginning point must exist. A similar algorithm (shown in Fig. 11) is used to define a preliminary estimate of the endpoint of the utterance. We call these beginning and ending points N_1 and N_2, respectively.

Until now, we have only used energy measurements to find the endpoint locations; and these endpoint locations are conservative in that fairly tight thresholds are used to obtain these estimates. Thus, at this point, it is fairly safe to assume that, although part of the utterance may be outside the (N_1, N_2) interval, the actual endpoints are not within this interval. In relation to this, the algorithm proceeds to

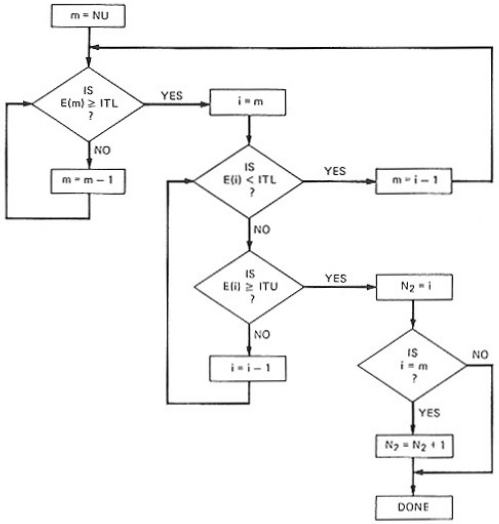

Fig. 11—Flowchart for the ending point initial estimate based on energy considerations.

examine the interval from N_1 to $N_1 - 25$, i.e., a 250-ms interval preceding the initial beginning point, and counts the number of intervals where the zero crossing rate exceeds the threshold $IZCT$. If the number of times the threshold was exceeded was three or more, the starting point is set back to the first point (in time) at which the threshold was exceeded. Otherwise, the beginning point is kept at N_1. The rationale behind this strategy is that for all cases of interest, exceeding a tight threshold on zero crossing rate is a strong reliable indication of unvoiced energy. Of course, it is still possible that a weak fricative will not pass this test, and will be missed. However, in these cases there is no simple, *reliable* method of distinguishing such a weak fricative from background silence.

A similar search procedure is used on the endpoint of the utterance to determine if there is unvoiced energy in the interval from N_2 to $N_2 + 25$. The endpoint is readjusted based on the zero crossing test results in this interval.

To illustrate the use of the endpoint algorithm, Fig. 12 shows representative contours of the energy and zero crossings for an utterance. Using the energy criterion alone, the algorithm chooses the point N_1 as the beginning of the utterance and N_2 as the end of the utterance. By searching the interval from N_1 to $N_1 - 25$, the algorithm finds a large number of intervals with zero crossing rates exceeding the threshold; thus, the beginning point is moved to $\hat{N}_1$, the first point (in time) that exceeded the zero crossing threshold. Similar examination of the interval from N_2 to $N_2 + 25$ shows no significant number of

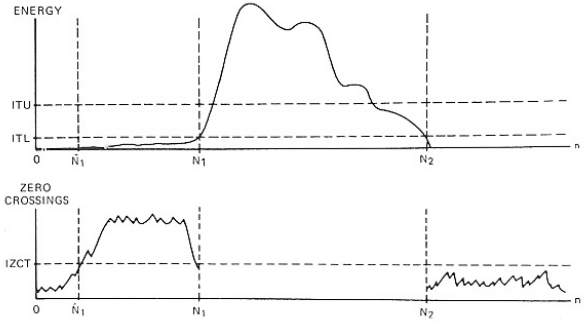

Fig. 12—Typical example of energy and zero crossings data for a word beginning with a strong fricative.

intervals with high zero crossings; thus, the point N_2 is retained as the endpoint of the utterance.

In Section IV, we give examples of the use of the endpoint algorithm for a large number of words with different speakers and different acoustic environments.

IV. EXAMPLES OF THE USE OF THE ENDPOINT ALGORITHM

The endpoint algorithm described in Section III was implemented on the DDP-516 computer facility of the Bell Laboratories Acoustics Research Department. The algorithm was tested using the two modes of recording described in Section II: high-quality tape recordings from a soundproof booth and on-line recordings using a close-talking microphone.

Figures 13 and 14 show examples of how the algorithm worked on typical isolated words. In Fig. 13 there are eight plots of the energy function for eight different words (of two different speakers). Some of the words were recorded on-line (marked MIKE) and others were recorded on tape (marked TAPE) from the soundproof booth. The markers on each plot show the beginning point and ending point of each word, as determined by the automatic algorithm. For the example in Fig. 13a (the word "nine"), the energy thresholds were sufficient to locate the endpoints. For the example in Fig. 13b (the word "replace"), the zero crossing algorithm was used to determine the ending point due to the final fricative /s/. It should be noted that even though the final /s/ has fairly large energy, since the energy thresholds were set conservatively, the energy criterion was not able to find the actual endpoint of the word. Instead, the zero crossing algorithm was relied upon in this case. In Fig. 13c, the final /t/ in the word "delete" was correctly located because of the significant zero crossing rate over the 70-ms burst when the /t/ was released. Thus, even though there was little energy or zero crossing activity for about 50 ms in the stop gap, the algorithm was able to correctly identify the endpoint because of the strength of the burst. On the other hand, if the burst had been weak, the ending point would have been located at the beginning of the stop gap.

Figure 13d is an example in which the energy during the silence was significant in a couple of places prior to the beginning of the word "subtract," yet the algorithm successfully eliminated these places from consideration because of the low zero crossing rates. In this example, a relatively weak burst in the final /t/ was correctly labeled as the endpoint.

Figures 13e through 13h show examples of words with fricatives at either the beginning or end of the word. In all cases, the algorithm was

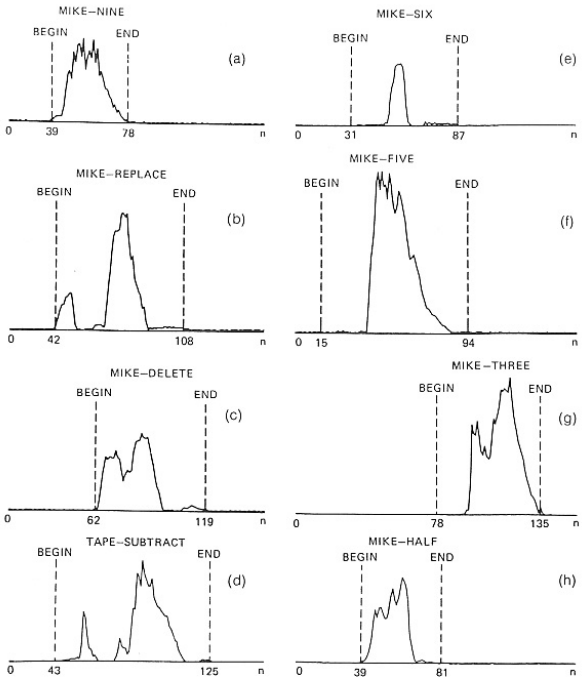

Fig. 13—Sequence of energy plots showing how the endpoint algorithm performed over a variety of words.

able to correctly place the appropriate endpoint so that a reasonable amount of unvoiced duration was included within the boundaries of the word.

Figure 14 shows three examples of how the algorithm performed for the word "four." It can be seen from the location of the beginning point that, although the level of the initial /f/ varied from strong to weak, the zero crossing indicator was able to find positive indications of the frication noise in all three cases. As discussed earlier, there are many examples where initial or final fricatives (mainly /f/ and /th/)

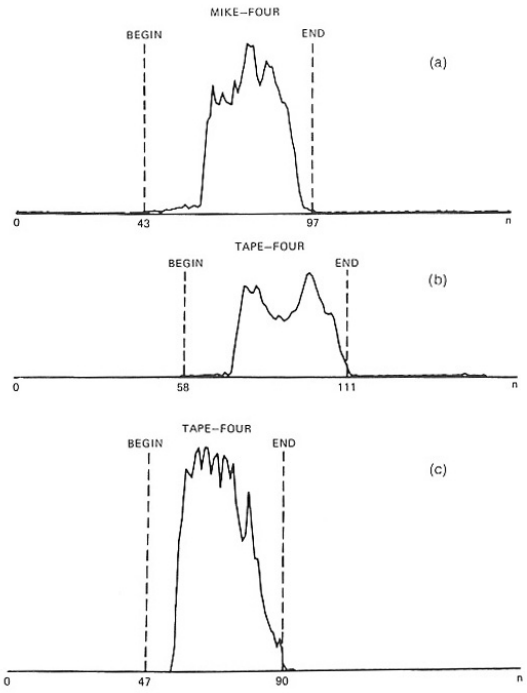

Fig. 14—Energy plots and endpoint assignments for three variations of the word "four."

were so weak they were indistinguishable from the background silence. In Section V, we discuss more sophisticated techniques for distinguishing such weak fricatives from background silence.

Two sets of formal tests were made on the algorithm. In one test, the 54-word vocabulary used by B. Gold in his word-recognition experiments[3] was read by two males and two females. For this vocabulary, the algorithm made no gross errors in locating the beginning and ending points. The algorithm did make a number of small errors of the type discussed earlier, such as losing weak fricatives or releases of

stops; however, none of these errors seriously affected the human recognition (based solely on listening) of the utterance from the portion that the algorithm did locate correctly. Thus, in some pragmatic sense, such errors can be tolerated for word recognition purposes; although for such applications as computer voice response, these small errors would probably be significant.

The second test involved 10 speakers each repeating the 10 digits from zero to nine in 10 separate sessions. (These data were actually measured for a digit-recognition experiment that used this endpoint location algorithm.) For this test, there were essentially no gross errors in locating the endpoints; in fact, it was determined that for purposes of word recognition, the algorithm was essentially error free.

V. DISCUSSION OF THE ENDPOINT-LOCATION PROBLEM

The problem of accurately locating the endpoints of an utterance is actually a specific case of the more general problem of labeling an interval of a signal as silence, unvoiced, or voiced. If one had a perfect technique for this three-level decision, the endpoint-location problem would be trivially solved. However, such an ideal algorithm does not exist as yet. Therefore, we have looked for partial solutions to this more specific problem of isolating speech from a noisy background.

The solution to this problem was based on the premise that somewhere within the given interval there was an utterance and that it would be easy to isolate the broad region in which the speech was located using energy measures alone. From this interval, we set very conservative thresholds on the speech energy (normalized to the maximum speech energy) to get a good first guess at the endpoints of the utterance. The zero crossing rate of the waveform outside these initial estimates of the endpoint was used to provide better estimates as to the existence of unvoiced speech energy in a broad region on either side of the initial endpoints.

The question now arises as to how to make the algorithm work better. One of our key goals in the original formulation was to make the algorithm fast and efficient. To this end, the readily available parameters of short-time energy and zero crossing rate were the only ones used in the decision-making process. To increase the sophistication and thereby the accuracy of the algorithm would require the inclusion of other speech parameters, such as predictor coefficients, autocorrelation coefficients, etc. The use of such additional measurements is predicated upon knowledge of how they differ for silence and for speech. Atal[4] has suggested a reasonable pattern-recognition approach for making the distinction between the three classes of silence, unvoiced speech, or voiced speech. This method, although promising, is much slower in

running and, thus, cannot be relied upon in an on-line environment. It does, however, give good indications that the problems associated with this decision are not totally untractable.

VI. SUMMARY

We have presented a fast, efficient algorithm for locating the endpoints of an utterance in a background of noise. The algorithm is based on two measurements made on the speech: short-time energy and zero crossing rate. Although the algorithm does make small errors in finding the exact endpoints of the utterance, it was designed to minimize the number of gross errors (off by more than 50 ms) in the analysis. The algorithm has been found to be sufficiently reliable and accurate that it is currently being used in on-line experiments on word recognition.

REFERENCES

1. H. F. Silverman and N. R. Dixon, "A Parametrically Controlled Spectral Analysis System for Speech," IEEE Trans. on Acoustics, Speech, and Signal Processing, *ASSP-22*, No. 5 (October 1974), pp. 362–381.
2. R. W. Schafer and L. R. Rabiner, "Parametric Representations of Speech," Proc. IEEE Speech Recognition Symposium, Pittsburgh, April 1974.
3. B. Gold, "Word Recognition Computer Program," MIT Research Lab. of Electronics, Technical Report 452, Cambridge, June 1966.
4. B. Atal, personal communication.

Appendix C

One Pass Dynamic Programming Algorithm

.

.

.

.

Appendix D

Component Based Face Detection Algorithm

.

.

.

.

.

.

.

.

Bibliography

[1] L.R.Rabiner and M.R.Sambour. ”An algorithm for determining the endpoints of isolated utterances”, The Bell System Technical Journal, vol.54, no.2, pp.297315, Feb 1975

[2] J. Picone, ”Signal modeling techniques in speech recognition,” in Proc. IEEE, vol. 81, 1993, pp. 1215–1247

[3] J.R. DELLER, JR., J.H.L. HANSEN and J.G. PROAKIS, Discrete-Time Processing of Speech Signals (2d ed.), IEEE Press, 2000

[4] V. Ramasubramanian, Amitava Das, V. Praveen Kumar, Text-dependent Speaker-recognition using One-pass Dynamic Programming Algorithm, 2006

[5] H. Ney. The use of a one–stage dynamic programming algorithm for connected word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, 32(3):263–271, 1984

[6] B. Xie, D. Comaniciu, V. Ramesh, T. Boult, M. Simon, Component Fusion for Face Detection in the Presence of Heteroscedastic Noise , in the 25th Pattern Recognition Symposium of the German Association for Pattern Recognition (DAGM’03), September 2003, Magdeburg, Germany

[7] http://www.acoustics.hut.fi/ slemmett/dippa/chap3.html

[8] Gouvea, E.B., Stern, R.M., ”Speaker Normalization Through Formant-Based Warping of the Frequency Scale”, Proc. Eurospeech 97

[9] W. Abdulla, Cross-words reference template for DTW based speech recognition systems, IEEE TENCON 2003, Bangalore, India, pages 14-17, October 2003

[10] http://en.wikipedia.org/wiki/Biometric

[11] http://biometrics.cse.msu.edu/Presentations/AnilJain_BiometricsSecurity_Japan_Jan06.pdf

[12] D. A. Reynolds, “An overview of automatic speaker recognition technology” in Proc. of ICASSP, vol. 5, pp. 4072–4075, 2002.

[13] R. Gross and J. Shi and J. Cohn,”Quo Vadis Face Recognition”, Third Workshop on Empirical Evaluation Methods in Computer Vision, 2001