VoiceR

A System for Speaker Recognition using Gaussian Mixture Models

Degree Project in Computer Technology and Software Development, First Level
Haninge 2012
VoiceR, a System for Speaker Recognition using Gaussian Mixture Models

PETTER LÖFGREN
Abstract

Automatic Speaker Recognition (ASR) is the scientific field regarding the identification and verification of a person based on his/her speech characteristics. The usage and performance of Microsoft Kinect units as recording devices, as well as the performance of the Alize/LIA_RAL toolkit for ASR was investigated. In order to complete the task, a study of written research material regarding ASR was conducted and the VoiceR prototype system was build to implement the theoretical knowledge and toolkit functionality.

The results show that the performance of the Alize/LIA_RAL toolkit is not optimal if the recording device characteristics of the background speaker model and the audio records of the users differ, and if the verification segments are highly text-independent. The results also indicate that Kinect units are viable as recording devices and successful identifications were made, though the amount of test data was limited and the results are therefore of limited statistical significance.

This degree project report explains the theoretical background knowledge required for building ASR systems, describes the toolkits currently available for building such systems, documents the implementation of the Alize/LIA_RAL toolkit in a C# environment and presents the results of test runs made with the toolkit.

Keywords: Automatic Speaker Recognition, Gaussian Mixture Models, Kinect
Sammanfattning

Automatisk talarigenkännning är det vetenskapliga område som behandlar identifiering och verifiering av personer baserat på särdrag i deras röster. Användandet och prestandan hos Microsoft Kinect-enheter som inspelningsenheter, samt prestandan hos ramverket Alize/LIA_RAL undersöktes. För att kunna genomföra uppgiften gjordes en studie av skriftligt forskningsmaterial om automatisk talarigenkännning och prototyp-systemet VoiceR byggdes för att implementera de teoretiska kunskaperna och Alize/LIA_RALs funktioner.

Resultaten visar att Alize/LIA_RALs prestanda var låg om inspelningsmediernas egenskaper för bakgrundsmodellen och de inspelade användarnas röster skiljer sig åt och om verifieringsuttalandena är textoberoende. Resultaten visar också att Kinect-enheter fungerar som inspelningsenheter och lyckade identifieringar gjordes. Dock var mängden testdata mycket begränsad och resultaten kan därför inte statistiskt säkerställas fullt ut.

Denna rapport förklarar den teoretiska bakgrundskunskap som krävs för att bygga system för talarigenkännning, beskriver vilka ramverk som är tillgängliga för att bygga sådana system, dokumenterar hur Alize/LIA_RAL implementerades i en C# miljö och presenterar resultaten av de provkörningar som gjordes.

Nyckelord: Automatisk talarigenkännning, Gaussiska blandningsmodeller, Kinect
Acknowledgements

Many thanks to Mats Blomberg from KTH TMH for answering questions about statistical modeling and automatic speaker recognition in general and for letting me use one of their speech corpora. I would also like to thank Svante Granqvist from KTH STH for answering questions about signal processing and for putting me in contact with KTH TMH.
## Table of Contents

1. Introduction .................................................................................................................. 1  
   1.1 Background .............................................................................................................. 1  
   1.2 Goals ....................................................................................................................... 2  
   1.3 Delimitations .......................................................................................................... 2  
   1.4 Methods .................................................................................................................. 2  
   1.5 Working conditions .............................................................................................. 3  

2 Theory .......................................................................................................................... 5  
   2.1 The speech signal ................................................................................................... 5  
   2.2 Fourier analysis and transformation ....................................................................... 5  
   2.3 Band-pass filters ................................................................................................... 6  
   2.4 Spectrum ............................................................................................................... 7  
   2.5 Cepstrum ............................................................................................................... 8  
   2.6 Speaker recognition ............................................................................................... 8  
      2.6.1 System types .................................................................................................. 9  
         2.6.2 Voice Features ............................................................................................ 9  
            2.6.2.1 Short-term spectral features ................................................................. 10  
            2.6.2.2 Voice Features .................................................................................... 10  
       2.6.3 Voice modeling .............................................................................................. 11  
          2.6.3.1 Hidden Markov Model ....................................................................... 11  
          2.6.3.2 HMM for speech recognition ................................................................. 12  
          2.6.3.3 HMM for speaker recognition ............................................................... 12  
          2.6.3.4 Gaussian Mixture Model ................................................................... 13  
          2.6.3.5 GMM for speaker recognition .............................................................. 15  

3 The system ..................................................................................................................... 17  
   3.1 Purpose .................................................................................................................... 17  
   3.2 Frameworks and tools ........................................................................................... 18  
      3.2.1 SPro ............................................................................................................... 18  
      3.2.2 Alize and LIA_RAL ..................................................................................... 19  
      3.2.3 Other frameworks ....................................................................................... 21  
         3.2.3.1 BECARS ............................................................................................... 21  
         3.2.3.2 Hidden Markov Model Toolkit .............................................................. 22  
   3.3 System description .................................................................................................. 23  
      3.3.1 Client library .................................................................................................. 23  
      3.3.2 Web service .................................................................................................... 23  
         3.3.2.1 Web service operations ...................................................................... 24  
         3.3.2.2 Data contracts ...................................................................................... 25  
      3.3.3 Database ......................................................................................................... 28  
      3.3.4 Demo application ......................................................................................... 29  
   3.4 Flow design ............................................................................................................. 29  

4 Measurements .............................................................................................................. 31  
   4.1 Preconditions ........................................................................................................... 31  
   4.2 Evaluation methods ............................................................................................... 31  

5 Results .......................................................................................................................... 33  
   5.1 World model based on features with 60 coefficients ............................................. 33  
      5.1.1 Waxholm ....................................................................................................... 33  
      5.1.2 Kinect ............................................................................................................ 34  
   5.2 World model based on features with 24 coefficients ............................................ 35  
      5.2.1 Waxholm ....................................................................................................... 36  
      5.2.2 Kinect ............................................................................................................ 36
1. Introduction

Automatic speaker recognition (ASR) has been developed for computers since the early eighties and has evolved into a powerful and sophisticated technique for identifying individuals by their voices. It is important to understand the difference between automatic speaker recognition and automatic speech recognition before reading the rest of this report, since these two terms are often confused. Speech recognition deals with identifying which words were said during a recording session whereas speaker recognition is, as previously stated, the act of identifying individuals based on their voices alone. Though these techniques have some similarities they differ significantly in some cases, especially in the case of text-independent speaker recognition which is explained in section 2.6.3 and further in section 2.6.3.4.

The need for ASR is most significantly noted in forensics and telephone based secure systems. In forensics, it may be crucial to be able identify a suspect as a speaker in a recorded conversation. For secure telephone system, such as banking, ASR can be used as an additional security measure to verify that the speaker really is who he/she claims to be.

To be able to accurately verify that a voice belongs to a claimant is crucial in the kinds of applications described above. The less likely the system is to make inaccurate verifications or rejections, the higher is the performance of the system.

This report discusses the basic theory behind ASR and proposes an implementation using a Microsoft Kinect unit for recording audio data. It discusses various ways in which a human voice may be parameterized and modeled, which frameworks and tools that exist for building an ASR system and evaluates the performance of the implementation. This implementation is not a system where security is crucial, and the demands on the performance of the system are therefore somewhat reduced.

Some of the files used for speaker verification experiments originate from the Waxholm corpus which was constructed by the Royal Institute of Technology, The Department of Speech, Music and Hearing (KTH TMH). A description of the corpus is available in [1].

1.1 Background

The request for an automatic speaker recognition system came from the company Nordicstation. It was requested that an application demo program should be implemented as a result of the research done on ASR. The demo application is that of an automatic reception desk. Users of the system should be able to register themselves and their voices in the system. Once registered, users should be able to notify the system of their presence at the desk either by saying a selection of predetermined words or by saying arbitrary words.

The company also requested that the audio recording of new users were to be done via the microphone array of a Microsoft Kinect unit. The use of a Kinect unit provides some additional possibilities, such as detecting whether a human is positioned in front of the camera and the closeness of this person, which may be used to verify that someone wishes to make a registration or verification.
1.2 Goals

The primary goals for this project are divided into two parts, the first being the analytical part which constitutes the majority of this document, and the second being the implementation part, which is also documented. For the analysis, the goals were as follows:

1. Experimental research and documentation with the purpose of discovering what possibilities exist for completing the implementation when using the Microsoft Kinect unit as the main recording channel.
2. Research and documentation with the purpose of analyzing how ASR is performed.
3. A comparison between chosen existent algorithms, tools and frameworks for ASR.

The implementation constructed shall be made based on the discoveries made in the analytical part. The goals were to:

1. Construct a module that utilizes the Kinect unit for controlling the audio recording. This module should be able to verify that the audio it records originates from the active user.
2. Construct a module for analysis of the recorded voice data.
3. Construct a database module for storage of user/voice pairs.
4. Construct an automatic reception desk demo application that utilizes the three mentioned modules.

1.3 Delimitations

It was decided before the start of the project and during execution of the project that the following delimitations should be put in place:

1. The automatic reception desk application may be highly simplified, and only used for demonstration purposes. The important part is the three modules mentioned in the previous section.
2. Only one tool/framework for ASR needs to be implemented. The comparison made between tools and algorithms is therefore purely theoretical.

1.4 Methods

The methods used for completing the construction of the system were to research the currently existing scientific literature regarding ASR and a practical implementation of a prototype system based on the facts discovered during the research.

The following frameworks, programs and libraries were used to build the prototype:

- The .NET Framework 4 was used for the general VoiceR architecture.
- Entity Framework was used for O/R mapping in the data access layer.
- A RESTful Windows Communication Foundation service was set up to handle client to server communication.
- Microsoft SQL Express (SQL Server 9.0.5000) was the database management system (DBMS) used to manage the database.
• Microsoft Kinect SDK 1.0 was used to interface with the Kinect unit used for recording audio.
• SPro 4.0 was used to extract feature vector files from the audio files containing voices.
• Alize 2.0 is the mathematical kernel used to perform general statistical calculations.
• LIA_RAL 2.0 is the ASR toolkit which use Alize to perform algorithms for speaker recognition.
• Cygwin 2.774 was used to build the C/C++ based programs (SPro and LIA_RAL) to Windows executables.
• MATLAB 7.12.0 was used together with a set of MATLAB script files made by NIST in order to plot the diagrams found in chapter 5.

1.5 Working conditions

This section presents the conditions under which this project was undertaken. It describes some of the problems regarding the execution of the task and the impact of these on the final result.

The author had no previous knowledge of signal processing, voice modeling or acoustics in general when the degree project began. Because of this, a lot of research needed to be done to understand the basic concepts, math and models which are discussed in the research literature. This has impacted the analytical part of this report in the sense that these fundamentals are discussed in addition to the topics that are directly relevant for the execution of the task.

All documentation for the use of Microsoft Kinect unit is written in compliance with the .NET framework and it was therefore decided that the source code for the program was to be written in C#. This posed a problem when implementing the prototype because most ASR programming libraries are written in C or C++, and interfacing C# with already written C/C++ source code is not a trivial matter.

Another problem was the lack of a scientifically composed database containing recorded audio of many different human voices. Most of the existing databases are only available commercially, with costs in the range of hundreds to thousands Euro. The voice audio data used for constructing the universal background model (see section 2.6.3.5 for explanation) was gathered from a vast set of recordings of customers giving feedback to various Swedish companies. Therefore, some of the recordings are made in noisy environments and the gender of each speaker is not known in advance. Further, the sessions vary from tens of seconds to several minutes and each session is assumed to be made by a unique speaker. The Waxholm speech corpus from KTH TMH, which contains 68 speakers, was used in order to test the system performance.
2 Theory

This chapter presents the basic theory behind voice processing and modeling. It discusses some purely mathematical concepts such as Fourier analysis, but also filtering, spectrum and cepstrum analysis. These areas will only be covered briefly, and more detailed explanations of feature extraction and the statistical voice modeling is explained thereafter.

2.1 The speech signal

Before discussing the mathematical theory of voice and signal processing, a short description of how the human voice system works is given. Speech is generated by the movement of air from the lungs through the throat where the vocal cords are more or less tensed. The vocal cords allows for small portions of air to pass through which in produces resonances in the oral cavity [2]. The wave of air is then changed further in the oral cavity by the tongue and teeth. The resulting air wave can be perceived as speech by the human ear.

This movement of air can be sampled to binary format using an A/D converter. The resulting values describe the waveform of a sound signal. It is these values that are subject for further processing when extracting features from a voice.

2.2 Fourier analysis and transformation

Fourier analysis is the mathematical process of either decomposing a periodic function into its basic parts, or constructing a periodic function from its known parts. In the application domain of signal processing (and thus speaker recognition), Fourier analysis may be utilized in order to extract the frequencies that a voice signal is composed of.

For this degree project, it is the task of decomposing a signal into its fundamental sine waves that is of importance since this makes it possible to extract the features of a voice. The decomposing task is called Fourier transformation, and when applied on finite signals that can be processed by a computer, it is called Discrete Fourier transformation (DFT).

Consider for example the signal portrayed in Fig. 1. The sampled signal is portrayed with its amplitude along the vertical axis, and the horizontal axis represents the time at each sample. While this information does not provide data about the frequency of the signal, it can still be discerned that it is one single simple periodic signal. Since it is not composed of multiple waves (and thus multiple frequencies), a DFT should only generate one distinct frequency.

Mathematically, this signal may be represented as a simple harmonic motion:

\[ u(t) = A \cos(\omega t) + B \sin(\omega t) \]  

Fig. 1: A simple waveform signal

\[ 1 \]
Further, utilizing Euler's formula, equation (1) may be written as a complex function:

$$u(t) = ae^{i\omega t} + be^{-i\omega t}$$  \hspace{1cm} (2)

The previously mentioned signal is simple. A more complex signal, one that is composed by several simple signals can be mathematically described as the sum of the simple signals [3]:

$$u(t) = \sum_{k=1}^{n} (a_k e^{i\omega_k t} + b_k e^{-i\omega_k t})$$  \hspace{1cm} (3)

If these signals are considered to be periodic, with the fundamental frequency $\Omega = 2\pi/T$ the formula in (3) may be generalized to [3]:

$$u(t) = \sum_{n=-N}^{N} \hat{u}_n e^{i n \Omega t}$$  \hspace{1cm} (4)

The purpose of Fourier analysis is to find the value of the $\hat{u}_n$ coefficients for each simple wave that composes the signal. The actual process of doing this will not be described here because of space limitations and since it is a common mathematical process in the signal processing field.

### 2.3 Band-pass filters

A band-pass filter is a filter used in signal processing in order to reduce the amplitude of a signal in two or more frequency bands [4]. These filters are usually constructed to increasingly allow more of the signal to pass in a certain frequency range. This is either done in a nonlinear or linear fashion. When constructed in a linear fashion, band-pass filters may be referred to as triangular filters. Fig. 2 shows the magnitude versus frequency of a signal after it has passed through a band-pass filter made in a nonlinear fashion. As can be seen, the amplitude of the signal has been dampened in the lower and higher cutoff frequencies $f_0$ and $f_1$. This reduces the impact of the frequencies below $f_0$ and above $f_1$ on the signal as a whole.

![Fig. 2: The magnitude of a signal after it has passed through a band-pass filter with cutoff frequencies at $f_0$ and $f_1$](image)

Triangular filters work in the same way, except for the fact that the signal is allowed to pass through linearly.
2.4 Spectrum

A spectrum represents the frequencies extracted by a Fourier transformation made on an audio signal in a single window. A window is representative of a small time interval, usually 10 to 30 ms in which the audio signal is assumed to be stationary [5]. A single audio fragment in a window is denoted as $x[n]$ where $n$ is the index of the current window. This audio segment is called a frame. In order to visualize the frequencies of the audio signal as a whole, each frequency extracted in each window is usually given a grayscale color which represents the amplitude [6]. Fig. 3 shows a speech audio signal in waveform format in the upper picture and its spectral representation (spectrogram) in the lower. In the spectrogram below, the vertical axis represents frequency (in kHz) and the horizontal axis represents time.

A phone is the smallest component that makes up a flow of speech [7], and each phone is represented by a formant, which in turn is the peak (dark area) in the spectrum. The dark areas in the spectrogram indicate that those frequencies are of higher amplitude, and are therefore more significant for the spoken phone in that frame. It is possible to analyze which formant is related to which phone by looking at the energy (intensity of the darkness in the spectrogram) of the frequencies in the current window. For example, phones made with tongue and teeth such as [t, s] generally contain high energies throughout the whole frequency band of human speech, and especially high energies in the higher frequencies. In fact, the formant after the first period of silence in Fig. 3 is a [t].

With this in mind, it should be possible to analyze what spoken words are uttered in an audio segment by watching the transitions between formants, since the formants represent phones and phones are components of words. It is possible to statistically model this by using Hidden Markov Models (see section 2.6.3.1 for further details).

The spectrogram in Fig. 3 is visualized in the time domain, which simply means that time is represented on the x-axis. The signal can be closer examined by looking at the amplitude of the frequencies for each individual window. One such window is shown in Fig. 4.
The peaks shown in Fig. 4, are equivalent to the dark areas in the spectrogram in Fig. 3, and are therefore formants. Since it is the formants which contain the information of the spoken phone, the data in a frame can be abbreviated by connecting the peaks in a smooth curve, called the spectral envelope. The formants may then be easily obtained by finding the local maxima and minima of the spectral envelope. Obtaining the spectral envelope requires some additional processing of the spectrum as explained in section 2.5. Further, a spectrum (in a single frame) can be thought of as a combination of the spectral envelope and the excitation signal which represents the air stream that appears when speaking.

2.5 Cepstrum

A cepstrum can be thought of as a spectrum of a spectrum. It is mathematically described as the inverse DFT of the logarithm of the magnitude of the DFT of a signal [8]. Algorithmically, for each frame $x[n]$ of the signal, this can be expressed as:

1. Perform DFT on $x[n]$(calculate the spectrum).
2. Compute the logarithm of the result received from the first step.
3. Perform the inverted DFT on the result received from the second step.

The reason for using the cepstrum instead of the regular spectrum of a voice signal is that the cepstrum makes it possible to separate the spectral envelope and the excitation signal from the spectrum. The inverted DFT on the spectrum will extract two main frequencies, one for the spectral envelope (low frequent) and one for the excitation signal (high frequent) [6]. By filtering out the high frequencies from this cepstrum a vector of cepstrum coefficients which represents the spectral envelope is attained. Transforming the spectrum into a cepstrum also discards the phase of the signal, which has been found to have little importance for speaker identification [5].

2.6 Speaker recognition

Identifying a person by his/her voice is a task that requires several quite complex processing steps, where each step is dependent on the next. The steps and decisions needed for constructing an ASR system is presented in this section, starting with a presentation of two of types of systems, continuing with describing voice features and ends with discussing two different ways of statistically modeling a person's voice.
2.6.1 System types

In general, there are two types of speaker recognition systems: text-dependent and text-independent. The users does not need to be cooperative in text-independent systems. This means that they might not be aware that their voices are being recorded and analyzed. Further, this also means that no data apart from the voice signal is available to the analyzing system, and it may therefore be harder to analyze the signal. This kind of system is commonly used in forensics [5].

Text-dependent systems assumes that the users are cooperative (in the sense that users are fully aware that their voices are being recorded and processed) and that they read predetermined texts when registering or verifying themselves [4]. However, it is not necessarily required that exactly the same utterances are spoken in the registration and verification phase. The way in which registration and verification utterances are assumed to be spoken can be divided into several classes amongst some are:

- **Vocabulary driven systems**: The words used in registration are a set of words existing in a vocabulary kept with the system. During verification, a subset of the words in the vocabulary is used. The system requires these words to be spoken.

- **Text-independent, system driven systems**: The system presents a set of words that the user should use, but in difference to the previously mentioned class it merely expects these words to be spoken, without forcing the user to actually speak them.

For the system being constructed in this project either of these system classes may be used, depending on the requirements of user friendliness combined with system accuracy. The former class would provide higher accuracy but lower user friendliness, and vice verse for the latter approach, as explained below.

Rigid text-dependence (as in vocabulary driven systems) simplifies the task of determining whether the spoken sequence audio quality is good enough. By using speech recognition on each word uttered during registration, a certain amount of confidence may be attributed for each word spoken[9], which may act as a qualifier for the incoming sound. The loss of user friendliness may in this case be that users are having trouble pronouncing specific words in a correct way, and thus being unable to perform registration or verification.

2.6.2 Voice Features

Voice features can be thought of as the characteristics of a person's voice. These characteristics can be divided into several categories, ranging from the pure physiological to word usage and mannerisms. In [5] the voice features are divided into five categories: Short-term spectral features, voice source features, prosodic features, spectro-temporal features and high-level features.

- The short-term spectral features of a voice is identified as the timbre of the voice sound and its resonance properties of the throat and oral/nasal cavities. This data is the spectral envelope which is mentioned in section 2.5.

- The voice source features represents the physiological properties of the organs generating human speech, such as the slack of the vocal cords and the shape of the glottis.

- Prosodic and spectral features are for example the speech rhythm, intonation and syllable stress of the words spoken by a person.

- High-level features represent the mannerisms of a person's speech, such as commonly recurring words, accent or language.
2.6.2.1 Short-term spectral features

Spectral features are extracted from the raw voice signal data. In order to process the features of the voice, the signal data is divided into frames of 20 to 30 ms. The frames that results from such a division of the voice signal can be expressed as \( x[n;m] \), \( n = m - N_{sl} + 1, m - N_{sl} + 2, \ldots, m \). In this expression, \( N_{sl} \) is the number of samples for each frame with \( m \) being the last sample for that frame.

Once the data has been divided it can be transformed using a discrete Fourier transformation which extracts the spectrum of the voice. This spectrum can then be further processed, and features can be extracted.

Examples of such features are mel-frequency cepstral coefficients (MFCCs) and linear predictive cepstral coefficients (LPCCs).

MFCCs are constructed by passing the speech signal through a set of band-pass filters spaced along the mel-frequency scale (Fig. 5), computing the log-energy for each filter (which includes performing DFT on the \( x[n;m] \) frames) and the computing the discrete cosine transformations of these energies \([10]\). The reasoning behind MFFCs as being qualified candidates for representation of the voice is that they closely model the frequency resolution of the inner ear.

The computation of LPCCs does not involve DFT, rather it is based on the coefficients that constitutes the all pole model \([11]\) called the Linear Prediction Coefficients (LPCs). In order to understand LPCCs one must first understand LPC which are explained briefly below.

LPCs are defined as the future estimation of samples that constitutes the input signal \([11]\). This can be shown mathematically as:

\[
\hat{x}_n = -\sum_{i=1}^{p} a_i x_{n-i}
\]

where \( \hat{x}_n \) is the estimate and \( a_i \) is the line coefficient. The main task in computing the LPC is to minimize the error between the real sampled data and the estimate. LPCCs can be derived from the LPCs as shown by Huang et.al \([12],[8]:

\[
c_n = \begin{cases} 
\ln(G) & n = 0 \\
\frac{1}{n} \sum_{k=1}^{n-1} k c_k a_{n-k} & 1 \leq n \leq p 
\end{cases}
\]

where \( a_n \) is the line coefficient of the LPC.
The other voice features will not be further described, since the short-term spectral features are best fitted when starting to build ASR systems [5].

2.6.3 Voice modeling

In modern ASR systems, human voices are most commonly modeled using statistical models. The two most common models are Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs), where HMMs may be used for text-dependent systems and GMMs are used for text-independent systems.

2.6.3.1 Hidden Markov Model

HMMs have been used extensively in the field of speech (word) recognition, and as seen in [13], a HMM based speech recognition system can be adapted to be used for speaker recognition. In such a case, the speaker recognition system needs to be text-dependent. Following is an explanation of HMM when applied to speech recognition, and following that will be an explanation of how this is used for speaker identification.

A HMM can be thought of as a state machine of which the states are hidden from view and the only way to determine its current state is to examine the output that is being generated. To further elaborate on this, Markov chains needs to be explained.

A Markov chain is a set of states which changes randomly over time. The probability for a change from one state to the other is independent of previous states before the current one. That is, the transition from state B to C is not dependent on whether state A or C was the previous state. This can be seen in Fig. 6, which illustrates a Markov chain with three states. A fully visible Markov chain such as the one in Fig. 6 is the simplest form of Markov model since all the states are known (visible) and the probabilities for each transition is known [14].

![Fig. 6: Three-state Markov chain. Each node is a state and the edges represent the transition probabilities. Transitioning from the current state B is not dependent on previous transitions.](image)

For a Hidden Markov Model, the states are not known in advance. However, as the state changes, an output is generated, which in turn is visible. In the case of speech recognition, each state would be a spoken phone, and the observations are the extracted speech vectors from the utterance [15] and the HMM itself represents a whole word.
2.6.3.2 HMM for speech recognition

If the observed speech vectors (observations) are denoted as a sequence $O$, the problem of finding the correct word that the utterance represents can be expressed mathematically as [16]:

$$\text{argmax}_i \{ P(w_i | O) \}$$

(7)

where $w_i$ is the $i$th word registered in the system. The formula can be read as “Find the index of the word which is most likely to have been spoken given the observed feature vectors.” As stated in [16], it is not feasible to directly compute $i$ due to the dimensionality $O$. This is instead done by using a HMM. Utilizing Bayes' theorem, $P(w_i | O)$ can be calculated by knowing $P(O | w_i)$ and assuming that $P(O | w_i) = P(O | M_i)$ where $M_i$ is the HMM representing $w_i$, the probability that a certain word was spoken can be calculated.

Assuming that the sequence $X$ in which the model changes states are known, the joint probability that $O$ originated from a Markov model that transitioned through the state sequence $X$ can be calculated. This is done by multiplying the probability for each state transition with the probability of each observation seen as a result of the transition, for each transition. This is expressed as

$$P(O \cap (X | M)) = \prod_{t=1}^{T} a_{x(t)x(t+1)} b_{x(t)}(o_t)$$

(8)

where $a_{x(t)x(t+1)}$ is the probability for a transition from state $i$ to $j$ and $b_{x(t)}(o_t)$ is the probability that a certain observation is seen at transition time (the index of the transition) $t$ and state $j$, and $T$ is the index of the final transition (to the exit state).

For a Hidden Markov Model, this calculation is not directly possible, because there is no beforehand knowledge of the state sequence, which means that $X$ is unknown. In [16] this is solved by the following statement:

"Given that $X$ is unknown, the required likelihood is computed by summing over all possible state sequences $X = x(1), x(2), x(3), ..., x(T)$, that is $P(O | M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} a_{x(t)x(t+1)} b_{x(t)}(o_t)"$

However, it should be noted that the probabilities for the state transitions ($a_{ij}$) need to be known for each possible state. HMMs may be iteratively trained for this purpose, with the starter transition probabilities set to be evenly distributed for each state. As the model is trained, the state transition probabilities are adjusted to fit the spoken word model [16].

2.6.3.3 HMM for speaker recognition

For a speaker recognition system based on HMMs, each word spoken by the user during enrollment is stored as a HMM representing that word as spoken by the user. When an unknown user wishes to be identified, each word spoken during the session is matched against all the HMMs (representing words) for each speaker. If the word itself is found in a set of word HMMs for a specific speaker, the probability of the spoken word is retained. If no word was recognized for the current speaker who wishes to be identified, the probability score is discarded.

This process repeats for all the speakers in the system, and the probability score for each speaker is stored (if the word was recognized). When all the speakers' HMMs have been processed, the speaker that corresponds to the highest probability score is considered to be the correctly identified speaker. The algorithm is shown in Fig. 7.
That a speaker is identified this way does not necessarily mean that he/she is the correct speaker, since the algorithm has merely selected one of the speakers in the list. An unregistered user (an impostor) may have similar voice features as one of the registered users (a client), and without any further control, the impostor would be allowed into the system.

To solve this problem, a threshold is set for each client. If the probability attained from the previously described algorithm is higher than this threshold, the unknown speaker is verified. This threshold can’t be set globally for all the clients; rather it must be calculated for each one. This may be done by letting the registered user perform a number of verifications directly after registration, and storing the mean and standard deviation of the generated probability scores. Then compare the HMMs of the utterances spoken during the verification, perform the algorithm shown in Fig. 7 and store the mean and variances for these as well. The threshold is calculated as [13]:

\[ T = \frac{\mu_u \sigma_c + \mu_c \sigma_u}{2 \sigma_c} \]  

Where \( \mu_u \) and \( \sigma_u \) are the mean and standard deviation of the unknown speakers probability scores respectively, and \( \mu_c \) and \( \sigma_c \) are the corresponding values for the collection of probability scores generated. This value is also known as the equal error rate (EER), where the chance of wrongly accepting an impostor and wrongly rejecting a client are equal.

2.6.3.4 Gaussian Mixture Model

In difference to speaker recognition using HMMs, a system based on GMMs utilizes the speaker feature vectors directly. Feature vectors can be thought as collections of the coefficients extracted from the mathematical transformation of the voice signals. For instance, a feature vector can contain the coefficients that make up the MFCC representation of the signal. In order to explain GMMs, some further elaboration on feature vectors is required.
Feature vectors may be visualized as a point in a space of \( n \) dimensions, where each element in the vector represents a "coordinate" for that point. The whole feature vector would then represent a single point in the \( n \)-dimensional room. Since these points represent speaker characteristics, it would be possible to compare speakers to each other by calculating the distance between these points, and a small distance between two points would indicate that the speakers are similar. To take this reasoning one step further, it would also be possible to represent the feature vectors extracted from each frame of the mathematically converted audio signal as such points.

When comparing speakers to each other one possible solution (assuming that the feature vectors extracted during registration are stored) is to compare the feature vectors extracted from the verification speech signal. This is done by computing the distance between the points from the verifier and the registered points. As imaginable, this would lead to a lot of computations for each verification if a lot of speakers are registered in the system. Therefore, points that reside closely to each other may be clustered together by a method such as k-means clustering [5] and each cluster would represent a single speaker registered in the system. This method of compressing vector data is called vector quantization (VQ). Feature vectors from unknown speakers only need to be compared to the clusters, and not every other feature vector stored.

GMMs are similar to VQ, but instead of assigning each feature vector to a speaker it is given a statistical chance of belonging to a speaker [5]. The idea of GMMs can also be derived from the Gaussian distribution function (also called normal distribution). Simply put, a mixture of probability functions can be thought of as the combination of several probability functions, as seen in Fig. 8. Though the figure only depicts a mixture in one dimension, it is possible to extend the idea to multiple dimensions [17].

![Fig. 8: An example of the most simplistic version of a Gaussian Mixture Model in one dimension. The black line represents the GMM, and the others represent its components.](image)

To further illustrate the concept of GMMs in the context of speaker recognition, Fig. 8 can be thought of as being composed of feature vectors which contain one single element that “points to a coordinate in a one dimensional room”; that is an arbitrary real decimal value at the x-axis in Fig. 8. Each of the components (Gauss bells) indicate that the values are clustered at these locations and since these feature vector values represents the characteristics of a speaker, the GMM as a whole is representative of a speaker voice in a statistical sense. In practice, each feature vector contains 24 – 40 elements which means that the dimensionality of the GMM is of equal size.
Training GMMs to represent human voices requires a lot of voice data for each speaker. One significant problem with the GMM approach is that each model requires huge amounts of voice audio data for each speaker to be obtained, usually in the range of tens to hundreds of hours [5]. The training is performed by an algorithm known as Expectation Maximization (EM). This algorithm takes the feature vector coefficients from a spoken audio segment as input and outputs the optimized parameters for each component of the GMM. The algorithm is iterative, and in each new iteration the training parameters are refined to fit the feature vector coefficients more accurately. This poses the problem of over training, which means that the parameters are so tightly trained to a specific speaker that even slight variances in the voice would result in a rejection of the true speaker at verification. It is therefore important to choose an appropriate number of iterations, and according to [5], only a small number of EM iterations are needed.

Training a GMM from scratch requires (as previously mentioned) a lot of speech audio data to be collected. Having this amount of data is not practically feasible for most applications and the solution to the problem is to train a background model. A background model is a GMM that contains voice characteristics from a large amount of different speakers. The background model may then be adapted for each new speaker to become more representative for that speaker. This adaptation may be done by the maximum a posteriori (MAP) algorithm [5].

### 2.6.3.5 GMM for speaker recognition

When testing whether a feature vector belongs to a speaker voice model, a similarity score is calculated based on the observed feature vector, the background model and the speaker voice model. The equation for the probability that a speaker model belongs to the observed feature vector is

\[
\log(P(m|O)) = \log(P(O|m)) - \log(P(O|m_w))
\]

where \(O\) is the observed feature vector, \(m\) is a speaker model and \(m_w\) is the background model [18].

This score is called the log-likelihood ratio (LLR) score, and is dependent on comparisons between the speaker model and the background model. An LLR score below 0 indicates a rejection and a score above 0 indicates a possible match. These calculations are further detailed in [18] and are not explained further here.

The rejection or acceptance of a certain speaker is based on a fixed threshold that \(\log(P(m|O))\) should exceed in order to decisively determine that a speaker model belongs to the observed feature vector. However, this computation may be suboptimal if the functions for determining the probabilities in the right hand side of (10) are not fully known [19]. This calls for the need of the score to be normalized.

There exist a few different kinds of score normalization of which two will be explained here: Z-normalization and T-normalization. For Z-normalization a set of feature vectors spoken by impostors are used to normalize the LLR score. The Z-normalized score is calculated by the LLR scores of the impostor feature vectors based on the speaker voice model (and the background model used to generate that model) [20]. An advantage with this technique is that it can be used when the voice model is created by storing the impostor LLR scores.
T-normalization is based on the voice models of a set of impostor speakers. When a user wishes to be verified, the feature vector of his/her speech is compared against all the impostor speaker models. This method is considerably slower than Z-normalization, and it needs to be computed each time a user wishes to be verified. A combination of Z-normalization when registering and T-normalization when verifying has shown to produce good results when the channel and utterances used by the impostors does not vary to much from the ones used when registering and verifying [5].
3 The system

The implementation of the system and the research necessary for the direct implementation is presented in this section of the report. The system process is divided into a set of sub tasks that describe a stage of the system workflow. These sub tasks are:

- Audio recording
- Feature extraction
- Voice modeling
- Database definition and storage
- Service architecture
- Speaker verification

The audio recording section deals with how data should be recorded with regard to recording device, sample rate and quality, and file format. The recording device is a Microsoft Kinect unit as specified by the requirements (see section 1.2). Since the target OS platform is Windows, using the Microsoft WAVE file format for recorded audio files is a natural choice.

The purpose of the feature extraction section is to extract data from the recorded audio files and parametrize it into representational feature vectors (containing for example MFCCs or LPCCs). Identifying which frameworks are freely accessible and their functionality regarding this matter is important, as well as how their outputs relate to the rest of the system process. Additional processing of the audio files or feature vector coefficients must also be considered.

Voice modeling deals with how the voice feature vectors are used to form statistical models from the feature vectors.

In order to store user identities, voice models and other necessary components for automatic speaker recognition, a database schema needs to be defined. The database should be able to associate the speaker and user data as well as what feature coefficients were used.

A general service and system architecture must be defined to describe exactly how the system is constructed and how its different parts operates with each other. Matters such as web service data contract definitions and class relationships are described. Finally the method of speaker verification and threshold for acceptance must be determined.

3.1 Purpose

The purpose of the implementation is to demonstrate how a system for automatic speaker recognition may be developed given the criteria mentioned in section 1.3. The general performance of the system should also be obtained after implementing all the processing steps. These results can be found in chapter 4 and 5.

A demo application using the implemented functionality is also constructed to demonstrate the practical application of an ASR system in the form of an automatic reception desk.
3.2 Frameworks and tools

A research of some frameworks and tools for signal processing and voice modeling that currently may be used for building an ASR system is described in the following sections. Tools for feature extraction are discussed and then the mathematical framework Alize and its corresponding ASR library is explained and compared to other tools that currently exist. The comparison is mainly theoretical because of time limitations regarding the implementation of several tools and frameworks in the system.

3.2.1 SPro

SPro is an open source toolkit for analyzing audio data. It provides mathematical functions (such as fast Fourier transformation and linear prediction) for extracting audio features from raw audio data files. Readable audio file formats include raw PCM, WAVE files and SPHERE files, and the output feature files may contain MFCCs or LPCCs amongst others. The library does not contain any functionality for feature extraction on the levels above short-time spectral features which are explained in section 2.6.2.1.

The SPro toolkit can be compiled and built to a set of executable files. Each of these files provides various functionalities and is configurable with a set of options which will be briefly described below:

- *sfbank*: This program allows filter bank analysis to be performed on the audio signal by passing the received audio data through a set of filters (see section 2.3 for further explanation of filters) and computing the log-energy output of each filter bank.
- *sfbcep*: Similar to the sfbank program this program also utilizes filter banks to transform the initial audio data. It extracts the cepstral coefficients from the audio data by computing the discrete cosine transform for each log-energy output of each filter.
- *slpcep*: The slpcep program allows linear prediction to be made on the cepstral coefficients.

Four options are of significant importance for the first two programs described above: The -m option which spaces the filter bank filters along the mel-frequency scale instead of linearly. In the case of sfbcep, using this option produces MFCCs. The default option without the -m flag defines that the filters are spaced evenly.

The -p option controls how many filters are set (and thus the number of output coefficients) which is important for the next section of the report, since it controls the size of the feature vectors used when modeling a voice. The default value is 12 even though the literature on the subject recommends 24 – 40 coefficients [10].

The -D and -A options allows for the first and second derivatives of the coefficients to be calculated and added to the output feature vector.

In order to process the raw audio data information, the audio signal must be stationary. This is simulated by framing the audio signal in such a way that the audio data in each frame can be assumed to be stationary. SPro performs this by using a framing window with a width of 20 to 30 ms [21].

The program source code can either be directly interfaced by including the necessary library files, or executed as a separate process. For the development of the prototype system, the second approach was used because of the C++/C# problem described in section 1.5.
3.2.2 Alize and LIA_RAL

Alize is an open source library developed at the university of Avignon in France. It contains low-level mathematical and statistical functions used to analyze and process audio data. Alize is therefore generalized and not specifically targeted to speaker recognition. Because of this, the developers of Alize have also constructed a secondary library called LIA_RAL, which does have functions for speaker recognition.

The library described in this section will be LIA_RAL, since it contains all the functions for speaker recognition and thus abstracting the Alize functions from the application programmer. LIA_RAL is built around the idea of modular applications with each application performing one designated task. In general, there are two ways of working with LIA_RAL. The application programmer can either interface to the C++ source code directly or each application module can be built separately and be run with semi-dynamic configuration with some parameters stored in configuration files and some are provided at run time as command line arguments. The latter method is denoted as the pipeline method, and it is this method that is used for the development of the Alize/LIA_RAL part of the project.

As previously mentioned, LIA_RAL consists of several models, but not every one of these are used. The library does not contain any module for parameterizing raw audio data to mathematical coefficients (such as LPCCs or MFCCs, see section 2.6.2.1), but an external program must be used for this. The program that the developers of LIA_RAL used during testing of the system was SPro, which is detailed in section 3.2.1. It was discovered during the Alize/LIA_RAL development phase that WAVE files sampled in 8 kHz and 16bit (128 kbps) worked well with the rest of the pipeline. This is probably because most speaker recognition systems are used over the telephone network which carries audio signals sampled in 8 kHz [22].

When constructing a system using the pipeline method, the whole pipeline process needs to be executed several times (phases), with some processes being switched for others. These different phases are world training, target training and verification. World training involves either creating two background models (see section 2.6.3.4) for each gender, or one single model with mixed genders. The model(s) are used as a base for the voice model for each new registered speaker. In order to do this, huge amounts of audio data from unique male and female speakers are required (tens or hundreds of hours [5]). Obtaining this data was a major difficulty, since most databases are only available commercially and can be quite expensive.

The world training phase may be executed once per system installment, or once globally if the world models are stored in a centralized database and each sub system downloads the model when initializing. The train target phase is run each time a new speaker registers in the system, and an individual speaker model is created by adapting the background model. The verification phase is executed when an unknown speaker wishes to be identified and the feature vectors extracted from the speaker is compared to each of the target models stored in the system and a probability score is calculated for each target model. Fig. 9 shows the general flow of the pipeline process, with the start set to when the audio signals are received by the SPro program.
It is important to notice that the Alize/LIA_RAL framework is based on the generating of files; it does not store the data in these files in a database. The database part of the above diagram is an application specific design used for this project. Therefore, every mention of a database in the section below should be read with that fact in mind. When using Alize/LIA_RAL without any database, the files are stored and read directly on the client hard drive.

Following is a list of the processes used in the Alize/LIA_RAL part of the development in the order that they are used in the pipeline:

1. **SPro**: Although not part of the Alize/LIA_RAL framework, the SPro programs are used to extract the mathematical features of the voice audio signal. It is configured to take a WAV file as input and outputs a file containing the mel-frequency cepstral coefficients for each frame.

2. **EnergyDetector**: The EnergyDetector program's main task is to remove silence from the .prm file. This is done by measuring the energy (amplitude) of each of the coefficients in the received file. It is important to note that this program does not modify the original file, but rather it produces an additional file called the label file (.lbl). This file indicates where the voiced frames of the .prm exist and must be used throughout the rest of the pipeline.

3. **NormFeat**: This program normalizes the voiced frames of the .prm file so that background noise is filtered out. It takes both the .prm and .lbl file as input and produces a .norm.prm file as output.

4. **TrainWorld**: This program is only run once. It requires special attendance because it needs a lot of voice feature data. Instead of receiving each feature file separately, it takes a list of feature files to base the world model on. Because of this, the three previous states will need to have been run several times before, one time each for the audio files that constitutes the world speakers. Once the program is done processing the feature files, a Gaussian mixture model based on the feature files is generated and stored in the database.

---

**Fig. 9**: The SPro and Alize/LIA_RAL program pipeline as implemented in the system prototype.
5. **TrainTarget**: In difference to TrainWorld, this program is run every time that a user wishes to register in the system. It fetches the world model from the database and adapts the target model to fit the speaker by using his/her normalized feature vectors. The target model is then stored in the database.

6. **ComputeTest**: This program and the branch that follows, is executed when an unknown speaker wishes to be identified. It fetches all the target GMM files from the database and iteratively compares the feature vector files received from the previous step with each target file. A score and a decision of whether the received feature vectors actually belong to a specific target model is written to a file on the hard drive.

7. **ComputeNorm**: Even though ComputeTest outputs a decision, it may be inaccurate and further normalization is required. This is done by a method called Z-norm. The Z-norm methods requires that two or more feature files from impostors are compared to the target GMM files [17] and the score is normalized based on the scores generated.

### 3.2.3 Other frameworks

This section discusses two other ASR frameworks: BECARS, which is a framework developed in Lebanon for speaker verification using GMMs and HTK, which was developed by the Speech, Vision and Robotics group of the Cambridge University Engineering Department. This tool provides functionality for building HMMs, which in turn can be used for text-dependent speaker recognition. A brief explanation of both the frameworks will be given first, and their functionality and use for ASR will be discussed after that.

#### 3.2.3.1 BECARS

BECARS is made up of a set of modules which may be integrated by application programmers directly using the C source code that makes up the BECARS system, or by invoking processes to start up each pre-built component with a set of program line arguments. The BECARS framework are also dependent on a set of shell script files to be run in order to provide the modules with additional argument information such as the names of the files to be processed [23]. Four modules makes up the BECARS system: gmm_trn, gmm_adp_tree, gmm_tst, gmm_llk. The modules functionality are described below as they should be executed in chronological order if building a system in a pipeline fashion as explained in the previous section:

1. **gmm_trn**: This module takes a set of feature vectors and trains a GMM based on them, based on a maximum likelihood criterion (see end of section 2.6.3.4 for further explanation). The program may be used to train an initial world model for all the background speakers.

2. **gmm_adp_tree**: This module is used to further train the GMM generated from the previous step, and allows for MAP algorithms to be run on the model. It is also possible to adapt the background speaker generated in the previous step by specifying the feature vector location in a provided script file.

3. **gmm_llk**: This module takes a previously generated GMM and compares it to a set of feature vectors. It produces a text with the log-likelihood ratio for each frame of the feature vector file.

4. **gmm_tst**: This module calculates a score and outputs a decision of whether the provided feature vectors belongs to any of the previously created speaker GMMs.
The BECARS framework takes a minimalistic approach to ASR, providing a limited amount of modules and parameters. Further, the use of user made script files in order to configure the models are not a particularly user friendly approach. Since this feature is poorly documented, it may limit the practical use of the framework. However, the framework does provide sufficient functionality for GMM based ASR, provided that it is supplied with feature vector files. BECARS support SPro files which means that the feature vector files can be generated using SPro as explained above.

One obvious difference from the Alize/LIA_RAL system is that BECARS assumes the feature vector files to be of high quality, while Alize/LIA_RAL offers functionality to filter out non-speech frames and to reduce the noise of the features by normalizing them. One possible solution to this problem is to run the NormFeat modules from Alize/LIA_RAL and then run the gmm_trn or gmm_adp_tree BECARS modules. Seeing as NormFeat also outputs a file of SPro format, this could serve to filter out noise.

3.2.3.2 Hidden Markov Model Toolkit

The Hidden Markov Model Toolkit (HTK) is a framework containing functionality for building general purpose HMMs. This toolkit provides tools for signal processing, labeling, general statistical and mathematical functions and HMM definitions [16]. Since the toolkit is totally based on HMMs, using it for speaker analysis is preferably done in a text-dependent way (as explained in section 2.6.3.3). This makes it vastly different from the Alize/LIA_RAL system which does not support text-dependent speaker recognition.

HTK uses a set of modules for the previously mentioned features. The modules will be briefly described but any comparison to Alice/LIA_RAL will be omitted since the toolkits are so dissimilar. HTK Tools are broken down into four categories: The preparation tools, the training tools, the recognition tools and the analytical tools [16]. These categories are briefly described below:

The most notable of the preparation tools are the HSLab and HCopy which deals with recording and parameterization of the recorded files audio respectively. These tools allow for data to either be processed directly from a database (which is the most common case) or the user may collect the data by hand.

The training tools are used for training HMMs and allow the user to define HMM prototypes that can be trained to model formant transitions. The model constructed by defining the basic characteristics and the initial state transition probabilities for each state and then training the model on sets of speech data. This is either done with each audio file being labeled to indicate where phones are located, or by reading the audio data directly with preset values for the mean and variance of each HMM to be trained.

HTK's recognition tool HVite is used for word recognition. It requires the allowed word sequences to be defined, in addition to a dictionary which states how each word should be pronounced. It then converts the word sequences to phone sequences and attaches the previously defined HMM prototypes to each phone, which allows for HMM based speech recognition [16]. Since each HMM prototype is available to the application developer, constructing a HMM based speaker system should be possible by using the method mentioned in section 2.6.3.2.

The analytical tool HResult makes it possible to evaluate the performance of a speech recognition system. HResult can analyze the performance on a speaker basis, which may make it suitable for evaluating a HMM based speaker verification system.
3.3 System description

The developed system is divided into three parts, a client library that processes audio files, a database that stores speaker and user information and a web service that contains layers for performing necessary business logic on the data received from the client and on the data fetched from the database. A demo application utilizing the client functionality is also implemented and will be described at the end of the chapter.

The system utilizes Alize/LIA_RAL to perform the mathematical computations needed for feature processing, voice model training, score computation and score normalization. It also uses SPro to extract features from the provided WAVE audio files.

3.3.1 Client library

The client library is responsible for the initial audio processing, feature extraction and training voice models in the case of registrations. It also computes the impostor log likelihood scores and Z-normalization scores (see section 2.6.3.5) and sends this data to the web service for further processing. Note that the client library does not deal with recording audio. This is expected to have been done by the demo application.

The client exposes a class called the VoiceHandler which is the main entry point of the client application. This class lets demo application developers provide lists containing search paths to the recorded audio files which make up the audio files recorded during registration, or the path to a single audio file for verification purposes.

In order to obtain the best performance from the system, it is necessary to provide a set of audio files used for building the user voice model (called the registration audio files, RAFs) but also an amount of audio segments of voice verifications (called the registration verification audio files, RVAFs) which were recorded during the registration process. The RVAFs are used to compute a set of LLR scores. At the service side, the mean of the scores are computed and used to determine the acceptance threshold for the speaker.

The client is reliant on a configuration file which specifies the web URL to the web service endpoint, it is also possible to assign a different mathematical library to the system other than Alize/LIA_RAL (such as BECARS) but currently no other libraries are implemented.

When verifying unknown users an ordered list of LLR scores are returned. This list contains strings of user name and probability scores with the highest scoring user located at the first element. The demo application is responsible for handling this data and optionally setting a global threshold defining the lowest probability for which users are verified.

3.3.2 Web service

The web service is implemented as a RESTful Windows Communication Foundation (WCF) service. It exposes a set of endpoint operations for registering new users and voice models in the system and for verifying unknown users. It also exposes functions for inserting and extracting background models that the target models are based on. The client program utilizes all these functionalities except for inserting world models. The endpoint used for accessing the web service is called /VoiceService/ and should be added to the base address of the web service.

When registering new users in the system, the service expects to receive UserDataIn data contract (see section 3.3.2.2 for data contract definitions). It is important to note that the service expects to receive a GMM trained for the registering user in binary format and a primary key to the world
model that the user model is based on. The received impostor scores are directly stored in the
database and the verification LLR scores are used to calculate a mean used to determine the
verification acceptance threshold.

Since the web service is RESTful it does not publish any data contract information. The web service
operations and the data contracts are therefore defined in the following sections.

3.3.2.1 Web service operations

Each operation defines a set of attributes. All operations are invoked by the HTTP POST method,
have a return value, an input parameter, and a URI template which defines the path to the web
service operation, relative to the endpoint address. The values returned by the web service and the
input parameters used when calling them are either primitive data types or a data contract objects.
All data returned from the web service is serialized to JSON formatted strings and all data sent to
the web service should be formatted in the same way. The structure of the JSON strings are
described in detail in section 3.3.2.2.

RegisterUser
Allows new users and their voice models to be stored in the system database. It also processes the
impostor scores and Z-normalized score values calculated by the client library.

- Return value: A boolean value indicating whether the registration succeeded or not.
- Input parameter: A JSON formatted string representing the UserDataIn data contract, see
  section 3.3.2.2 for further details.
- Uri template: /register

VerifyVoice
Checks whether data from a feature vector file originates from any of the speakers stored in the
database by iteratively comparing the feature vectors to each of the stored voice models. For each
iteration, an LLR score is calculated and stored in a list together with the name of the user who are
associated with the current voice model.

- Return value: A JSON formatted string corresponding to the VoiceVerificationOut data
  contract containing LLR score and user name pairs.
- Input parameter: A JSON formatted string corresponding to the VoiceVerificationIn data
  contract containing the binary feature vectors and speech labeling data.
- Uri template: /verify

GetWorldModel
Gets a world model (universal background model) from the database based on the primary key of
the database tuple containing the world model. This means that the primary key must be known
beforehand which is quite unlikely in the case of third party client library developers. The
FindWorldModels operation is more suitable in this case.

- Return value: A JSON formatted string representing a single WorldModelOut data contract.
- Input parameter: An integer value of the primary key to find the world model by.
- Uri template: /getworld
FindWorldModels

Returns a list of world models stored in the database. The input is a WorldModelFilter which specifies a set of conditions for the models to be extracted. The conditions are exclusive, that is, the world model must match all of them to be extracted.

- Return value: A JSON formatted string representing a list of WorldModelOut data contracts. If no world models are found, an empty list will be returned.
- Input parameter: A JSON formatted string representing a WorldModelFilter data contract which specifies the conditions of the world models to be extracted.
- Uri template: /getworldbyfilter

3.3.2.2 Data contracts

This section explains the data contracts used by the web service in detail. Both the data contracts internal representation and their JSON string equivalent are explained.

ImpostorScore

The ImpostorScore data contract allows the client library to send the impostor scores that were calculated during the registration process. These impostor scores are used for Z-normalization of the score calculated when verifying unknown user voices. It is only supposed to be used in conjunction with the UserDataIn contract used for user registration.

Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A name for the impostor. This name may be arbitrary with the restriction that it must be unique for all the impostors associated with a user.</td>
</tr>
<tr>
<td>Score</td>
<td>The score of the impostor to user comparison.</td>
</tr>
<tr>
<td>Outcome</td>
<td>The outcome of the score, 1 for success, 0 for false.</td>
</tr>
<tr>
<td>Gender</td>
<td>The gender of the impostor as where 1 represents male and 2 represents female.</td>
</tr>
</tbody>
</table>

JSON representation:

```json
{  "Name" : "string_value",  "Score" : double_value,  "Outcome" : int_value,  "Gender" : int_value }
```
**UserDataIn**

The UserDataIn contract defines the attributes needed for registering a user and his/her voice in the database.

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A name for the impostor. This name may be arbitrary with the restriction that it must be unique for all the impostors associated with a user.</td>
</tr>
<tr>
<td>Gender</td>
<td>The gender of the user making the registration</td>
</tr>
<tr>
<td>VoiceModelData</td>
<td>An array of binary data that represents the GMM generated by the client when registering new users.</td>
</tr>
<tr>
<td>WorldPK</td>
<td>The primary key of the world GMM model used for generating the GMM for the user voice (see section 2.6.3.4 for details).</td>
</tr>
<tr>
<td>ImpostorScores</td>
<td>A list of ImpostorScore data contract objects representing the impostor scores calculated at the registration.</td>
</tr>
<tr>
<td>VerificationScores</td>
<td>The LLR scores calculated from the RVAFs, see section 3.3.1 for details.</td>
</tr>
</tbody>
</table>

**JSON representation:**

```json
{
    "Name" : "string_value",
    "Gender" : int_value,
    "VoiceModelData" : [byte_values],
    "WorldPK : int_value,
    "ImpostorScores" : { "Name" : "string_value", "Score" : double_value, "Outcome" : int_value, "Gender" : int_value },
    "VerificationScores" : [double_values]
}
```

**VerificationScore**

This data contract represents the result of a verification made against the system. It contains the name of each user registered in the database and an LLR score indicating the probability that the user that wishes to be verified is the true speaker.

**Members**

<table>
<thead>
<tr>
<th>UserName</th>
<th>The name of a user registered in the database</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbabilityScore</td>
<td>The LLR probability that this user was the user that made the verification.</td>
</tr>
</tbody>
</table>

**JSON representation:**

```json
{ “UserName” : “string_value”, “ProbabilityScore” : double_value }
```
VoiceVerificationIn

This data contract contains the necessary data for making a verification of an unknown speaker voice. In addition to the binary feature vector data, the corresponding label file, which specifies when the user is speaking, is also required.

Members

<table>
<thead>
<tr>
<th>NormalizedFeatureFileData</th>
<th>The normalized feature vectors extracted from the voice verification audio segment in binary format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpeechLabel</td>
<td>The label used for indicating where the speaker speaks.</td>
</tr>
<tr>
<td>SpeechLabelData</td>
<td>The actual speech labeling data which is a list of strings where each string is formatted like [ss:mm label], where ss is the second and mm is the millisecond of the speech data.</td>
</tr>
</tbody>
</table>

JSON representation:

```
{ “NormalizedFeatureFileData” : [byta_values], “SpeechLabel” : “string_value”, “SpeechLabelData” : [“string_values”] }
```

VoiceVerificationOut

Simple wrapper data contract object containing a list of VerificationScore data contracts. Returned by the VerifyVoice operation.

Members

| VerificationScores | A list of VerificationScore data contracts. |

JSON representation:

```
{ “VerificationScores” : [{“UserName” : “string_value”, “ProbabilityScore” : double_value}] }
```

WorldModelOut

This data contract is returned by the GetWorldModel and FindWorldModels operations. It represents a world model as it is stored in the database.

Members

<table>
<thead>
<tr>
<th>PrimaryKey</th>
<th>The primary key of the tuple containing the world model in the database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelData</td>
<td>The binary data of the voice model.</td>
</tr>
<tr>
<td>MaleFeatures</td>
<td>A boolean value indicating that the world model was based on feature vectors from male speakers.</td>
</tr>
<tr>
<td>FemaleFeatures</td>
<td>A boolean value indicating that the world model was based on feature vectors from female speakers. Both MaleFeatures and FemaleFeatures may be true.</td>
</tr>
<tr>
<td>SizeKB</td>
<td>The size in kilobytes of the world model. May be null.</td>
</tr>
</tbody>
</table>

JSON representation:

```
{ “PrimaryKey” : int_value, “ModelData” : [byte_values], “MaleFeatures” : bool_value, “FemaleFeatures” : bool_value, “SizeKB” : int_value } ```
WorldModelFilter

This data contract is used when searching for world models in the database. It specifies a set of search criteria which are exclusive.

**Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaleFeatures</td>
<td>Boolean value set to true if the model must contain male features.</td>
</tr>
<tr>
<td>FemaleFeatures</td>
<td>Boolean value set to true if the model must contain female features.</td>
</tr>
<tr>
<td>SizeKBGreaterThan</td>
<td>Integer value specifying the value that the size of the world model should be greater than.</td>
</tr>
<tr>
<td>SizeKBLessThan</td>
<td>Integer value specifying the value that the size of the world model should be less than.</td>
</tr>
<tr>
<td>Components</td>
<td>The number of mixture components that the world model should have.</td>
</tr>
<tr>
<td>Dimensionality</td>
<td>The dimensionality of the world model, which is directly related to the size of the feature vectors used when creating the model (see section 2.6.3.4 for details).</td>
</tr>
</tbody>
</table>

3.3.3 Database

The DBMS used for storing user information and voice models is Microsoft SQL Server 9.0.5000 and the constructed database is relational. This has huge advantages over using simple files (as the Alize/LIA_RAL toolkit does) in that it is easy to keep track of the variables necessary for registration and verification purposes. These variables include keeping track of which world model were used for generating which user voice model, the generated impostor scores and models and also the characteristics of the world model.

The file dependency of Alize/LIA_RAL and the relational database approach used in the system are cause of some problems.

First, it is important that the binary data (representing voice models) sent from the client library is stored in the database in the exact same way as it was stored in file by Alize/LIA_RAL. Because of this, great care must be taken to keep the files and database binary data in the same byte orders. If files generated by Alize/LIA_RAL on a big endian CPU are stored in the database, serious problems will arise when retrieving and reading that data on a little endian system. This is an area of improvement in future releases.

Second, for the sake of data consistency, it is important that the relationship between user voice models and world models are defined in a correct way. This relationship is currently handled by the voice client library, which submits the primary key of the world model used to generate the user voice models. Because of the binary file format of the data it is hard if not impossible to perform this consistency check in the database layer, since the world and voice model data only contains floating point values indicating the mean and variances of the GMM mixture components. This is why the task of associating user voice models and world models are handled by the client library before submitting the data to the web service and thus the database layer.
3.3.4 Demo application

The demo application consists of a simple Windows Presentation Foundation (WPF) project which allows the user to register and verify their voices and identities in the system. The application uses a Microsoft Kinect unit for audio recording as specified in section 1.2.

The demo application makes some assumptions about the conditions in which the audio is recorded in. It is assumed that only one single speaker is speaking during registration and verification. This is because the current client library implementation does not make any attempt to label speech data with different speaker identities. It is further assumed that the audio files recorded contain a sufficient amount of data (over 10 seconds per audio file) and that most of the data recorded is of human speech.

3.4 Flow design

This section of the report aims at describing how data is processed and passed between the various components previously described. It aims at providing an overview of the system as a whole. The system will be described with the demo application as starting point, and proceed to demonstrate the interaction between the client library, the web service and the database manager.

The demo applications are responsible for four actions: Recording audio of human voices, providing the name and gender for the users to be registered to the client library, providing the absolute paths to the registration and verification files, and to indicate the installation path of the client library. The reasoning for the last action is that the client library needs to run external executables in separate processes and the demo application should only have to include the client library assembly in order to use it. The demo application may invoke two methods of the client library, either to register a voice in the system, or to verify an unknown speaker.

The client library expects two lists to be provided when registering new users. These lists contain the absolute paths to the audio files recorded by the system. The first list is based on RAFs and the second is based on RVAFs (see section 3.3.1) and are used for model training and verification score estimation respectively. The client library fetches an appropriate world model from the web service to base the speaker voice model on. The model is then sent to the web service together with the computed impostor and Z-normalized scores. For verification, the client library expects a single path to an audio file. This audio file is processed to a normalized feature vector which is sent in binary format to the web service which compares it to the stored speaker models.

When receiving registrations, the web service calculates the mean of the received LLR scores. This mean score is then lowered by a small amount and used as the threshold for verification. Fig. 10 shows the system flow for registration and Fig. 11 shows the flow for verification.
Fig. 10: System flow for the registration process.

Fig. 11: System flow for verification. The user names and probabilities are returned to the demo-application at the end.
4 Measurements

This section will discuss the preconditions and the methods used for collecting the measurements that were made upon the system.

4.1 Preconditions

The most constraining precondition posed on the system is the lack of an scientifically composed speech corpus for background model training (as mentioned in section 1.5). The voice data used for background model training comes from speakers using both landline telephones and cell phones with varying audio quality and noise levels. There existed no metadata describing the contents of each audio file such as the gender of the speaker or what handset was used for the session. The background model is therefore mixed gender and mixed handset and contains 855 different speakers all of which speaks Swedish.

Because of the context in which the recordings were made (that is, customer feedback to a company) it is assumed that each speaker has only been recorded once. This makes it almost impossible to use subsets the previously mentioned speech data for training user models for the sake of evaluating the performance of the Alize/LIA_RAL configuration. The Waxholm corpus was used for this task and it contains 68 speakers.

Another serious restraint is the lack of recordings made with Kinect units. Gathering speech data was therefore needed in order to train user models. Because of this, a small amount (even less than the Waxholm corpus) of user voice models could be generated. These models are also based on the same background model as the voice models from the Waxholm corpus.

4.2 Evaluation methods

The measurements were taken in four sessions, with two different sets of SPro and Alize/LIA_RAL configurations for both the Waxholm and the Kinect users.

The number of speakers in the Waxholm corpus is small and the measurements and the results should be considered with this in mind. The same considerations must also be made towards the Kinect speakers, though in a much higher grade since these are much fewer in numbers.

The Waxholm corpus is not used together with the rest of the VoiceR implementation since the purpose of that framework is to evaluate the performance of the recordings from a Kinect. Instead, the speech files are processed directly from the hard drive and all the Alize/LIA_RAL programs are run on the local client computer.

The Waxholm recordings are short in length; each one is about 3 to 7 seconds. Each speaker model is therefore constructed from a set of feature files representing 8 to 10 recordings. The verifications made against the models are also made up of recording sessions of similar length.
It is important to note that the Waxholm and Kinect measurements are not comparable to each other. The reasoning behind using the Waxholm corpus is because of the extremely small amount of available Kinect speakers, and it is only used to evaluate how the different configurations of the Alize/LIA_RAL and SPro toolkits performs with a reasonable amount of users. Also, the nature of the words spoken in the Waxholm corpus is based on scenarios in which the speaker is asked to pose questions to a service desk [1]. This means that the speakers do not simply speak text shown on a monitor, and thus the text-independence for the Waxholm corpus is higher than that for the Kinect speakers, which were all asked to speak the same predetermined words.

The Kinect speakers amount to 6 persons with a total of 264 verification sessions. Each voice model was created by having each speaker count from 1 to 30, and the subsequent verifications were also based on counting different numbers and the full name of the speaker.

The same Alize/LIA_RAL configurations were used for both the Waxholm and Kinect speakers and can be found in Appendix A.
5 Results

This chapter presents the results taken from the measurements and is divided into two parts, one for each version of the the underlying background model. These sections are in turn divided further into sub sections displaying the results of test runs of speakers from Waxholm and Kinect corpus respectively.

The results of all the measurement sessions are presented in Detection Error Tradeoff (DET) diagrams. A DET diagram plots the false rejection probabilities against the false acceptance probabilities. A transform is applied to both the horizontal and vertical axes in such a way that if both the mentioned probability distributions are normally distributed, the resulting plot in the diagram represents a straight declining line [24].

The line that constitutes the DET diagram will intersect a point in which the false rejection and false acceptance probabilities are equal. This point is called the equal error rate and shows the general performance of the system. A low EER value suggests a better performing system in the sense that it is more likely to make a correct verification (or rejection) of an unknown speaker.

Because of the small amount of Kinect speakers, the DET diagram will be complemented by tables showing the mean LLR scores of each speaker and model. These tables should be read accordingly: Each row shows the mean LLR scores for utterances made by speaker in the leftmost column as compared by each voice model shown in the top row. If the system performance is good, the highest number in each row should be where the speaker and voice model code are the same. The highest score in each row in the table is marked in green, and the lowest is marked in red. The gender of each speaker is noted by the first letter in their name where a capital M indicates a male speaker and a capital F indicates a female.

Since the mentioned tables only show mean values, no detailed intelligence can be retrieved as of the number of times that the system failed to correctly identify each speaker. Therefore another table is also used, which show the number of times that a speaker got the highest LLR score in relation to the total number of verification sessions made for the same speaker.

It is important to note that the results cannot be viewed as being of statistical significance for the Waxholm corpus and especially not for the Kinect speakers, seeing as the number of users and verification sessions are low for both databases. The results should therefore be viewed with caution.

5.1 World model based on features with 60 coefficients

The feature extractions made on all the audio file were configured to extract 19 cepstral coefficients along with their first and second order derivatives. This results in three sets in which each one contains 19 coefficients. In addition, an energy coefficient is calculated for each set and the total number of cepstral coefficients are therefore 60.

5.1.1 Waxholm

Following is the obtained results of the Waxholm corpus. The experiments were conducted with a total of 134 300 verification utterances of which 1975 were true speaker scores and the rest were impostor scores. The resulting DET curve can be seen in Fig. 12.
The DET curve shows an ERR value of about 32 %. This is a quite high number seeing as modern high performing systems attain EER values between 5 and 10 %. One plausible explanation for this low performance is the poor quality of the background model.

### 5.1.2 Kinect

The DET curve for the Kinect speakers in the current configuration is shown in Fig. 13. As previously mentioned, a DET curve should be linear if the underlying data is normally distributed. This cannot be said for Fig. 13 since it flattens out at the EER point which in turn indicates that the data is not normally distributed and the EER value of about 17 % should be viewed with caution.
The table below show the mean LLR score as utterances were inserted from each speaker into every speaker voice model. As seen, the matching speaker/model pairs attained the highest LLR score on average which indicates that correct speaker identification were performed. Interestingly, the lowest scores were obtained by speakers of the opposite gender, making for a clear distinction between the two. As mentioned in section 2.6.3.5, any scores above 0 indicates a possible match so, in theory, all the cells that are of white or red color should have a score under 0. This is clearly not the case for these measurements.

<table>
<thead>
<tr>
<th>Speaker voice models</th>
<th>Speakers</th>
<th>F0001</th>
<th>F0002</th>
<th>M0003</th>
<th>M0004</th>
<th>M0005</th>
<th>M0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0001</td>
<td>1.4692965</td>
<td>0.917369071</td>
<td>0.623227043</td>
<td>0.574906</td>
<td>0.381554371</td>
<td>0.3091911286</td>
<td></td>
</tr>
<tr>
<td>F0002</td>
<td>0.933550929</td>
<td>1.5789882</td>
<td>0.537911157</td>
<td>0.719235214</td>
<td>0.215610714</td>
<td>0.304442857</td>
<td></td>
</tr>
<tr>
<td>M0003</td>
<td>0.556530862</td>
<td>0.607949363</td>
<td>1.251778475</td>
<td>1.02586413</td>
<td>0.6223315</td>
<td>0.708369875</td>
<td></td>
</tr>
<tr>
<td>M0004</td>
<td>0.587928248</td>
<td>0.632109663</td>
<td>1.069210188</td>
<td>1.855393125</td>
<td>0.7198175</td>
<td>1.126335256</td>
<td></td>
</tr>
<tr>
<td>M0005</td>
<td>0.480463857</td>
<td>0.283004571</td>
<td>0.819870786</td>
<td>0.708113943</td>
<td>1.404381857</td>
<td>0.700785</td>
<td></td>
</tr>
<tr>
<td>M0006</td>
<td>0.418049143</td>
<td>0.432559014</td>
<td>0.889551571</td>
<td>1.213466286</td>
<td>0.706309657</td>
<td>2.3724284</td>
<td></td>
</tr>
</tbody>
</table>

The following table shows how many times each speaker attained the highest LLR scores for each of his/her verification sessions. If a speaker did not get the highest LLR score, another (impostor) speaker was identified as being more likely to be the actual speaker. Therefore the table shows the amount of correct identifications in relation to the number of verification sessions per user.

<table>
<thead>
<tr>
<th>User</th>
<th># Highest Score</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0001</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>F0002</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>M0003</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>M0004</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>M0005</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>M0006</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Summing the times that correct identifications were made and dividing this with the total number if sessions yields the success rate of the test run. The success rate is 36/44 or approximately 82%.

5.2 World model based on features with 24 coefficients

The world model used for the following experiments is based on features with 24 coefficient. The reasoning behind using another background model is that some previous research [10] recommends the number of feature coefficient to be between 24 and 40. This configuration only uses the raw cepstral coefficients.
5.2.1 Waxholm

The results for the Waxholm corpus in the 24 coefficient configuration are shown in Fig. 14. Compared to the data attained for the 60 coefficient configuration, a slight decrease in performance can be seen as the EER value has been shifted from 32 % to about 37 %.

![Fig. 14: DET curve for LLR scores for the Waxholm corpus]

This result indicates that using the first and second derivatives of the cepstral feature vector (which is the case in the 60 coefficient configuration) improves the system slightly.

5.2.2 Kinect

The DET curve of the results obtained from the Kinect speakers in 24 coefficient configuration is presented in Fig. 15. As can be noted, the EER value has not changed in comparison to the 60 coefficient configuration. The curve does however indicate a higher false rejection rate as the false acceptance rate lowers, which is a quite significant changed from the 60 coefficient configuration.

![Fig. 15: DET curve for LLR scores for the Kinect speakers]
The table for the mean LLR scores show that slightly lower values were obtained when testing as compared to the 60 coefficient configuration. Speakers were still correctly identified on average.

<table>
<thead>
<tr>
<th>Speakers</th>
<th>F0001</th>
<th>F0002</th>
<th>M0003</th>
<th>M0004</th>
<th>M0005</th>
<th>M0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0001</td>
<td>☒ 1,307191</td>
<td>0,738863079</td>
<td>0,467204519</td>
<td>0,439227757</td>
<td>0,304462665</td>
<td>0,203857407</td>
</tr>
<tr>
<td>F0002</td>
<td>0,72254825</td>
<td>☒ 1,184923386</td>
<td>0,4180244</td>
<td>0,524158464</td>
<td>☒ 0,179066914</td>
<td>0,212281614</td>
</tr>
<tr>
<td>M0003</td>
<td>0,429769681</td>
<td>0,500173738</td>
<td>☒ 1,143199619</td>
<td>0,848445425</td>
<td>0,542641794</td>
<td>0,566850688</td>
</tr>
<tr>
<td>M0004</td>
<td>0,498850916</td>
<td>0,530379494</td>
<td>0,834765575</td>
<td>☒ 1,443550288</td>
<td>0,570088628</td>
<td>0,837033997</td>
</tr>
<tr>
<td>M0005</td>
<td>0,377132887</td>
<td>0,265088993</td>
<td>0,6872128</td>
<td>0,569001971</td>
<td>☒ 1,1262323</td>
<td>0,526584738</td>
</tr>
<tr>
<td>M0006</td>
<td>0,339001679</td>
<td>0,353259207</td>
<td>0,721795264</td>
<td>0,943895164</td>
<td>0,521970182</td>
<td>☒ 1,90621245</td>
</tr>
</tbody>
</table>

A minimal raise in performance can be seen in the table below since M0003 obtained the highest LLR score one additional time compared to the 60 coefficient configuration.

<table>
<thead>
<tr>
<th>User</th>
<th># Highest Score</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0001</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>F0002</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>M0003</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>M0004</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>M0005</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>M0006</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The success rate of the system is slightly higher for this configuration: 37/44 or approximately 84.1%.
6 Conclusions

This chapter will discuss the usage of the Alize/LIA_RAL toolkit, the constructed VoiceR system that implemented that toolkit and the results from the previous chapter. An evaluation of the practical possibilities of the VoiceR system will also be given.

6.1 The toolkits

Learning and configuring the Alize/LIA_RAL toolkit proved to be the major challenge when building the VoiceR system. Care had to be taken to set each of the parameters to the correct values without any proper documentation of their significance. Another practical hindrance was that no description of how various file contents should be organized is given.

The toolkit does however use proper well established algorithms and mathematical models and produces results that are to be expected (based on the results found in chapter 5).

Should one consider using the Alize/LIA_RAL toolkit, it is important to note that both frameworks have documentation of sub standard qualities in the sense that some parts are written in English and other parts are written in French. Further, it does not provide any reliable information for getting started and the arguments for each program are not explained in detail.

6.2 The VoiceR system

The constructed library, web service and database resulted in a quite simple C# interface for ASR demo application developers. The pipeline methodology worked fairly well but interfacing directly with the source code should yield a faster system since no external processes will have to be started and wait for each other.

Some problems exist for the system, especially for the web service implementation. The buffer size of the services receiving raw feature vectors may overflow if the audio file that these are based on is large. Further, the flexibility of configuring the VoiceR system is hugely based on the already complex configuration style of the underlying Alize/LIA_RAL system, and future work might be done to abstract the configuration to a single set of VoiceR configuration files which translates to Alize/LIA_RAL.

One line of thought that was considered in the beginning of the project was that the VoiceR system should be able to support other ASR frameworks, not just Alize/LIA_RAL. This idea was not pursued due to time limitations and the VoiceR system is currently only supporting Alize/LIA_RAL tools.

A small demo application was developed, but it is currently not a fully functioning automatic reception desk. It allows users to register themselves in the database but after verification, the system simply presents the computed LLR scores and no further action is taken. Further development of the demo application could simply be based on picking the highest LLR score (if it exceeds a certain threshold) and contacting a web service with the name of the user for further processing.

An automatic reception desk is of course not the only implementation that the VoiceR system can handle. Other possible implementations include security/authorization (though such an demo application should be coupled with other security measures) and speaker dependent voice control.
6.3 The test results

The results presented in chapter 5 show some quite interesting properties of the Alize/LIA_RAL system and configuration. To summarize: Using a world model based on feature vectors with 24 coefficients proved to yield not as good results as the one based on 60 coefficients. That the 60 coefficient background model is of higher dimensionality than the recommended 24 to 40 does not seem to have any negative impact on the system performance.

The results of the Waxholm measurements show some pretty high EER values. The reason behind is probably that the background model used for user voice model training is of poor quality. If a well performing ASR system is to be built, it is important that the background model is based on voice recordings of similar characteristics as the recordings being made by the users of the system. That is, if the ASR system is supposed to be used over telephone, the background model should be based on speakers speaking into a telephone. Another source of improvement is to separate the background model into two, one for female and one for male speakers.

Though the amount of Kinect speakers were small, the raw LLR values looked promising, which is somewhat surprising when considering the high EER value of the Waxholm test runs. The reason as to why seemingly perform better for the Kinect speakers is probably due to a higher dependency on text and the high quality recordings of a Kincet camera. The background models used for the Kinect speakers were the same as the one used for the Waxholm corpus.

That the mean LLR scores of the Kinect speakers were higher than 0 is probably due to the significant difference between the recordings made to build the background model and the recordings made when registering and verifying the users. This does not have to pose any real problems if the threshold for correctly verifying speakers is increased. Examining the tabular data in chapter 5, such a threshold might be set to 1 for the Kinect speakers since almost every speaker that (on average) obtained a score above 1 turned out to be the correct speaker.

Given the small amount of users for both the Waxholm and the Kinect corpus, it is important to state that the results cannot be viewed as being of statistical significance and both the conclusions and measurements must be viewed with this in mind.
7 Recommendations

In this chapter some guidelines are pointed out, should the company (Nordicstation) wish to further develop or investigate ASR.

The GMM approach offers more flexibility as it is used for text-independent applications. If the users are not cooperative (e.g. surveillance) or if the flexibility is needed for high user friendliness regarding word pronunciation or similar, then building a GMM system is highly recommended. Text-dependent HMM based system may yield a higher performance at the cost of flexibility.

For GMM based systems, the need for a scientifically composed speech corpus for background model training cannot be underestimated and acquiring one which fit the needs of the application (such as the spoken language of the users and clear male/female segregation of the audio files) will result in higher system performance.

Further investigations into HMM based systems may also be undertaken as well as combinations between the different system types.

The results shown in chapter 5 indicates that there should not be any problems using the Microsoft Kinect unit as recording device, and ASR coupled with other biometric recognition systems (such as facial recognition) is possible to further improve the verification success rates.
References


5. *An overview of text-independent speaker recognition – From features to supervectors*, Tomi Kinnunen and Haizhou Li, 2009-07-01


11. *All-Pole Modelling of Speech: Mathematical Analysis Combined with Objective and Subjective Evaluation of Seven Selected Methods* – Carlo Magi 2005-12-05


20. A Tutorial on Text-Independent Speaker Verification - Frédéric Bimbot et.al 2003-08-08

21. SPro documentation – Guillaume Gravier 2004-03-05

22. Phone lines demystified – Telecom Audio 2002-09-09


24. The DET Curve in Assessment of Detection Task Performance – A. Martin, G. Doddington, T. Kamn, M. Ordowski, M. Przybocki
Appendix A – SPro and Alize/LIA_RAL configurations

This appendix documents the configuration parameters used for SPro and the Alize/LIA_RAL executables. The Alize/LIA_RAL configurations are only shown for the 60 coefficient configurations because the only parameter that needs to be changed for each configuration is vectSize, which is changed from 60 to 24 for the 24 coefficient configuration.

SPro

The SPro program does not have a whole configuration file, it is simply run as a command line program and the arguments are read at the same time. The sfbcep program was used and run in the following configurations:

```
sfbcep.exe -F WA VE -p 19 -e -D -A [InputFile] [OutputFile]
sfbcep.exe -F WA VE -p 24 [InputFile] [OutputFile]
```

for the 60 coefficient and 24 coefficient configurations respectively. The InputFile is an audio file in .WAV format and the OutputFile is the feature vector file generated by the program. These arguments are provided dynamically by the system depending on the demo-application (see sections 3.3.1 and 3.3.4).

EnergyDetector

The value of inputFeatureFilename parameter is generated by the system, but it always corresponds to a feature vector file generated by the SPro program. The configuration file is:

```
*** EnergyDetector Config File
***
inputFeatureFilename [GENERATED]
loadFeatureFileExtension .prm
saveFeatureFileExtension .norm.prm
minLLK -200
maxLLK 200
bigEndian false
loadFeatureFileFormat SPRO4
saveFeatureFileFormat SPRO4
saveFeatureFileSPro3DataKind FBCEPSTRA
featureServerBufferSize ALL_FEATURES
featureFilesPath ./data/
mixtureFilesPath ./data/
lstPath ./data/
labelOutputFrames speech
labelSelectedFrames all
addDefaultLabel true
defaultLabel all
saveLabelFileExtension .lbl
labelFilesPath ./data/
frameLength 0.01
writeAllFeatures true
```
segmentalMode file
nbTrainIt 8
varianceFlooring 0.0001
varianceCeiling 1.5
alpha 0.125
mixtureDistribCount 3
vectSize 60
baggedFrameProbabilityInit 0.1
debug false
verbose false

NormFeat

The inputFeatureFilename is dynamically generated by the system but it corresponds to the feature file generated by the SPro program and the label file generated by the EnergyDetector program (same name but different extensions for both files) The configuration for the NormFeat program is:

*** NormFeat config File
***

inputFeatureFilename [GENERATED]
bigEndian false
loadFeatureFileFormat SPRO4
saveFeatureFileFormat SPRO4
loadFeatureFileExtension .prm
saveFeatureFileExtension .norm.prm
featureServerBufferSize ALL_FEATURES
featureFilesPath ./data/
labelFilesPath ./data/
sampleRate 100
saveFeatureFileSPro3DataKind FBCEPSTRA
inputFeatureFilename 0_1
labelSelectedFrames speech
segmentalMode false
writeAllFeatures true
frameLength 0.01
mode norm
featureServerMode FEATURE_WRITABLE
featureServerMemAlloc 1000000

TrainTarget

It is important to note that the contents of the files model._ndx and world that the parameters TargetIdList and inputWorldFilename points to, are generated by the program. The world file is fetched from the database.

*** TrainTarget Configuration File
***
distribType GD
mixtureDistribCount 512
maxLLK 200
minLLK -200
bigEndian false
saveMixtureFileFormat RAW
loadMixtureFileFormat RAW
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
loadMixtureFileExtension .gmm
saveMixtureFileExtension .gmm
loadFeatureFileExtension .norm.prm
featureFilesPath ./data/
mixtureFilesPath ./data/
labelFilesPath ./data/
lstPath ./data/
baggedFrameProbability 0.4
mixtureServer false
labelSelectedFrames speech
useIdForSelectedFrame false
normalizeModel true
targetIdList ./data/model_ndx
TargetIdList ./data/model_ndx
nbTrainIt 3
nbTrainFinalIt 0
inputWorldFilename world
alpha 0.75
MAPAlgo MAPOccDep
meanAdapt true
MAPRegFactorMean 10
featureServerMask 0-59
frameLength 0.01
vectSize 60
MAPRegFactor 14

**ComputeTest**

The *world* file is fetched from the database and the verification_ndx is a static file that is generated by the system and specifies which impostor features to compare against the speaker model.

*** ComputeTest Config File

***

distribType GD
loadMixtureFileExtension .gmm
saveMixtureFileExtension .gmm
loadFeatureFileExtension .norm.prm
mixtureDistribCount 512
maxLLK 200
minLLK -200
bigEndian false
saveMixtureFileFormat  RAW
loadMixtureFileFormat  RAW
loadFeatureFileFormat  SPRO4
featureServerBufferSize  ALL_FEATURES
featureFilesPath  ./data/
mixtureFilesPath  ./data/
labelSelectedFrames  speech
labelFilesPath  ./data/
frameLength  0.01
segmentalMode  completeLLR
topDistribsCount  20
computeLLKWithTopDistribs  COMPLETE
ndxFilename  data/verification_ndx
inputWorldFilename  world
outputFilename  ./data/result.res
gender  M
channelCompensation  NOTUSED
Scoring  true
vectSize  60
featureServerMask  0-59

**ComputeNorm**

Used for Z-normalization. The *ImpostorMerge* file is a score file generated by the ComputeTest program in which the impostors are compared against a speaker voice model. The *result* file is the log likelihood score calculated at verification time between a feature vector from an unknown speaker and a target model. This is done by the ComputeTest program as well.

*** ComputeNorm configuration file
***
maxScoreDistribNb  500
maxIdNB  1000
maxSegNb  1000
selectType  noSelect
normType  znorm
znormNistFile  ./data/impostor/ImpostorMerge.imp
testNistFile  ./data/result.res
outputFileBaseName  ./data/finalsore
cohortFilePath  ./data/
cohortFileExtension  imp

**TrainWorld**

This program is not really used in the system pipeline. It is run in a separate application and stored directly to the database. The configuration is provided for the sake of reference. The *norm_feats* is a list of feature vector files normalized by the NormFeat program.

*** TrainWorld configuration file
***
distribType  GD
mixtureDistribCount: 512
maxLLK: 200
minLLK: -200
bigEndian: false
saveMixtureFileFormat: RAW
loadMixtureFileFormat: RAW
loadFeatureFileFormat: SPRO4
featureServerBufferSize: ALL_FEATURES
loadMixtureFileExtension: .gmm
saveMixtureFileExtension: .gmm
loadFeatureFileExtension: .norm.prm
featureFilesPath: ./data/
mixtureFilesPath: /
labelFilesPath: ./data/
lstPath: ./data/
labelSelectedFrames: speech
addDefaultLabel: true
defaultLabel: speech
normalizeModel: true
featureServerMask: 0-59
frameLength: 0.01
vectSize: 60
inputFeatureFilename: ./data/norm_feats.lst
baggedFrameProbabilityInit: 0.04
baggedFrameProbability: 0.2
initVarianceCeiling: 10
initVarianceFlooring: 0.5
finalVarianceFlooring: 0.5
finalVarianceCeiling: 10
nbTrainIt: 5
nbTrainFinalIt: 10
outputWorldFilename: world
fileInit: false