Evaluation of the *ALIZE / LIA_RAL* Speaker Verification Toolkit on an Embedded System

Aitor Hernández López

Technische Universität Wien
Institut für Computertechnik (ICT)

February of 2015

Under supervision of

Univ. Prof. Dr. Hermann Kaindl
and
Dr. Dominik Ertl

Abstract

Text-independent *speaker verification* is the computing task of verifying a user's claimed identity using only characteristics extracted from their voices, regardless of the spoken text. Nowadays, a lot of speaker verification applications are being implemented in software, and using these systems on embedded systems (PDAs, cell phones, integrated computers) multiplies their potential in security, automotive, or entertainment applications, among others. Comprehension of speaker verification requires a knowledge of voice processing and a high mathematical level. Embedded system performance is not the same as offered by a workstation. So, in-depth knowledge of the target platform where the system will be implemented and about cross-compilation tools necessary to adapt the software to the new platform is required, too. Also execution time and memory requirements have to be taken into account to get a good quality of speaker verification.

In this thesis we evaluate the performance and viability of a speaker verification software on an embedded system. We present a comprehensive study of the toolkit and the target embedded system. The verification system used in this thesis is the *ALIZE / LIA_RAL Toolkit*. This software is able to recognize the identity of a client previously trained in a database, and works independently of the text spoken. We have tested the toolkit on a 32-bit *RISC ARM* architecture set computer. We expect the toolkit can be ported to comparable embedded system with a reasonable effort.

The findings confirm that the speaker verification results on work station are comparable than in an embedded system. However, time and memory requirements are not the same in both platforms. Taking into account these results, we propose an optimization in the speaker verification test to reduce resource requirements.

Resumen

La verificación de locutor independiente del texto es la acción de validar la identidad de un usuario usando únicamente características extraidas de su voz, sin tener en cuenta el texto pronunciado. Hoy en día, multitud de software de verificación de locutor ha sido implementado para funcionar en ordenadores personales, pero usar estas aplicaciones en sistemas embedidos (Smartphones, teléfonos, ordenadores integrados) multiplica su potencial en campos como la seguridad, el sector del automóvil u otras aplicaciones de entretenimiento. La comprensión teórica de los sistemas de verificación de locutor requiere conocimientos de procesado de voz y un nivel alto de matemática algorítmica. El rendimiento de estos sistemas embedidos no es el mismo que los que ofrecen los ordenadores personales, así que hace falta un conocimiento exhaustivo de la plataforma en la cual se va a integrar la aplicación, así como un conocimiento de las herramientas de compilación cruzadas necesarias para adaptar el software a la nueva plataforma. Los requerimientos de tiempo y memoria también deben ser tenidos en cuenta para garantizar una buena calidad de verificación.

En este proyecto, se evaluará el rendimiento y la viabilidad de un sistema de verificación de locutor integrado en un sistema embedido. Se presenta un estudio exhaustivo de las herramientas del software, así como de la plataforma de destino utilizada. El sistema de verificación usado en este proyecto ha sido la herramienta ALIZE / LIA_RAL. Este software es capaz de reconocer la identidad de un cliente entrenado con anterioridad y almacenado en una base de datos, y trabaja independientemente del texto pronunciado. El software ha sido testado en una máquina de pruebas con un procesador de 32-bit RISC ARM, pero el sistema podría ser portado a otros sistemas sin problemas añadidos.

Los hallazgos durante el proyecto confirman que los resultados de la verificación en un sistema embedido son similares a los obtenidos en el PC. Sin embargo, los requerimientos de tiempo y memoria no son los mismos en las dos plataformas. Teniendo en cuenta estos resultados, se propone una optimización de los parámetros de configuración utilizados en el proceso de test para reducir considerablemente los recursos utilizados.

Contents

List of Figures

List of Tables

Este trabajo va dedicado con mucho cariño a mis padres y a mi hermana, por su tenacidad y su apoyo encomiable durante la realización de este largo y duro proyecto. También a Dominik, por su paciencia durante mi estancia en Viena, a mis amigos, y a toda la gente que de alguna manera u otra forma parte de mi vida.

Chapter 1

Introduction

In our everyday lives there are many types of interfaces for human-computer interaction, for instance: graphical user interfaces, speech interfaces, etc. However, amongst these types, speech input is regarded as a very powerful form because of its *rich character*. Among the context of spoken input (words), *rich character* also refers to gender, attitude, emotion, health situation and identity of a speaker. Such information is very important for an efficient communicative interaction between humans and gains importance for human-computer interaction (HCI).

From the signal processing point of view, speech input can be characterized in terms of the signal carrying message information. The waveform could be one of the representations of speech, and this kind of signal is very useful in practical applications. We can get three types of information from a speech signal: Speech text (transcription of the message), language (English, German, etc.) and speaker identity (person who utters the message) [7]. In Figure 1.1, we can see these three main types illustrated.

Due to its large number of applications, Automatic Speaker Verification (ASV) has become an attractive domain of study in the area of signal processing and information in the last decades. In combination with Automatic Speech Recognition (ASR), machines capable of "understanding" humans were in the mind of scientists and investigators for years [4]. More recently, significant steps have been made in this direction, for example using Hidden Markov Models (HMM) for speech and speaker recognition [14, 17].

Currently, applications that incorporate ASV or ASR are used in many areas (robotics, telephony, entertainment). Electronic devices capable of speech recognition evolve towards miniaturization (mobile phones, laptops, micro controllers, etc.), which can lead to a decrease in available device resources (processing power, memory, etc.). Potentially, this leads to a loss of quality in *speaker recognition rates*[1].

There are three main areas where Speaker Recognition (SR) is intended to be used: *authentication*, *surveillance* and *forensic*. SR for authentication is well understood and allows users to identify themselves using only their voices as recognition parameters. This

[1]Recognition rate refers to the quality in SR process, i.e. percentage of speakers that a recognition tool can recognize correctly.

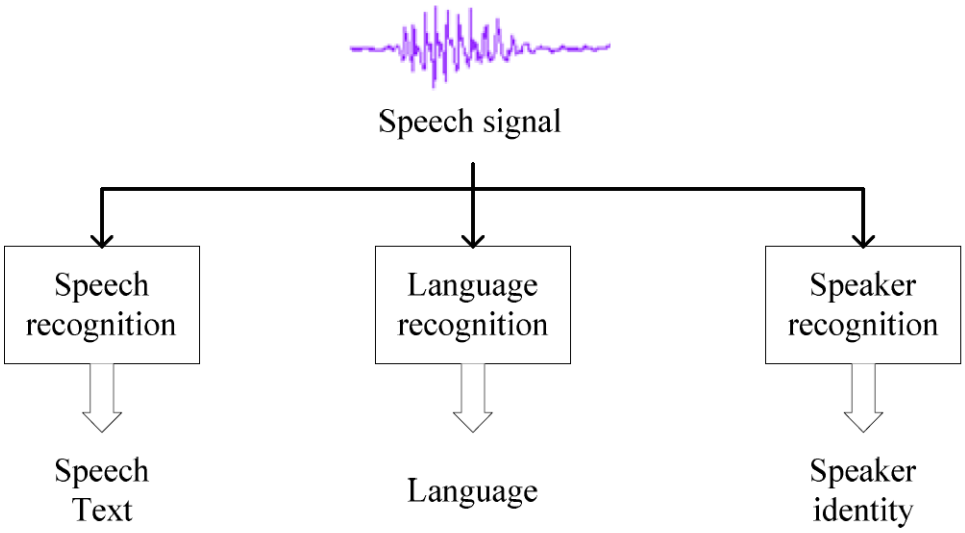

Figure 1.1: Three main types of information transmitted in a speech signal.

Speech text, language and identity of speaker are the three main recognition fields of any speech input.

can be more efficient than carrying keys or memorizing PIN codes. SR for surveillance may help one to filter a high amount of data, for example telephone or radio conversations. SR in forensic can help one to prove the identity of a recorded voice and can help to convict a criminal or discharge an innocent person in court.

When using such applications, one might not want to carry a laptop or a desktop system, which makes an embedded device suited as an underlying platform. Accordingly, one important challenge is to get sufficient performance of a given system having a set of resource constraints.

In this master thesis, we analyze, improve and test the voice processing and speaker recognition software ALIZE / LIA_RAL. The ALIZE library is a set of algorithms for automatic speaker recognition. LIA_RAL is the application that uses the ALIZE library.

The main goal of this work is to evaluate this software to analyze, improve and test its performance in an embedded system, and to compare the system results with those obtained on a laptop. Finally, we present an optimization of the source code, to get the same results with less time and processing requirements on both platforms.

1.1 Objectives of this Master Thesis

This thesis studies a speaker verification software that is ported on an embedded device. The main objectives are:

- To study the historical perspectives and current research in the field of speaker verification software on embedded devices.
- To analyze the structure of a speaker verification system, specifically the ALIZE / LIA_RAL software.
- To choose a proper operative system and cross-compiling tools to port this software to an embedded device.
- To analyze performance results on an embedded device, and to compare it with the results obtained with a laptop.
- To present and improve source code of the ALIZE/ LIA_RAL software to get a solution with a better performance.
- To analyze the results after improving the software, and to compare them with previous iterations, as well as with the results from a laptop.

1.2 Motivation and Opportunities

Theory and practice of speaker recognition have improved recently. Systems capable of getting recognition results with error rates close to 0%, almost in real time, can be achieved today with low effort. The idea of porting such recognition systems on embedded devices increases significantly the range of applications and opportunities of speech technologies.

There are several embedded verification systems, in addition to many other non-integrated systems [20]. ALIZE / LIA_RAL software has never been ported and tested in such an integrated system. Taking into account the good results of the PC version, we study its performance on an embedded system.

1.3 Working Plan

The working plan followed during this master thesis is in chronological order:

- Studying speech processing and speaker recognition documentation.
- Gaining in-depth-knowledge of the embedded systems and cross-compilation tools.
- Understanding, compiling and testing of ALIZE / LIA_RAL software on a laptop.
- Cross-compiling and porting of the software to an embedded system.
- Testing of the results on an embedded system and compraising with the results from the laptop.
- Improving the source code, analyzing the new results, and comparing them again with the laptop results.

1.4 Overview

The material of this thesis is arranged into nine chapters as follows:

- Chapter 1 introduces the work and gives a brief description of speaker verification.
- Chapter 2 Presents background state-of-the-art of the speaker verification systems, as well as speech processing on embedded systems.
- Chapter 3 describes the four principle steps of the speaker verification process: Feature extraction, training stage, testing stage and decision making.
- Chapter 4 presents the tools ALIZE and LIA_RAL and describes them in detail.
- Chapter 5 explains the embedded device used in this master thesis, its principle features and the cross-compilation process.
- Chapter 6 explains different improvements used to optimize performance.
- Chapter 7 describes the setup of data used on testing and summarizes the results obtained before and after the improvements.
- Chapter 8 discusses the speaker verification results that we gained from the embedded and laptop systems.
- Chapter 9 concludes this master thesis and provides an outlook.

Chapter 2

Background and State-of-the-Art

In this section, we present background information and the state-of-the-art of speaker verification. First, we introduce biometrics concept. Then, we explain speech processing and speaker recognition, as well as embedded systems state-of-the-art. Finally, we explain specific works on speaker verification. Due to the huge research effort in this field, we only consider SV work on embedded systems related to this master thesis.

2.1 Biometrics and Speaker Recognition

The biometrics concept comes from the two words bio (Greek: life) and geometry (Greek: measured). Thus, this combination consists of the characterization of human physical or behavioral traits, in order to uniquely identify humans. Nowadays, access control and surveillance are the most relevant applications where biometrics are used.

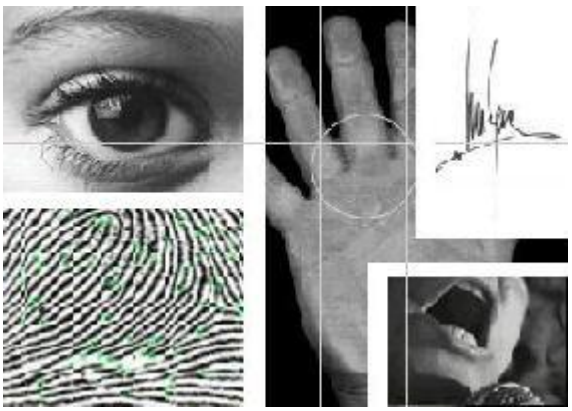

Figure 2.1: Different types of biometrics.

Iris scanning, fingerprints, hand scanning, signatures analysis and voice recognition are the most common biometric technologies used nowadays.

Due to the possible transferability and falsification of personal documents such as passports or driving licenses, new identification methods are required. In the past, intrinsic physical human properties have been shown on signatures, photographs or fingerprints. Currently, since the advent of the digital era, technology is still evolving in this area and

these characteristics are now based on mathematical mechanisms of recognition, capable of automatically making comparisons between two photographs, voices, or fingerprints (Figure 2.1) from several features or line drawings on the human features. Table 2.1 compares important parameters of wide-spreaded biometric techniques.[1]. Although these technologies work properly, human characteristics can be altered, damaged or even eliminated.

Technology	Cost Effective	Ease of Use	Special Hardware Requirement	Low Maintenance Costs
Fingerprints	Yes	Yes	Yes	No
Hand Geometry	No	Yes	Yes	No
Voice Verification	Yes	Yes	No	Yes
Iris Scanning	No	No	Yes	No
Facial Recognition	Yes	Yes	Yes	No

Table 2.1: Most common biometric technologies.

Table extracted from http://nextdoornerd.blogspot.com.es/2010/03/cheap-safe-biometric-technology-voice.html.

2.2 The Automatic Speaker Recognition

Voice messages typically provide more information than the message itself. Voice messages can provide us more information than a written one. For example, the speaker's sex, age, mood, or environmental conditions are factors that can be extracted from a voice message. This extra data, which is independent of the message, can bring us credibility of the message, the authenticity of the speaker, or many other message properties.

Thanks to the advancements of our electronic and software technology, two fields of research have received more attention in recent years due to the advances in computational speed: Speech recognition and speaker recognition. The first is the art of obtaining the transcript of the message, i.e., *what* the speaker is saying. In this process, the system should ignore age, voice, or the speaker's sex and has to focus on the message content. The second, speaker recognition, determines *who* is speaking, ignoring the message content. Now, age, dialect or voice tone have to be taken into account, because they can help to identify the speaker and they can help us with the recognition. In both cases, environmental factors, such as noise or the quality of the recordings, can adversely influence the outcome of the process and have to be taken into account.

There are two types of speaker recognition, depending on the delivered message, namely text-dependent and text-independent recognition. Speaker *identification* combines speech recognition with speaker recognition. A speaker identification system requires a speaker

[1]"http://nextdoornerd.blogspot.com/"

to pronounce a given text, with which it will have carried out the training process. For security applications, a user can log in using a voice password. Text-independent systems are able to identify a trained speaker from a different message compared to the testing message. Nowadays, most of the research focuses on speaker recognition for text-independent systems, which allows one a much broader scope. Being text-independent allows one to perform SR on surveillance or access control applications that do not require a concrete statement of text.

Theory of Speaker Verification

Speaker recognition has two main fields: *Speaker identification* and *speaker verification*. Speaker identification allows one to identify a concrete voice from a list of potential customers belonging to a database. It answers the question: "Who is speaking?". Speaker verification compares a voice message with a particular customer and accepts or rejects it based on the score. Now, the system can answer the question: "Did the speaker utter this segment?". In the following, we describe speaker verification in more detail.

Nowadays, the widely used technological bases for speaker verification are:

- Hidden Markov Models (HMM): Take each client as a Markov model.
- Gaussian Mixture Models (GMM): Use a Gaussian sum distribution to model each human.

A HMM is a statistical method which assumes that the system model is a Markov process of unknown parameters. The main goal is to determine the unknown parameters (or hidden, hence the name) of the chain from the observable parameters. The extracted parameters can be used to carry out subsequent analysis. It is very common to use HMM in pattern recognition applications [17, 15].

HMMs are commonly used at the level of phonemes[2], each using them to create an HMM. Essentially, a state machine is created and it uses audio frames to determine the probability of the next state. In text-dependent systems, a concrete phoneme is expected. Here, a comparison process between a speech segment and a model is more precise. However, Rabiner et al. [16] probed that GMM are more efficient than HMM in text-independent systems. GMM are a parametric probability density function represented as a weighted sum of Gaussian component densities [15]. GMM will be explained in detail below.

In addition to the above techniques, there are others such as neural networks or "support vector machines", that are still under investigation in the field of speaker recognition as shown in [19].

[2]Any of the distinct units of sound that distinguish one word from another. (http://www.wordreference.com/definition/phoneme)

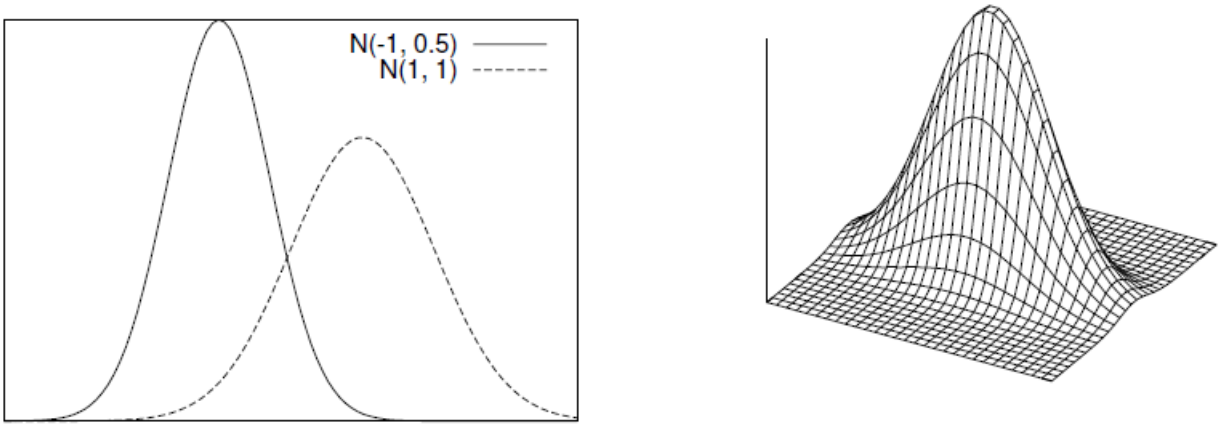

Figure 2.2: $\mathbb{R}$ and $\mathbb{R}^2$graph Gauss distributions. http://www.it.lut.fi/project/gmmbayes/

Gaussian distribution is one of the most common probability density function and models many natural, social or psychological events.

Gaussian Mixture Models

In the last decade, GMMs have dominated text-independent speaker recognition systems. This kind of approximation is appropriate for such tasks, because it does not consider the text said, but exploits the spectral voice features to discriminate between speakers. The use of GMMs in text-independent speaker recognition has been described first in the work of Reynolds and Rose [9]. Since then, systems based on GMMs appeared in many publications and numerous conference and are also used in the annual competitions of the National Institute of Standards and Technology (NIST).

In the field of statistics and probability, a Gauss distribution is the most common continue probability distribution[3] that can appear in real world. The graph of the associated probability density function is "bell"-shaped, and is known as Gauss bell, as shown in Figure 2.2.

Given a speech segment X and speaker S, the goal of speaker verification is to determine if speaker S generated X. This can be formalized as a hypothesis test between the following basic assumptions: H_0: X was pronounced by the speaker S. H_1: X was not pronounced by the speaker S.

$$\frac{P(X|H_o)}{P(X|H_1)} \begin{cases} \geq & \vartheta \; accept \; H_o \\ < & \vartheta \; reject \; H_o \end{cases}$$

where $P(X|H_i)$, $i = 0, 1$ is the probability of hypothesis H_i evaluated for a particular voice segment. ϑ is the decision threshold for accepting or rejecting H_0. Theoretically it should be 0, but in practical applications it is interesting to adjust the threshold to

[3]The probability distribution of a random variable is a function that assigns to each event random variable defined on the probability that the event occurs.

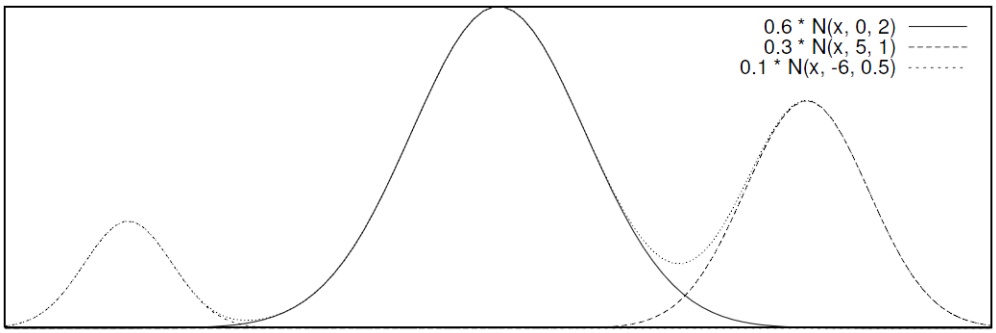

Figure 2.3: A Gaussian Mixture Model in $\mathbb{R}$ with three components.

Note that the sum of the weights w_i must be 1, or $p(x|\lambda)$ will not be a probability: $0.6 + 0.3 + 0.1 = 1$.

control the ratio between the probabilities of errors in the two possible directions of the decision. So, the main objective of a speaker recognition system is to use a given method to calculate both probabilities, $P(X|H_0)$ and $P(X|H_1)$.

The basic idea of the GMM method is to calculate the probability $P(X|H_i)$ as a finite sum of N distributions

$$p(x|\lambda) = \sum_{i=1}^{N} w_i p_i(x)$$

where $p_i(x)$ are individual distributions, and in the case of GMMs, Gaussian distributions. The weights w_i determine how much influence each distribution has. Each Gaussian distribution is given by:

$$p_i(x) = \frac{1}{(2\Pi)^{D/2}|\sum_i|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_i)^{'}(\sum_i)^{-1}(x-\mu_i)\right\}$$

where μ_i is a Dx1 dimension mean vector and $\sum_i$ is the DxD covariance matrix. Thus, each speaker is represented by a Gaussian mixture model denoted $\lambda = \{w_i, \mu_i, \sum_i\}$, i=1...M. Note that since $0 \leq p(x|\lambda), p_i(x) \geq 1$, the sum of the weights must be 1, or $p(x|\lambda)$ will not be a probability. It can easily be seen that a distribution with a large variance and a large weight will have a great influence on the mixture, as shown in Figure 2.3.

The application of GMMs to speaker verification requires at least two algorithms. First, some way of determining the means, variances and weights is needed, to make the GMM model a speaker's voice characteristics. Second, a method of "evaluating" audio segments using the model must be found, to verify the identity of the claimed speaker. Since the output of the GMM is a single probability, it has to be transformed into a "yes" or "no", so to accept or reject the hypothesis.

Expectation Maximization Method

The most common technique for training the models in speaker recognition is by using an Expectation Maximization (EM) algorithm [2]. The purpose of the EM algorithm is to redefine the mixture parameters to increase the likelihood of the generated *feature vector*[4] X, given model λ:

$$p(X|\lambda^{k+1}) \geq p(X|\lambda^{k})$$

In the area of speaker recognition, the EM-algorithm needs a given likelihood or an a posteriori probability to be a complete algorithm. Thus, in statistical terms, the EM-algorithm is one way of finding the maximum likelihood (ML) or maximum-a-posteriori probability (MAP) to estimate the data of a hypothesis. For the purpose of this master thesis are both MAP and ML variants of the EM-algorithm for GMMs needed. However, the derivation of the equations is very intricate, and the reader is referred to Bilmes [5] for the ML-version of the algorithm, and to the work provided by Reynolds et al. [8] for derivations of the MAP-version.

Speaker Verification Toolkits

In this subsection, we will presenta popular non-commercial software products used for Speaker Verification (SV). Note that there is a huge number of payment software toolkits for this purpose, but we will compare ALIZE / LIA_RAL only with open source software toolkits.

ALIZE/LIA_RAL is an open-source platform for biometric authentication with the GPL license, used specifically for speaker or speech recognition. This software was created in the Mistral project[5] at the University of Avignon, France. This project is divided into the ALIZE library and the high-level LIA_RAL toolkit [12].

ALIZE contains all needed functions to use Gaussian mixtures and speech processing.

The ALIZE project was created under supervision of the ELISA consortium[6]. It pretended to share and update all SR research projects in France in order to create a strong and free common speaker recognition software. According to the documentation [12], the main objectives of ALIZE are:

- To propose a toolkit making for faster development of new ideas, with a (proved) state-of-the-art level of performance.

[4] N-dimensional vector of numerical features that defines an object. In speech processing, "object" is referred to part of a recording, or speech segment.

[5] http://mistral.univ-avignon.fr/index_en.html

[6] http://elisa.ddl.ish-lyon.cnrs.fr/

- To encourage laboratories and research groups to evaluate new proposals using the toolkit and both standard databases and protocols like NIST SRE[7] ones.
- To help the understanding of speaker recognition algorithms (like EM training, MAP adaptation or Viterbi algorithm [8]), parameters and limits.
- To test new commercial speaker or speech recognition applications.
- To facilitate the knowledge transfer between the academic labs and between academic labs and companies.

LIA_RAL (Laboratoire Informatique d'Avignon Recognizer Architecture Library) is a speaker verification toolkit that uses all low-level functions of the ALIZE library. The LIA_RAL is commonly used for speaker recognition purposes, but it can be also used in the field of speech recognition.

LIA_RAL toolkit is divided into the packages *Spk_ Tools*, *Spk_ Utils*, and *Spk_ Det*. In this thesis, however, only *Spk_ Det* is used. It contains those components that are required by a biometric authentication system. The purpose of LIA_RAL is to wrap all algorithms in programs that can be run from a shell script.

Both, ALIZE and LIA_RAL, are implemented in C++and will be further explained in Chapter 4.

HTK (Hidden Markov Model Toolkit) is a well-known open-source toolkit used for speech and speaker recognition[9]. It was developed by the Speech Vision and Robotics Group of the Cambridge University Engineering Department (CUED) where it has been used to build CUED's large vocabulary speech recognition systems. HTK is a portable software for the manipulation of Hidden Markov Models used primary for speech recognition, although it has been used in many kinds of signal processing modules that include Markov models manipulation (speech synthesis, speaker recognition, image processing,...) as well.

MARF (Modular Audio Recognition Framework)[10] is a collection of speech, sound, text and language processing algorithms written in Java. A lot of applications have been created using MARF as main library, for example: *Math Testing Application, Language Identification Application, Artificial Neural Network Testing Application* or *Text-Independent Speaker Identification Application,* among others[11].

2.3 Embedded Systems

An embedded system is a device designed to perform one or more dedicated functions, often in a real-time computer system. Broadly speaking, we can define an embedded

[7] http://www.itl.nist.gov/iad/mig/tests/sre/

[8] The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the Viterbi path – that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.

[9] http://htk.eng.cam.ac.uk/

[10] http://marf.sourceforge.net/

[11] http://marf.sourceforge.net/#releases-dev

system like a computer with one or more integrated devices on the same board as seen in Figure 2.4. For example, an embedded system can be a PDA, a smart phone, or a built-in controller.

Nowadays, the industry is able to create physically smaller electronic systems that have the same technical features and functionality than larger systems years ago. To achieve price competitiveness and low dimensions of hardware devices, personal computers or laptops can be replaced by embedded systems with the same performance.

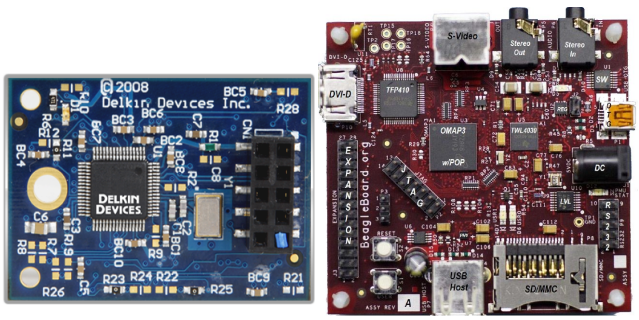

Figure 2.4: Examples of commercially available embedded devices.

Left: Delkin's embedded USB module (www.delkinoem.com).
Right: ARM Cortex-8 BeagleBoard (BeagleBoard.org) used during this master thesis.

Two of the main differences between an embedded system and a personal computer are the price and the size of the product, among the energy required, because embedded systems, as personal computers, are mass-produced by tens of thousands or millions of units benefiting from economies of scale. However, since an embedded system can be dedicated to concrete tasks, electrical and size design can be adjusted to this particular function. Embedded systems often use a relatively small processor and a small memory to reduce costs. On the other hand, a failure in one element can imply to repair or change the full board.

We can find an embedded system in a lot of products. Some of the most important applications of these systems are:

- In a factory, to control an assembly or production process: A machine that is responsible for a particular task contains numerous electronic and electrical systems for controlling motors or other mechanical device and all these factors have to be controlled by a micro processor, which often comes with a human-computer interface.
- In points of service in supermarkets or stores: Cash registers are becoming more complex, and they integrate numerical keypads, laser bar code readers, magnetic stripe or chip bank card readers, LCD alphanumeric display screens, etc. The embedded system in this case requires a lot of input and output connectors and robust features for continued work.

- In aircraft radars: Processing a signal from an aircraft radar requires high computing power, small dimensions and small weight. Also, it has to be robust enough for work on extreme conditions, like high or low temperature, low atmospheric pressure, vibrations, etc.
- In automotive technology: A car can include hundreds of microprocessors and micro controllers that control ignition, transmission, power steering, anti-lock brakes system (ABS), traction, etc. All this functionality is integrated in embedded devices.

In this master thesis, we will use the ARM Cortex BeagleBoard embedded system. The BeagleBoard is an open-source low-cost low-power device, based on the ARM Cortex-A8 chip. We depict such a BeagleBoard in Figure 2.4 (right). The board is designed and developed by Texas Instruments in association with Digi-Key.

The main purpose of the BeagleBoard project is to achieve a small, low-cost, expandable and powerful computer. BeagleBoard software is not a finished project. It is an "in development" project under the Creative Commons license, without official support except community forums or wiki pages[12].

A BeagleBoard has a lot of interfaces: USB, HDMI, Ethernet, MicroSD, JTAG or camera peripheral connections.The Linux system is the prefered operative system, so it can be used for many applications. In Chapter 5, we will discuss the BeagleBoard features in detail, and how we use the board in this master thesis.

2.4 Speaker Verification on Embedded Systems

As explained above, embedded systems offer a lot of opportunities. In the field of *signal processing*, embedded system also allow one to design compact and cheap systems for many purposes. In this master thesis, we will port the software of a speaker verification system to an embedded system software. It means, the embedded system can do the same work as performed with a laptop, like to process audio signal, to train clients and to test clients.

Speaker Verification technology in embedded systems will face the following issues and challenges:

1. An embedded system is commonly limited to a single microphone of the device, so the system has to be adapted to the environment. Some experiments used two or three microphones in the same embedded system, but it affects CPU consumption. In contrast, with laptops or personal computers, we can work with two or more microphones, as well as microphone arrays.
2. Models of speakers are typically stored on an extra device, like a smart card. Depending on the application of the system, the virtual size of this extra storage device can be an issue. In our case it will not be an issue, because the BeagleBoard has an SD card slot.

[12] http://elinux.org/BeagleBoard

3. Computational power and memory are typically lower. Embedded systems are often designed to be low-cost boards, so the micro processor power is not comparable with the power of a laptop. For example, with overclocking an ARM Cortex v8 (the CPU in BeagleBoard) we can get 1GHz clock frequenc. Meanwhile, a common midrange CPU in laptop or PC can work above 3 GHz easily. Moreover, BeagleBoard is a closed hardware package with 128 MB LPDDR RAM; in laptops or personal computers, we are working with 4 or 8 GB.

A lot of signal processing research projects try to port specific software to embedded systems because such systems offer a lot of opportunities. In recent years, several projects are focused on audio processing on embedded systems. Specifically, two projects are closely related to our thesis: the *TIESr Speech Recognition*[13] project and the *Watson Research Project on Speaker Verification in Embedded Environments* [3].

The TI Embedded Speech Recognizer (TIesr) is a fixed-point recognizer written in C++ and C. It is designed to balance resource usage, robustness to environment, and performance. It has a simple and easy-to-use user interface. This makes it an excellent recognizer for developing a wide variety of voice-enabled embedded applications. TIesr is a medium-sized phonetic-based recognizer that can accommodate a vocabulary of up to several hundred words. The words and phrases to be recognized can be dynamically changed by an application. The latest TIesr project release comes with general English language support, and tools to develop the support for other languages. TIesr has been tested on some Linux OS platforms, including the BeagleBoard and several OMAP EVM boards (35x, L138, Zoom), as well as platforms running Windows and Windows Mobile.

This project helps us to understand how a speech processing software has to be adapted to be able to run on a BeagleBoard. We also contacted Lorin Netsch, the TIesr project manager from Texas Instruments for questions about cross-compilations tools of the BeagleBoard.

In the Watson Research SV on the ES project, a low-resource, text-independent speaker verification (SV) system with an efficient voice model compression technique is described, including its implementation exclusively with integer arithmetic. The authors describe and discuss the individual algorithmic steps, the integer implementation issues and its error analysis. The performance of the system is evaluated with data collected via iPaq devices and the work discusses the impact of the model compression as well as integer approximation on the accuracy.

[13] https://gforge.ti.com/gf/project/tiesr/

Chapter 3

The Process of Speaker Verification

Before we explain the ALIZE / LIA_RAL software in more detail,we present the underlying theory of the speaker verification process. The steps presented in the following are commonly used in audio processing.

3.1 Overview

In this chapter, a revision of the *text-independent speaker verification* process is presented. All verification or identification processes follow the next steps:

1. Feature extraction is the first step in the Speaker Verification (SV) process. It consists of transforming a segment of speech data (wav, raw, etc.) into a parameter vector that is used in the next step. Feature extraction includes audio sampling, labeling, and feature normalization [6].

2. The second step is creating a Universal Background Model (UBM). According to the Reynolds definition in [10], "*A Universal Background Model (UBM) is a model used in a biometric verification system to represent general, person-independent feature characteristics to be compared against a model of person-specific feature characteristics when making an accept or reject decision*". A lot of input data is needed for this purpose.

3. In the training phase we create the gaussian mixture model (GMM) of the client. Training is performed using the Universal Background Model calculated in second step, and taking some input data from the speaker. The Universal Background model and the input data are combined using the Maximum A Porteriori adaptation (MAP).

4. In the testing phase we will gain a testing score of each client for each model. We will perform the feature extraction of the input speech segment and we will compare it with the speakers database.

5. The last step in the process decides if the system should accept the speaker as database member, or reject it. So, an acceptance threshold is defined depending

on system purpose (low false acceptance ratio, or low false rejection ratio), and the system takes a decision comparing the score obtained in testing phase with this threshold.

3.2 Signal Processing and Feature Extraction

The first step a biometric system performs is the extraction of discrete features from continuous data, and process them in a way that we can get relevant info from the person. According to [6], among others, in speech processing those steps are *audio sampling*, *labeling process*, *feature extraction*, and *feature normalization*.

Audio sampling Speech acquisition is realized with a microphone or headset telephone by converting a sound wave into an analog signal. The term sampling is referred to *the reduction of a continuous signal to a discrete signal. In signal processing, sampling is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal)*[1].

Labeling Labeling is used in text-dependent SV systems that are based on Hidden Markov Models. Basically, labeling means to assign a label or a name to every audio segment. In text-independent systems, it is interesting to process only the speech segments, because there is no available information in the silence segments. So it is usual to label "speech" segments and "silence" segments. Once the labeling is done, the next step can be done only in speech segments, saving memory and CPU processing power.

Feature extraction Normally, linear audio samples gained from the sampling step are not enough for the characterization of a speaker. Noise, or the change of voice leads to a different audio spectrum, so a more robust representation of such coefficients is needed. So, this step consists on extracting a real speaker feature vector from previous sampling values. These features are called Cepstral Coefficients (MFCC) and are coefficients for the representation of speech based on human hearing perception [18]. Moreover, the extraction is improved by using the Mel scale[2] to model the coefficients according to real human hearing perception. In Figure 3.1, we depict the block diagram of MFCC acquisition.

First, the audio signal is sampled at 8 kHz as explained in the sampling step, labeled and then windowed[3]. Normally, the signal is divided in segments of 25-30 ms, where the segments overlap each other 15 ms. The next step is to switch to the frequency domain with the help of the Fast Fourier Transform (FFT). Then the signal is filtered by using a filter bank of different frequencies to have a better resolution at low frequencies, which is comparable to the human hearing system. After the Mel filtering, we will obtain one coefficient for each filter, so, for example, using a filter bank of 40, we will obtain a vector of 40 coefficients for each frame. The logarithmic-spaced frequency band allows a better

[1] http://en.wikipedia.org/wiki/Sampling_(signal_processing)#Speech_sampling
[2] http://en.wikipedia.org/wiki/Mel_scale
[3] http://en.wikipedia.org/wiki/Window_function#Spectral_analysis

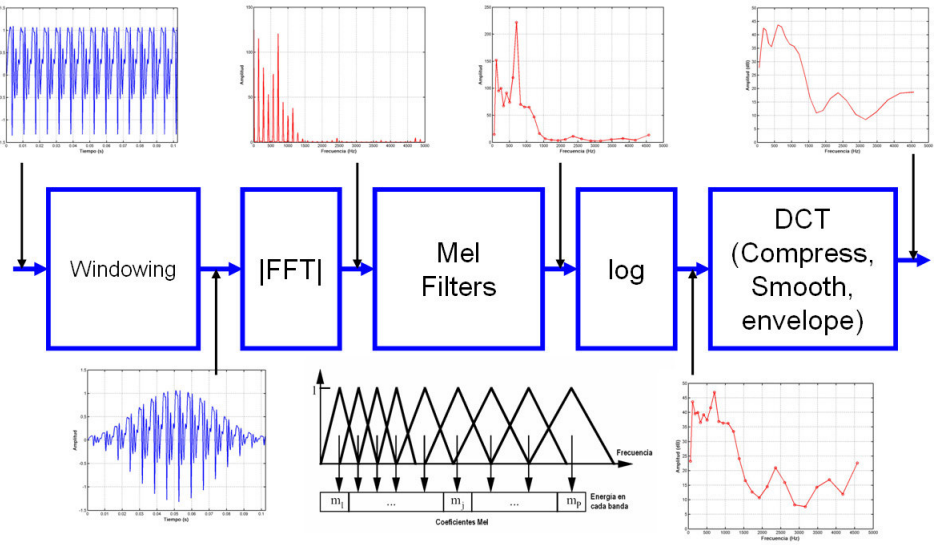

Figure 3.1: Block diagram of MFCC acquisition.

modeling of the human auditive response than a linear scale, so a log function and the Discrete Cosine Transform (DCT) are applied. This process implies more efficient data processing, for example, in audio compression.

Feature Normalization Real SV systems are sensitive to environmental noise. So, effects of the recording place or channel can be prejudicial for our verification process. To avoid this problem, or to reduce the effects of this noise, a feature normalization is done and basically consists on applying a feature update using the mean and the variance of each feature vector. For a feature x, the updated feature $x^{'}$ is defined as follows:

$$x' = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and the variance of the whole feature vector, respectively. It is possible to normalize only for the mean, but it is more usual and more robust to perform it on mean and variance.

3.3 Training Phase

The user models are created in the training phase. Therefore, it is necessary to create a background model first, and then adapt this model using extracted features of the client speech input [10].

Creating the Background Model The basic idea of *training* in SV based on GMM is not to create a unique client model using client feature vectors, because normally we

have not enough data of each client, and the model can become heavily *under-trained*[4]. Instead, a common solution is to train a general speaker model with utterances of a set of people, and then adapt this model to each user. The adaptation is done using user data. The name of this "general model"[5] is *Universal Background Model* (UBM). The model is the used as a common starting point for training models in general.

The UBM is trained by using the EM algorithm for 5 – 10 iterations [8]. The training data is collected from a special set of speakers with the only purpose of representing speech in general. If we want to create, for example, a gender-independent UBM, we have to use an equal distribution of male and female utterances.

Training speaker A user training is based on Maximum a Posteriori Adaptation (MAP). Like the EM algorithm, the MAP adaptation is a process of estimation in two steps. In the first step, the statistics of the training data to each UBM mixture are estimated. In the second step, these new statistics are combined with the statistical features of the UBM. Given the UBM and a feature vector $X = \{x_1, x_2, ..., x_T\}$ from a speaker, it is necessary to get a probabilistic alignment between the training vector and UBM mixtures. In Figure 3.2, a sketch of the alignment is shown.

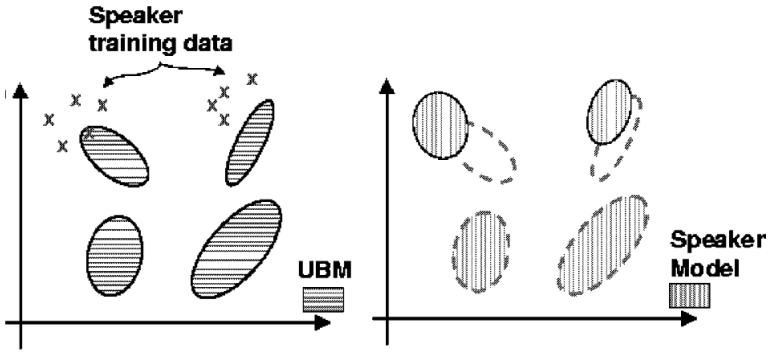

Figure 3.2: UBM model and client data alignment.

We use client data for adapting the UBM model. The speaker model is created aligning UBM with the input speech features.

It is important to know how this alignment is done to better understand the configuration of parameters of the MAP detector. This means for an i UBM mixture:

$$P(i|x_j) = \frac{w_i p_i(x_t)}{\sum_k^M w_k p_k(x_t)}$$

In the first step, using this value of $P(i|x_j)$ and x_t we can calculate the weight, the mean and the variance, which we will update in the second step. Weight, mean and variance are defined as follows:

[4]Not enough data collected for getting a robust client model

[5]It is named *general model* because the purpose of the model is to define speech in general.

$$N_i = \sum_j^T P(i|x_j,), \qquad E_i(x) = \frac{1}{\mathrm{N_i}} \sum_j^T P(i|x_j)x_j, \qquad E_i(x^2) = \frac{1}{\mathrm{N_i}} \sum_j^T P(i|x_j)x_j^2$$

Finally, the MAP algorithm consists of updates of these parameters. The parameter λ_i is defined as:

$$\lambda_i = \frac{\alpha^{\mathrm{w}}{}_{\mathrm{i}} \mathrm{N_i}}{n} + (1 - \alpha_i^w) w_i$$

So. an updated weight, mean and variance are defined as follows:

$$w_i^{'} = \frac{\lambda_i}{\sum_i^N \lambda_i}, \qquad \mu_i^{'} = \alpha_i^{\mu} E_i(x) + (1 - \alpha_i^{\mu})\mu_i, \qquad \sigma_i^{2'} = \alpha_i^{\sigma} E_i(x^2) + (1 - \alpha_i^{\sigma})(\sigma_i^2 + \mu_i^2)\sigma_i - \mu_i^{'2}$$

where N is the number of mixtures of distributions in the mixture. $\{\alpha^w, \alpha^\mu, \alpha^\sigma\}$are parameters that control balance between old and new weight, mean, and variance estimations. These coefficients are calculated using a relevance factor r^p that determines the relationship between previous and updated values.

$$\alpha_i^w = \frac{N_i}{N_i + r^p}$$

For example, using $r = 0 \rightarrow \alpha = 1$. It means $\mu_i^{'} = E_i(x)$ and no updates are performed.

3.4 Testing Phase

After the training phase, the speaker verification system is ready to be tested. We are able to extract features of a speech segment and create a speaker model using extracted features. Thus, the system should be able to verify speakers by using models and feature vectors. The first step in SV testing is to define *what the output of the testing shall be.*

The most common output used for testing is the *likelihood ratio* [13]:

$$\Lambda_M = \frac{p(H_M|x,\theta)}{p(\neg H_M|x,\theta)}$$

$$\log \Lambda_M = \log p(H_M|x,\theta) - \log p(\neg H_M|x,\theta)$$

where $p(\neg H_M|x,\theta)$ is defined as the probability of H_M not being the right hypothesis given x, i.e. M is not the speaker of x. The log ratio is commonly used as *log likelihood ratio* (LLR): $\log \Lambda_M$. For getting $p(\neg H_M|x,\theta)$ it is necessary to get information about all possible speakers in the world, except M. It is impossible to obtain all this information, but we can approximate this probability, too:

$$p(\neg H_M|x,\theta) = p(H_W|x,\theta)$$

where W is referred to the Universal Background Model.

3.5 Making the Decision

The last step of the verification process is to decide from which speaker the utterance comes from. In the SV context, the LLR threshold defined above is not enough to describe the performance of the system. Instead of finding a threshold, the Detection Error Tradeoff (DET) plot and the Equal Error Rate (EER) are used, as defined in Section 3.5.2.

Types of Output

We can group the SV output in four types: *True acceptance* (TA), *false acceptance* (FA), *true rejection* (TR), and *false rejection* (FR). TA occurs when a client or "true" speaker is accepted. FA occurs when an impostor is accepted. This is also called *false positive.* TR occurs when an impostor is rejected. FR occurs when a "true" speaker is rejected and is called *false negative.*

These rates are measured as a percentage. For measuring the quality of the system, typically only false rejections and false acceptances are used and visualized in a graph.

Detection Error Tradeoff and *Equal Error Rate*

The Detection Error Tradeoff plot was defined by Martin et al. [1]. In this graph, the two possible types of error are plotted: false acceptances and false rejections. The score distribution in the DET plot usually approximates two Gaussian distributions. In Figure 3.3[6] we show an example.

The Equal Error Rate (EER) is the standard numerical value to define the performance of a biometric verification system. It is defined as the point on the DET curve where the False Acceptance Rate and the False Rejection Rate have the same value.

It is important to note that a system with an EER of 8% does not mean necessarily that the system allows 8% of impostors[7]. One can choose his / her own threshold, taking into account the project needs.

Final Threshold

The final threshold is chosen after training, and can be set to the EER point as a starting point. Often, the threshold fixing is ignored, and instead the EER and DET-plot are used to show the overall system performance. In this master thesis, the threshold has been used just to obtain EER value and quality rates.

[6] http://rs2007.limsi.fr/index.php/Constrained_MLLR_for_Speaker_Recognition

[7] False user who claims to log in the SV system.

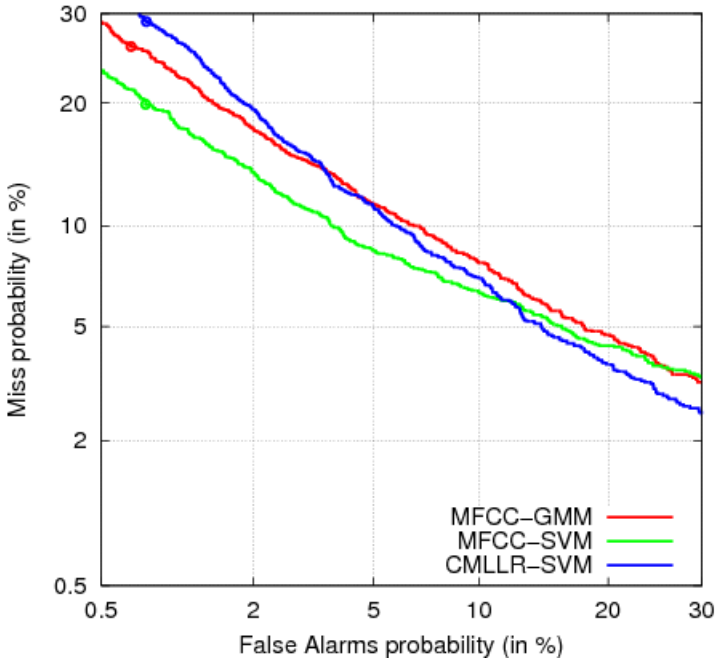

Figure 3.3: DET example of Speaker Recognition system.

In a DET curve false alarms are plotted against false rejections. In this example, EER is defined at 8%, in the case of MFCC-GMM.

Chapter 4

ALIZE Library and *LIA-RAL* Toolkit

As explained in previous chapters, in this master thesis we want to analyze the performance of ALIZE / LIA_RAL speaker verification library in an embedded system. In this chapter, and according to the documentation extracted from the official website (http://mistral.univ-avignon.fr/), we will describe the ALIZE and LIA_RAL speaker verification software [12] and we will explain its functionality relevant for this thesis in more detail. This toolkit has been designed to work with speech features, so the SPro4 has been used in this master thesis for this purpose. Its functionality and its most relevant features are explained in this chapter as well.

4.1 Overview

If we refer to the author, ALIZE is defined as *"an open-source platform (distributed under LGPL license) for biometric authentication"* [1]. This toolchain, defined as "low-level" library, contains all needed functions to the use of Gaussian mixtures. LIA_RAL is defined as a "high-level" toolkit, and is a "*set of tools to do all tasks required by a biometric authentication system*". In Figure 4.1 we can see a diagram of the main components of ALIZE and LIA_RAL distribution. The LIA_RAL package is based on ALIZE's low-level library and contains two task-specific sub-deliveries: LIA_SpkSeg, related to acoustic diarization and processing methods, and LIA_SpkDet, related to Speaker Verification tools. LIA_Utils contains a list of non-categorized tools that are included in LIA_RAL.

Both ALIZE and LIA_RAL SV software are implemented in C/C++, to allow multiplatform via the use of GNU Autotools[2]. In this chapter, we will explain each ALIZE and LIA_RAL function in relation to each stage described in Chapter 3. Neither ALIZE nor LIA_RAL do support feature extraction, so we will use SPro4 tool for this purpose. It will be explained in Section 4.4.

[1] http://mistral.univ-avignon.fr/index.html

[2] The Autotools consists of Autoconf, Automake, and Libtool toolkit to allow cross-compilation

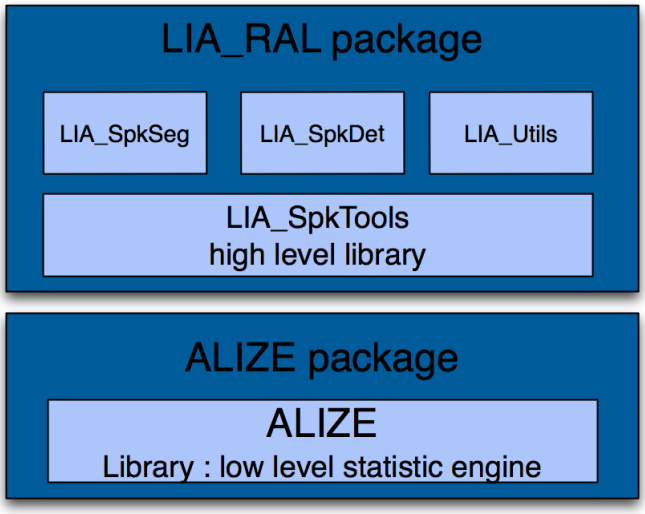

Figure 4.1: ALIZE and LIA_RAL distribution. The LIA_RAL package is based on ALIZE low-level library, and contains all the methods used in this master thesis.

Figure extracted from the Mistral project web page:
http://mistral.univ-avignon.fr/index.html.

4.2 *ALIZE* Library

Primarily, ALIZE is a framework of mathematics, statistics, and I/O functions on which our SV system is built. The main categories of functionality that exist in ALIZE are: [3]

- Command line and configuration file parser.
- File I/O management.
- Feature file reading and writing.
- Statistical functions.
- Gaussian distributions.
- Gaussian mixtures.
- Feature file labeling and segmentation.
- Line-based file I/O.
- Vectors and matrices management functions.
- Managers for storing named mixtures, feature files and segment.

[3]Information found at http://mistral.univ-avignon.fr/

Command line and configuration file parser. The ALIZE library uses configuration files and command line configuration. A user can define a configuration file to set some dynamic parameters, or define them on the command line. For example, defining a parameter *maxProb 1* in a configuration file is equivalent to type *–maxProb 1* in command line.

File I/O management. The library contains two classes for handling files. They are like a Java *Filestream*[4].

Feature file reading and writing. ALIZE supports some feature file formats for handling feature vectors. The ones mostly used are HTK, and SPro3/4. It is possible to use also *list files*, in order to concatenate feature files into a single stream.

Statistical functions. Basically, the library allows one to calculate mean and variance, as well to generate histograms. It also contains functions for the Expectation Maximization algorithm in general.

Gaussian distributions. ALIZE allows one to handle Gaussian full covariance matrices and diagonal variance matrix.

Gaussian mixtures. Almost all Gaussian distribution management code in ALIZE is targeted to Gaussian distributions. The ALIZE library is also designed to support other kinds of distributions. In this master thesis, this feature will be used to read input data.

Feature file labeling and segmentation. ALIZE has file labeling, with a module working in parallel with I/O. Segmentation will be used for optimization.

Line-based file I/O. All text files in ALIZE are distributed into lines and fields. One module of the library is targeted to reading and writing these fields.

Vectors and matrices management. This library contains basic algorithms to handle vectors and matrices.

Managers for storing named mixtures, feature files and segment. ALIZE contains a group of classes called *Servers* to handle the storage of named entities. I.e, all speaker models are stored in *mixtures server*, features are cached in *features server*, and labels are stored in *labels server*. It allows an automatic organization of files in memory. This automatic storage is essential for the data management, and lets ALIZE-dependent applications, such LIA_RAL toolkit, to work in an optimized and organized way.

[4]http://www.cs.williams.edu/javastructures/doc/structure/structure/FileStream.html

4.3 *LIA_RAL* Toolkit

The LIA_RAL toolkit is the software that wraps all the ALIZE algorithms in programs that can be run from a shell script. In the following, we describe the main LIA_RAL tools used in this master thesis.

Normalize Features

The NormFeat program reads a feature file and normalizes it. It is possible to normalize the extracted features using mean / variance normalization or Gaussianization. The first mode simply accumulates mean or variance, and then normalizes audio features using them. The second one performs feature warping [11] to get the Gaussian distribution from a histogram of feature vectors. In this master thesis, only the mean normalization will be used.

Energy Detector

The EnergyDetector tool reads a feature file and segments it based on speech energy. The program will discard segments with lower energy level than a threshold. Using the easiest way, *meanStd*, we train a GMM on the energy component and we find the distribution with the largest weight, w_i. So, we compute the threshold as:

$$\tau = \mu_i - \alpha\sigma_i$$

where τ is the threshold and (μ_i, σ_i) are mean and variance of top distribution. α is an empirical constant between 0 and 1 that can be defined. Normally (and also in this master thesis), only the mean is used to get the threshold ($\alpha = 0$).

Training Background Model

The creation of the Universal Background Model is done with the TrainWorld tool. The main purpose of this tool is to create a single GMM using a large amount of data. Firstly, the tool creates an equal mean and variance distribution, and it is adapted iteratively using the Expectation Maximization method. We can configure several parameters of this program. The most relevant are:

- Number of distributions: This will determine the number of Gaussian distributions that the background model will have. This number will set the distributions of future client models.
- Number of training iterations.
- Features selection: We can choose the amount of data for creating the UBM. This selection is based on probability.

Training Target

The purpose of the TrainTarget tool is to adapt the background model to a speaker model, using feature vectors and the MAP criterion. Like in the UBM training, not all feature vectors are necessary for doing the adaptation, so we can also fix the percentage of feature segments to use.

An input file for TrainTarget is a list with lines like:

```
SPEAKER1 speakers/spk1_001 speakers/spk1_002
SPEAKER2 speakers/spk2_001 speakers/spk2_002
SPEAKER3 speakers/spk3_001 speakers/spk3_002
SPEAKER4 speakers/spk4_001 speakers/spk4_002
```

In the first column the name of each speaker is declared. In the next columns, we define the utterances that we want to use to create the model. Normally, we will use two or three utterances. The more utterances we use, the better the model will be, because we will have more information from the speaker.

As explained in the Section 3.3, we will use a relevance factor r^p for calculating the family of variables α_i. The TrainTarget application allows us to define this relevance factor, or fix α_i, using the "MapOccDep" or the "MapConst2" mode respectively. In this master thesis, the "MapOccDep" mode will be used. It computes α_i as a linear combination of its value in the world model and its value obtained by an EM algorithm on the data.

Computing Test and Scoring

The ComputeTest tool uses the background model, the speaker models and a number of feature files, and calculates a score. It uses the standard log-likelihood ratio (LLR) measure for scores. An exemplary output file looks like this:

```
F spkFAML.MAP 1 FAML_Sr3 2.26765
F spkFDHH.MAP 0 FAML_Sr3 -1.7321
F spkFEAB.MAP 0 FAML_Sr3 -0.5291
F spkFHRO.MAP 0 FAML_Sr3 -2.1344
```

Here, the first column denotes F (female) or M (male) and the second and the fourth column are referred to the name of the model and audio file, respectively. In the last column the score is shown. We can use the third column to discriminate positive scores (1) or negative scores (0).

The most important parameter that can be set is the number of top distributions to be used in the result. We can greatly reduce the computational time growing up this number. If we evaluate, for example, only five of 512 or 1024 distributions, the performance can be improved. It will be a key point of improvement in the embedded software in this thesis.

Taking a Decision

The last step is the decision making. This process consists of comparing the *likelihood* resulting from the comparison between the claimed speaker model and the *incoming speech signal* with a decision threshold. If the likelihood is higher than the threshold, the claimed speaker will be accepted, else rejected.

As commented in Subsection 3.5.3, a threshold should be defined according to our needs. This master thesis evaluates the ALIZE / LIA_RAL software, so the threshold was not defined. We will use a collection of possibe threshold values for getting FR and FA curves, as well as equal error rate (see Section 3.5).

4.4 *SPro4* Toolkit Features Extraction

Because the LIA / LIA_RAL software does not allow audio feature extraction, we have to use another tool to extract feature vectors. SPro4 is a "*free speech signal processing toolkit which provides run-time commands implementing standard feature extraction algorithms for speech related applications and a C library to implement new algorithms and to use SPro files within your own programs*"[5].

Basically, the *SPro4* tool reads in an input audio file, processes it and extracts the feature vector. The output feature vector will be the starting point for the LIA_RAL system. The most common SPro4 parameters that can be set are:

- Format: input file format (wav, raw, sphere, etc).
- Buffer size: sets the input and output buffer size. The smaller the input buffer size, the more disk access is needed and, therefore, the slower the program is.
- Sample: input waveform sample rate. Commonly 8kHz.
- Normalization: system allows mean and variance normalization.
- Derivatives: SPro shows first and second order derivatives.

An exemplary feature vector is stored as shown below: :

```
-0.20995 3.858743 -0.08501 2.455754 0.085920
1.968982 0.090583 2.134615 -0.15941 2.456415
1.715665 -1.39294 0.938561 0.489811 0.806973
0.775145 -2.62302 -0.06644 -4.18761 -0.87814
(...)
```

In this master thesis, the Spro4 tool will be used as shown below for all audio files and all tests:

[5] http://www.irisa.fr/metiss/guig/spro/

```
./sfbcep -F wave -l 20 -d 10 -w Hamming -p 16 -e -D -k 0 audio/audiofile.wav features/
    audiofile.tmp.prm
```

- Format: Wave.
- Buffer size: 20 ms.
- Shift: 10 ms.
- Window: Hamming.
- Output cepstral coefficients: 16.
- Add log-energy and first order derivatives to the feature vector.
- Set the pre-emphasis coefficient to 0.

Chapter 5

Open Embedded as Operating System

The main goal of this master thesis is to adapt and configure a SV software, and run it on an embedded system. In this chapter, we describe the embedded system environment set-up and the necessary cross-compiling tools.

5.1 Embedded System Environment

The BeagleBoard is a computer development open source project. It is presented in a single device and it is based on the Texas Instruments OMAP3530 chip, as shown in Figure 5.1. The processor is an ARM Cortex-A8 core with a digital signal processing DSP core. The motivation of the project is to design a computer with low-power consumption, low-cost, compact size and expandable device. The project is in continuous development under the Creative Commons license, as specified on the first pages of the BeagleBoard user manual[1]. Moreover, the whole project is supported and commercialized by Texas Instruments and Digi-Key.

In this master thesis, revision B5 of the board will be used. We present the most significant features of it below. For more details of the board, please consult the manual.

In addition to the device, in this project we used a LCD monitor, a mouse, a keyboard and a USB Hub. The board needs an auxiliary energy supply. In Figure 5.2 we can see the laboratory setup.

The procedure in the device is similar to the procedure in a desktop PC under Linux environment. One needs at least one SD card to store Linux and its boot loaders[2]. In the next section, this procedure will be explained.

[1] http://BeagleBoard.org/static/BBSRM_latest.pdf

[2] Booting is a process that starts operating systems when the user turns on a computer system.

Figure 5.1: An exemplary BeagleBoard B5, used in this master thesis.

http://www.liquidware.com/

5.2 Angstrom as Operating System

As commented above, an advantage of the BeagleBoard is that it can be used as a computer using a Linux distribution. Depending on the application that one wants to use with the board, the decision of the operating system will vary.

In this master thesis, we use the Angstrom distribution[3]. This distribution was created using OpenEmbedded[4] tools in order to get a powerful operating system using minimal data storage. There are several pre-compiled Angstrom distributions, but it allows one to create a personalized operating system image with an online builder. The builder, called Narcissus[5], allows one to choose the desired characteristics for maximum features adjust as needed. The Narcissus tool allows one to create virtual images for several embedded systems. One can adapt the OS (command line interface, Desktop environment or based environment for PDA style devices) and add additional packages depending on the needs.

We choose a minimal configuration, without a graphical interface, and no additional packages. This is necessary to optimize the processing time of our application. Note that

[3]http://www.angstrom-distribution.org
[4]http://www.openembedded.org/wiki/Main_Page
[5]http://narcissus.angstrom-distribution.org/

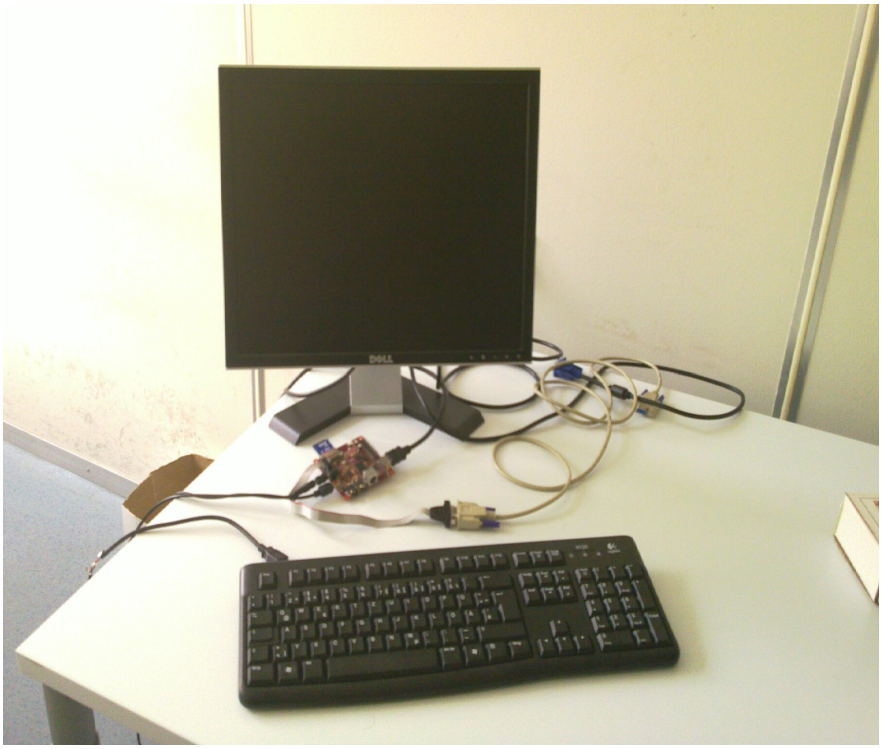

Figure 5.2: Using the BeagleBoard.

In the figure we can see the BeagleBoard with all peripherals used in this master thesis: keyboard, screen and SD card.

our final speaker verification version is an offline version. If we want to verify online, an input microphone and audio drivers would be required. However, such drivers can be cross-compiled[6] for ARM, too.

In this master thesis, we use a 2 GB SD card. It is possible that other versions of the operating system require larger cards for its correct functionality. We are using a minimal version, so 2 GB are enough for our work.

At first, the SD card formatting and partitioning is necessary. The SD card is divided into two parts: The OS and the system files needed for the boot process. Formatting and partition has been done using a script. This script is presented in Annex A.

5.3 Cross-compilation of the Speaker Verification System

As described in previous chapters, the speaker verification system proposed in this thesis involves the usage of three main applications: The SPro4 tool, for extraction of acoustic features of audio files, the ALIZE library, which includes all the signal processing functions, and the LIA_RAL toolkit, to train and test speakers using the functions of the

[6]See section 5.3

ALIZE library. The compilation and execution of these applications, however, is primarily intended for desktop computers or laptops. The porting process of the software to an embedded system requires a specific compilation according to the target system architecture. In this master thesis, we have to compile each program, as well as library, to run on an ARM architecture machine. However, we use a pre-configured OS, and the Angstrom version presented in Section 5.2 contains compiled libraries for the ARM architecture.

Another option is to compile directly on the BeagleBoard. This means that one copies the source code on the SD card, and compiles it using the GCC compiler of the OS of the BeagleBoard. This compiler, however, is not contained in the minimalist version used, and thus it is preferably to carry out cross-compiling on the host machine.

For the cross-compilation we will use the *Sourgery G++ GNU*-based toolchain. The cross compiler can be installed in any Linux distribution. For more information, we refer to the user manual[7].

In the following subsections, we will explain the process of cross-compiling used for each of the three main applications of our master thesis, using the GNU Build System.

The GNU Build System

It can be difficult to port a software program: the C compiler differs from system to system; certain library functions are missing on some systems; header files may have different names. One way to handle this is to write conditional code, with code blocks selected by means of preprocessor directives (#ifdef); but because of the wide variety of build environments this approach can become unmanageable. The GNU build system is designed to address this problem.

The GNU Build System, also known as the *Autotools*, is a toolkit developed by the GNU project. These tools are designed to help creating portable source code packages for various Unix systems. The GNU build system is part of the GNU toolchain and is widely used to develop open source software. Although the tools contained in the GNU build system are under the General Public License (GPL)[8], there are not restrictions to create private software using this toolchain.

The GNU build system includes GNU Autoconf, Automake and Libtool utilities. Other tools used are often the program GNU make, GNU get text, pkg-config and the GNU Compiler Collection (GCC).

GNU Autoconf is used to adapt the differences between different distributions of Unix. For example, some Unix systems may have features that do not exist or do not work on other systems. Autoconf can detect the problem and find the way to fix it. The output of Autoconf is a script called *configure*. Autoheader is also included in Autoconf, and is the tool used to manage the C header files.

[7] https://sourcery.mentor.com/GNUToolchain/doc7793/getting-started.pdf
[8] http://en.wikipedia.org/wiki/GNU_General_Public_License

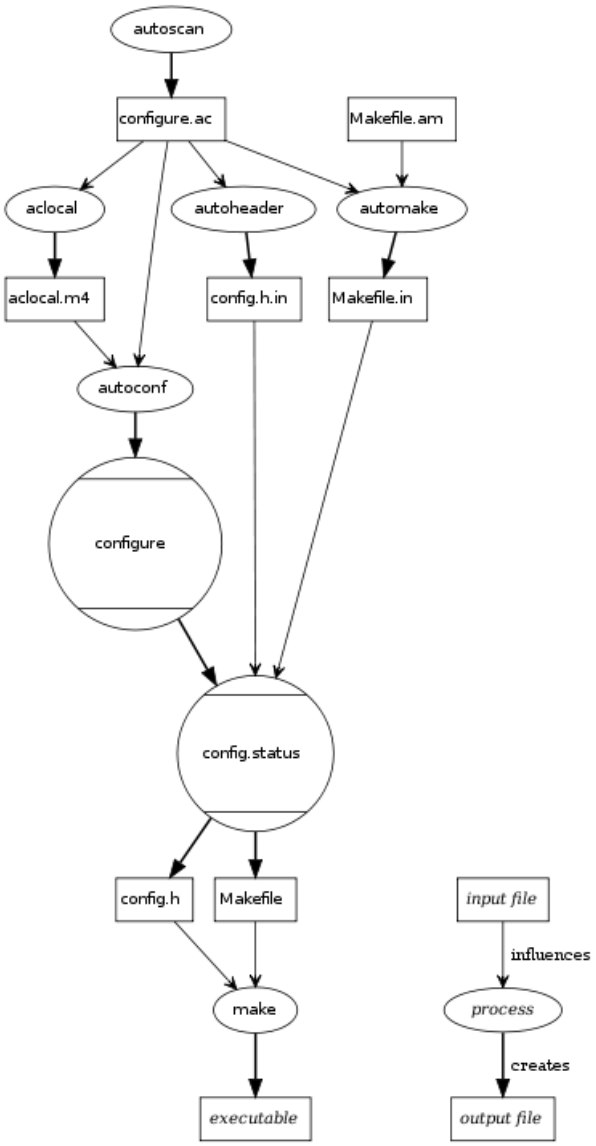

Figure 5.3: Main distribution of GNU Autotools.

Extracted from http://en.wikipedia.org/wiki/GNU_build_system.

Autoconf processes the configure.in and configure.ac files. When the tool is running the configuration script, Autoconf can also process the Makefile.in file to produce an output *Makefile*. In Figure 5.3 we can see file dependencies and distribution of Autotools.

GNU Automake is used to create portable Makefile files that are processing later using make tool. The tool uses Makefile.am as input file and transforms it into a Makefile.in. The Makefile.am is used by autoconf to generate the final Makefile.

Gnu Libtool can create static and dynamic libraries for various Unix OS. Libtool abstracts the process of creating libraries and simplify the differences between systems.

GNU Configuration for BeagleBoard

To make the cross-compilation one has to choose the appropriate settings in the autoconf of each program: ALIZE library, LIA_RAL toolkit and SPro 4.0 features extractor. We used the Open Source GNU cross-compilation tools supplied by Codesourcery[9] needed in cross-compilation tools of GNU autoconf. This cross-compiler is basically used as standard g++ compiler for an ARM target.

In this master thesis, the i386 microprocessor is used as build machine, so the command line use for the cross-compilation is:

```
./configure --build=i386 --target=arm-linux-gnueabi CXX=arm-linux-gnueabi-g++ CC=arm-
    linux-gnueabi-gcc CXXFLAGS=-march=none
```

where:

- –build is referred to host machine. We can use both –build and –host options
- –target is used to identify the target machine architecture.
- CXX and CC are referred to g++ and gcc compilers used.
- CXXFLAGS are the common flags that we use in a common compilation process.

[9] http://www.codesourcery.com/sgpp/lite/arm

Chapter 6

Configuration of *ALIZE / LIA_RAL*

The ALIZE / LIA_RAL package is used in many applications. In this chapter, we present a "how-to-use" in a shell environment, the compilation process and the configuration parameters used in this master thesis.

6.1 Shell-script Compilation

Using the LIA_RAL toolkit requires one to study the application (see Section 4.3), and its configuration parameters. Below, we will describe each application and we will discuss the best way to define and use each application parameter. The Spro4 feature extractor is necessary to process all files. So, a concatenation script is used. For more information, read Section 4.4 or visit the website[1].

Normalization and Energy Detection

The *NormFeat* tool is used to normalize the feature file. It is used as follows:

```
./LIA_RAL/LIA_SpkDet/NormFeat/NormFeat --config cfg/NormFeat.cfg --inputFeatureFilename
    ./lst/all.lst
```

The *EnergyDetector* tool is used for silence removing and labeling speaker features before processing them.

```
./LIA_RAL/LIA_SpkDet/EnergyDetector/EnergyDetector --config cfg/EnergyDetector.cfg --
    inputFeatureFilename ./lst/all.lst
```

Next, one has to re-normalize the label files. This is done with the *NormFeat* tool again, with another configuration file:

```
./LIA_RAL/LIA_SpkDet/NormFeat/NormFeat --config cfg/NormFeat_energy.cfg --
    inputFeatureFilename ./lst/all.lst
```

The script processFeatures.sh includes the three steps presented above.

[1] http://www.irisa.fr/metiss/guig/spro/spro-4.0.1/spro_4.html

Universal Background Model creation

Here, the Universal Background Model (UBM) is created, using all feature data that we have normalized before. TrainWorld is used:

```
LIA_RAL/LIA_SpkDet/TrainWorld/TrainWorld --inputFeatureFilename lists/UBM.lst --config
    cfg/TrainWorld.cfg
```

Client enrollment

Speakers that will be accepted by the system have to be trained now. It is necessary to use a list file with a specific .ndx file. This file contains the name of the speaker model, and the feature files used to create it. An exemplary of the speaker10 .ndx file is shown.

```
spk10.MAP 10_Sa 10_Sb 10_Sc 10_Sd 10_Se 10_Sf 10_Sg
```

The *TrainTarget* tool creates the speaker model using the background data and the speaker feature dataset specified in the .ndx file:

```
./LIA_RAL/LIA_SpkDet/TrainTarget/TrainTarget --config /cfg/trainTarget.cfg --targetIdList
    /lists/mixture.ndx --inputWorldFilename wld
```

To make it easier, the script trainSpeaker.sh performs the feature extraction, feature processing, ndx file creating and client enrollment one after another.

Testing of the speaker

This is the last step of the SV process. A result of the target against all clients on database is obtained, and we will get a score for each client enrolled. Using this score, we can take a decision. Here we use the *ComputeTest*:

```
./LIA_RAL/LIA_SpkDet/ComputeTest/ComputeTest --config cfg/ComputeTest.cfg --ndxFilename
    ./ndx/mixture.ndx --worldModelFilename wld --outputFilename res/result.res
```

The *ComputeTest* file saves the results in an output file with the result of each comparison. For a simple comparison with only two enrolled clients and one testing sample, the output file is shown as follows:

```
client1.MAP 1 sample1 5.42739
client2.MAP 0 sample1 -2.67043
```

where the columns represent the enrolled client, the decision taken, the sample name and the score obtained, respectively. In the example, we are comparing a speaker sample with his enrolled model, and with another speaker model enrolled in the system. The decision taken will be 1 (accepted) or 0 (denied) depending on the threshold that we are using. Anyway, this factor will be irrelevant in this mater thesis, as the score gives us all the information that we need to get conclusions.

For a better understanding, we present a more complex example. Now, we will compare 6 different samples of the same speaker (named FAML) against the entire database, with 4 clients enrolled. The enrolled speakers are named FAML, MREM, FEAB, MKBP. The next output file is obtained:

```
spkFAML.MAP 1 FAML_Sa 5.42739
spkFAML.MAP 1 FAML_Sb 6.73677
spkFAML.MAP 1 FAML_Sc 6.21802
spkFAML.MAP 1 FAML_Sd 5.50474
spkFAML.MAP 1 FAML_Se 6.60182
spkFAML.MAP 1 FAML_Sf 5.81713
spkFAML.MAP 1 FAML_Sg 5.6254
spkMREM.MAP 0 FAML_Sa -2.80681
spkMREM.MAP 0 FAML_Sb -2.83226
spkMREM.MAP 0 FAML_Sc -1.74641
spkMREM.MAP 0 FAML_Sd -2.20437
spkMREM.MAP 0 FAML_Se -1.96162
spkMREM.MAP 0 FAML_Sf -1.75959
spkMREM.MAP 0 FAML_Sg -1.20058
spkFEAB.MAP 1 FAML_Sa 1.17351
spkFEAB.MAP 1 FAML_Sb 0.908691
spkFEAB.MAP 1 FAML_Sc 1.19148
spkFEAB.MAP 1 FAML_Sd 0.703976
spkFEAB.MAP 1 FAML_Se 0.462561
spkFEAB.MAP 1 FAML_Sf 0.448675
spkFEAB.MAP 1 FAML_Sg 0.789436
spkMKBP.MAP 0 FAML_Sa -1.24249
spkMKBP.MAP 0 FAML_Sb -1.0661
spkMKBP.MAP 0 FAML_Sc -0.393202
spkMKBP.MAP 0 FAML_Sd -0.483468
spkMKBP.MAP 0 FAML_Se -0.959548
spkMKBP.MAP 0 FAML_Sf -1.19945
spkMKBP.MAP 0 FAML_Sg -1.03321
```

The speaker detector is obtaining high scores for the right speaker, and low scores for wrong speakers.

The script testSpeaker.sh launches the entire testing process. It includes feature extraction of test audio files, features processing of those files, and tests them against client Database.

6.2 Configuration Parameters

Every tool that we use for the speaker verification process allows one to configure several parameters. These parameters can be fixed on the command line, but it is easier to use dedicated configuration files .cfg.

Most of the parameters used in this master thesis are common for all the tools, but some of them are specific for each application. It is important to get a good configuration to

optimize the time and the quality of the verification process. In the following, we will present the description and possible values of the common parameters used in this master thesis. Later, the values of the specific parameters for each application are presented, too.

Typical configuration parameters

In Table 6.1 we present common parameters used in all the tools, and their calibrated values for this master thesis. These parameters have to be defined in all configuration files.

Parameter	***Description***	***Value***
frameLength	When working in segmental mode, specify the ratio between the time unit found in the label files and the frame-based time unit.	0.01
distribType	Specify the type of distribution.	GD
vectSize	Specify the size of vectors in the feature files.	34
loadFeatureFIleFormat	Specify the format of feature files.	SPRO4
sampleRate	Feature sample rate.	100
mixtureDistribCount	Specify the number of Gaussian distributions in the mixture.	64
maxLLk	Specify maximum likelihood values.	200
minLLk	Specify minimum likelihood values.	-200

Table 6.2: Common configuration parameters.

Specific configuration parameters

Here we show the most important specific parameters for each tool used in the speaker detection, as well as the values used in this master thesis. The parameters for normalization, training and testing are shown in Table 6.2, 6.3 and 6.4, respectively.

Parameter	*Description*	*Value*
saveFeatFileSpro3DataKind	Specifies the type feature data used in feature extraction using SPRO.	FBANK
labelSelectedFrames	Specify the label name to work with.	speech
nbTrainIt	Number of EM iterations to estimate energy distribution.	8
varianceFlooring	Variance control parameters.	0.5
varianceCeiling	Variance control parameters.	10
featureServerMask	In this case, energy detection is done on a single dimension of the input vectors.	16
segmentalMode	When working in segmental mode, stats and normalization are computed for each segment independently.	true

Table 6.3: Normalization configuration parameters.

Parameter	*Description*	*Value*
MAPAlgo	Specify the adaptation method to use: MAPOccDep or MAP-Const.	MAPOccDep
MAPRegFactorMean	Parameter used by the MAPOccDep adaptation technique.	14
meanAdapt	If true, training will use mean adaptation.	true
alpha	Parameter used by the MAPAlgo adaptation technique.	0.75
nbTrainIt	Number of EM iterations.	20
nbTrainFinalIt	Number of final EM iterations.	20
inputWorldFilename	Name of Universal Background Model.	wld

Table 6.4: Training configuration parameters.

6.3 Performance Optimization

As explained in previous chapters, a lot of parameters of the ALIZE / LIA_RAL toolkit can be used to improve the functionality of the software. This software was developed as a speech processing toolkit, not just as a speaker verification system, and we have to adjust the parameters for this usage correctly. Moreover, an embedded system usage has to be taken into account, and it is interesting to find a compromise between detection

Parameter	*Description*	*Value*
inputWorldFilename	Name of saved world model.	wld
ndxFilename	Name of .ndx file.	not fixed
topDistribsCount	Number of Gaussians used to compute the score.	10
computeLLKTopDistribs	Complete or partial computation when computing LLK with "top" distributions.	COMPLETE

Table 6.5: Testing configuration parameters.

rate and performance in the BeagleBoard. Thus, depending on the results obtained in the next chapter, the maximum optimized configuration files will be shown in Annex B.

Chapter 7

Tests and Results

In this chapter, we present the results of the tests with the *ALIZE / LIA_RAL* on the BeagleBoard. Apart from describing the setup test and presenting the SV quality performance of the software, the text below is a comparison of a laptop and an embedded platform, so the performance measures and memory requirements differences will be presented, too.

7.1 The *ELSDSR* Database

ELSDSR corpus design is a joint effort of the faculty, PhD students and Master students from the department of Informatics and mathematical modeling, Technical University of Denmark (DTU). The speech language is English, and it is spoken by 20 Danes, one Icelander and one Canadian. It was distributed among DTU Informatics as a first preliminary version in 2004.

The sampling frequency is chosen with 16 kHz having a bit rate of 16, and the audio format is WAV. Part of the text, which is suggested as training subdivision, was made with the attempt to capture all the possible pronunciation of English language including the vowels, consonants and diphthongs. The training text is the same for every speaker in the database. Regarding the suggested test subdivision, we will work with a forty-four sentences set (two sentences for each speaker).

For the training set, 161 (7*23) utterances were recorded; and for the test set, 46 (2*23) utterances were provided. On average, the duration for reading the training data is: 78.6s for male; 88.3s for female; and 83s for all. The duration for reading test data, on average, is: 16.1s (male); 19.6s (female); and 17.6s (for all)[1].

The ELSDSR database has enough samples to get robust conclusions. Using huge databases, with more information requires much more processing time for train models and collect data and does not supply any extra information to this thesis.

[1] http://www2.imm.dtu.dk/~lfen/elsdsr/

7.2 Setting up the Test Environment

As mentioned before, ELSDSR Database consists of 23 speakers: 13 males and 10 females. Every speaker entry consists of 7 utterances for training and 2 utterances for testing. The more clients are in the database, the more realistic the test will be, so the entire database will be used for enrollment in our thesis.

Theoretically, clients[2] and impostors[3] must be different. Due to the limited amount of data (we have just 23 people samples), all speakers are used as clients as well as impostors, so:

- Number of clients: 23
- Number of impostors: 23

Creating the universal background speech model requires a large amount of data, since it must include all possible speech data. The more data the UBM contains, the more robust the model will be. Typically, SV tests use very large databases to create these models. In this test, however, we will use all available training data to create the UBM, having 161 utterances.

This database is quite small, and it is not enough to get successful and realistic speaker verification results. Nevertheless, the main purpose of this master thesis is to compare the performance of the speaker verification system in both platforms, not to obtain accurate results of SV.

7.3 Performing the Test

After the preparation of the test environment, one can proceed with the testing. This part applies the numerous steps described in Chapter 6 using a database as ELSDSR. We just want to compare the verification results, so the training stage will be performed always in the laptop. Thus, the fetaures extraction, the Universal Background Model creation and the client enrollement will not be done on the BeagleBoard.

Once the UBM and the client models are created, they must be copied on the BeagleBoard SD card. After using the cross-compilation tools described in Subsection 5.3 we are able to launch the ComputeTest LIA/RAL tool on both laptop and BeagleBoard systems. In order to get results, the ComputeTest tool is launched using trained UBM and client models in both laptop and BeagleBoard platforms. It is important to use the same parameters in both platforms, in order to compare the requirements with the same configuration in both sides.

The relevance factor is usually considered to conclude the results of a speaker verification system. However, in this master thesis we do not want to evaluate the quality of the SV,

[2]Enrolled user in the database

[3]Unregistered user in the database, who tries to log in.

but the differences between running it on a laptop or and an embedded system. We use the same source code in both platforms, so Equal Error Rate and speaker detection quality should be exactly the same, as they are not memory and processor speed dependent.

7.4 Results

The two main factors that we need to measure are the *execution time* and the *memory*. The first one will describe the difference between the laptop and the BeagleBoard performance. The second one presents the viability of using this software on the embedded system, since we have a limited RAM memory.

We will present these ratios in both platforms with different parameter configurations.

EER optimization

First, we measure the Equal Error Rate optimization taking into account the acoustic features and the amount of data of our database. Before porting the ALIZE / LIA_RAL toolkit to the embedded system, one has to determine the number of Gaussian distributions in the mixture that optimizes the EER. As mentioned, the EER value is the intersection point between False Rejections and False Acceptances curves, shown in Figure 7.1, and Figure 7.2.

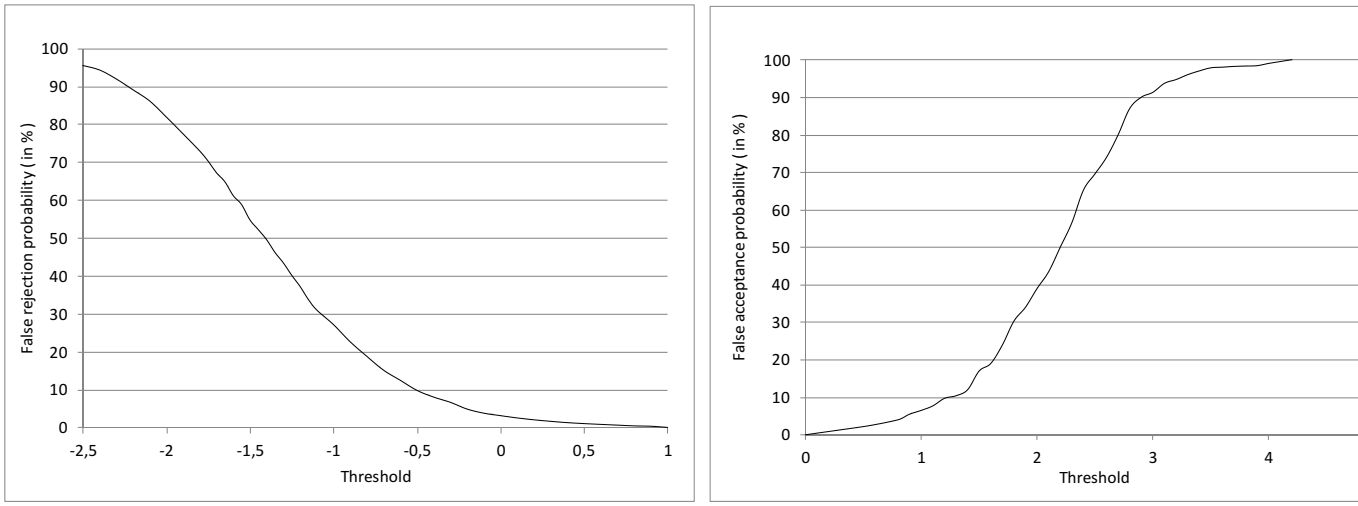

Figure 7.1: False Rejections (left) and False Acceptances (right) curves for 1024 Gaussians.

Data extracted from the development environment performed in this Master Thesis.

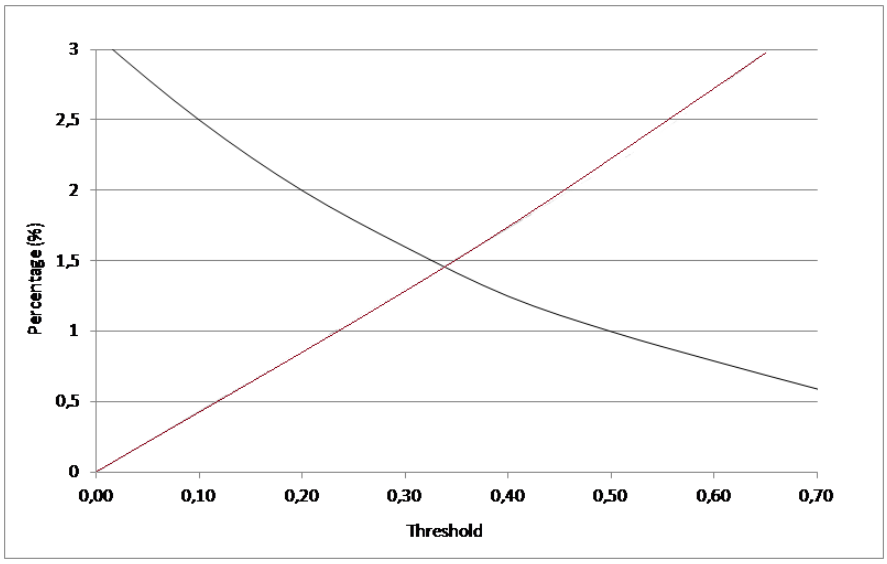

Figure 7.2: Equal Error Rate point in the 1042 GMM test.

ERR is the intersection point of false acceptances (red) and false rejections (black).

This intersection point can vary depending on a number of factors, so Table 7.1 presents the results in terms of accuracy varying Gaussian mixtures and iterations number. The results come from evaluating the database using the setting up described in section 7.2 and testing different combinations of mixtures distributions and world iterations (worldits). This last parameter describes the number of times the client model is calculated against the UBM model. Finally, the average model is obtained.

Mixtures	*iterations*	***Equal Error Rate***
32 GMM	*5 worldits*	6,02 %
	15 worldits	5,30 %
	20 worldits	4,92 %
64 GMM	*5 worldits*	3,55 %
	15 worldits	3,12 %
	20 worldits	3, 08 %
128 GMM	*5 worldits*	2,40 %
	15 worldits	2,22 %
	20 worldits	2,15 %
256 GMM	*5 worldits*	2,12 %
	15 worldits	2,05 %
	20 worldits	1,97 %
512 GMM	*5 worldits*	1,55 %
	15 worldits	1,50 %
	20 worldits	1,45 %
1024 GMM	*5 worldits*	1,46 %
	15 worldits	1,43 %
	20 worldits	1,42 %

Table 7.1: EER for different GMM configurations

Secondly, it is interesting to set the DET curve according to our results. In Figure 7.3, results are shown for 64, 128 and 256 Gaussians.

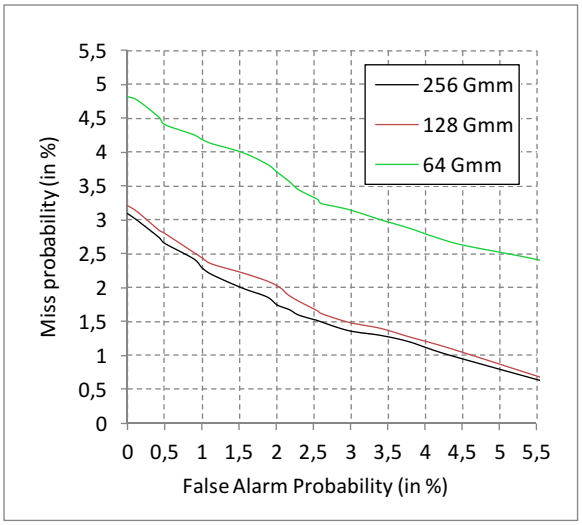

Figure 7.3: Recognition performance for each GMM count on ELSDSR data set using 15 world iterations.

Memory requirements

The BeagleBoard OMAP3530 processor includes 256MB NAND memory in all versions. In this tests the revision BeagleBoard.5 is used, which also includes an additional 128MB DDR. The laptop has 1GB RAM. Note that the feature extraction memory requirement is not contemplated in the table below:

Memory type	5 s (175Kb) audio file		8 s (256Kb) audio file	
	Laptop	**BeagleBoard**	***Laptop***	**BeagleBoard**
Virtual Memory	2,10 (0,21%)	2,38 (1,85%)	3,24 (0,32%)	3,45 (2,69%)
Resident Set Size	1,31 (0,13%)	1,66 (1,29%)	2,32 (0,23%)	2,71 (2,10%)

Table 7.2: RAM requirements on the PC / BeagleBoard for 1024 GMM.

The measures have been taken on a 1024 testing system, using the entire ELSDSR database

The measurements have been performed for a system of 1024 and 512 Gaussian, and the results are quite similar. This is why we can assume the no-dependency between the RAM consumption and the number of Gaussian Mixtures used in the test. As we can see, the memory requirements are not going to be a major handicap, so in the following such sections we will omit the memory measurements.

Time results

The time requirement of this speaker verification system is the most important part of the master thesis. The identification time depends on the number of feature vectors, the complexity of the speaker models and the number of speakers. We want to analyze if it is possible to perform it in a real test, taking into account the computational cost that it represents. The time of the test has been divided into two parts: The first one is to get the time of the processing phases that do not depend on the mixture distribution count used. It means, *features extraction*, *signal processing* and *normalization stages*. The second one consists on evaluating the *training* and *testing* stages using different number of mixtures counts and world iterations.

It is important to understand that the total amount of time expended by our application will be the sum of the time of both stages. Using t_{FE} as *features extraction* time and t_{SP} as *signal processing* time, we can define the total training time as:

$$T_{train} = (t_{FE} + t_{SP}) * n_{files} + t_{train}$$

On the other hand, the total testing time is defined as:

$$T_{test} = t_{FE} + t_{SP} + t_{test}$$

Firstly, we present the results of features extraction and signal processing in Table 7.3. The features extraction, performed by the SPRO application, and signal processing, does not depend on mixture count because the speaker model that uses this distributions has not yet been created. We also take into account the length of the audio file. The longer the speech input file is, the more time the toolkit requires to process its features.

	Features extraction			*Signal processing*		
Audio file length	***Laptop***	***BeagleBoard***	**R**	***Laptop***	***BeagleBoard***	**R**
3 sec	0,42	2,85	6,78	0,04	1,21	30
5 sec	0,80	3,51	4,38	0,06	1,74	29
8 sec	1,35	4,20	3.05	0,10	2,72	27
10 sec	1,95	4,90	2,51	0,14	3,64	26

Table 7.3: Feature extraction and signal processing execution times (in seconds). Ratio (R) is obtained form dividing BeagleBoard time with laptop time

Table 7.4 presents the results of evaluating the testing (t_{test}) and training (t_{train}) times. It is important to mention that this measure does not depend on the number or length of the input audio file. This measure only uses featured files which have a constant length. On the other hand, the number of mixtures has to be considered now. It is important to take into account the world iterations used to create and test the models, because it is the main relevant factor of the configuration parameters.

The results in Table 7.4 show that the proposed SV system for embedded systems can be used in a real-life scenario, in terms of accuracy (as seen in Subsection 7.3.1) and computational performance.

		Testing			*Training*		
Mixtures	*Iterations*	***Laptop***	***BeagleBoard***	**R**	***Laptop***	***BeagleBoard***	**R**
32 GMM	*5 worldits*	10,74	24,35	**2,26**	18	272	**15,11**
	15 worldits	11,33	29,12	**2.57**	19	300	**15,78**
	20 worldits	12,31	33,10	**2,68**	21	321	**15,28**
64 GMM	*5 worldits*	10,89	29,67	**2,72**	25	382	**15,28**
	15 worldits	12,70	31,88	**2,51**	27	405	**15**
	20 worldits	14,51	36.10	**2,48**	31	469	**15,12**
128 GMM	*5 worldits*	13,15	32,44	**2,46**	35	533	**15,22**
	15 worldits	16,20	35,67	**2,20**	37	579	**15,64**
	20 worldits	18,45	40,08	**2,17**	42	657	**15,63**
256 GMM	*5 worldits*	16,81	39.16	**2,33**	48	746	**15,56**
	15 worldits	20,75	45,12	**2,17**	51	776	**15,21**
	20 worldits	25,25	52,13	**2,06**	54	813	**15,05**
512 GMM	*5 worldits*	24,50	68,84	**2,81**	63	973	**15,44**
	15 worldits	30,41	78,21	**2,57**	67	1075	**16,06**
	20 worldits	32,20	85,90	**2,66**	71	1113	**15,67**
1024 GMM	*5 worldits*	30,15	82,91	**2.75**	80	1271	**15,89**
	15 worldits	37,12	90.59	**2,44**	84	1319	**15,70**
	20 worldits	44,15	101,66	**2.30**	89	1420	**15,95**

Table 7.4: Testing and training execution time comparison (in seconds).

Chapter 8

Discussion

In this chapter, a general discussion of the results obtained in the previous chapters will be presented.

During the research, cross-compilation was the stage that was more difficult to obtain. Some operating systems were tested on the BeagleBoard and the minimal Angstrom system was the one that presents the best performance in terms of processing time.

Another point worth interesting is the acquisition of the optimal parameters in the *ALIZE / LIA_RAL* configuration. When reading the literature about this Speaker Verification system, certain specific configurations can be disregarded on embedded systems, due to excessive processing time. Nevertheless, hard work of testing and analyzing specific parameter combinations was done. Using the measurements obtained in Table 7.4, the results presented in the Annex show the best optimization of embedded system, ensuring the same verification results as in the laptop.

We can determine that due to processing requirements, it is more reliable to do just the training stage in the embedded system, as has been done in this Master Thesis. If we take a look on Table 7.4, we can conclude that the time difference ratios between the laptop and the BeagleBoard grows up for the training process. For example, taking 64 GMM, the laptop spended 20 seconds for training stage. On the other hand, the ARM processor of the BeagleBoard took 400 seconds, i.e the training stage is 15 times slower in the embedded system. If we work with databases that take 3-4 minutes for the training in the laptop, the process would take more than 45 minutes in the BeagleBoard, that is not useful in practice when using embedded systems. However, the testing process could be executed with large amount of data, considering a physical implementation of the system, the run time is consistent compared to the execution time on laptop.

Chapter 9

Conclusion and Outlook

In this chapter, we conclude our thesis and present an outlook to future work to advance in this field of research.

9.1 Conclusion

In this thesis the performance viability of a speaker verication software on an embedded system has been evaluated. We present a comprehensive study of the ALIZE / LIA_RAL toolkit as embedded system, and the Beagleboard as a target system. We have also have performed tests that show the possibilities of this speaker verification toolkit on embedded systems. The evaluation has been carried out using the ELSDSR public database.

Our tests indicate that the ALIZE / LIA_RAL speaker verification system can be ported to the Beagleboard and most probably to other ARM embedded systems. As a finding, the tests conclude that we can get exactly the same Equal Error Ratio in the Beagleboard and in the laptop. However, the processing time is highly increased in the embedded system, compared to the laptop one. Furthermore, we presented an optimal configuration of the ALIZE / LIA_RAL parameters, in order to get the best results in the ARM system, in terms of EER.

As a conclusion, the processing time is the most relevant factor that differentiate the laptop and the embedded system functionality. Our tests indicate that time ratio is increased by a factor of ~15, and we can conclude that this system can be used only for testing purposes, as time for training is too long.

9.2 Outlook

This master thesis validates the correct functionality of the ALIZE / LIA_RAL toolkit on embedded systems, almost for testing purposes. In order to obtain our own speaker verification software, specific for embedded systems applications, the main goal will be:

- Test the ALIZE / LIA_RAL Speaker Verification toolkit with a larger database.

- Port this software to other ARM architecture, and compare the results with the obtained using the Beagleboard.
- Improve the ALIZE / LIA_RAL Speaker Verification source code omptimizing the code for these embedded systems.

Bibliography

[1] T. Kamm-M. Ordowski M. Przybocki A. Martin, G. Doddington. The det curve in assessment of detection task performance. *Proceedings of Eurospeech*, 4, 1997.

[2] D.B Rubin A.P. Dempster, N.M. Laird. Maximum-likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, Ser. B, 39, 1977.

[3] J. Pelecanos-G.N. Ramaswamy B. Tydlitat, J. Navratil. Text-independent speaker verification in embedded environments. *In proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, Honolulu*, 2007.

[4] L.R. Rabiner B.H. Juang. Automatic speech recognition: A brief history of the technology development. *Georgia Institute of Technology, Atlanta*, 2005.

[5] J. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 1998.

[6] T. Yingthawornsuk C. Ittichaichareon, S. Suksri. Speech recognition using mfcc. *International Conference on Computer Graphics, Simulation and Modeling*, 2012.

[7] L.P. Heck D.A. Reynolds. Automatic speaker recognition. *AAAS Meeting, Humans, Computers and Speech Symposium*, 2000.

[8] R.B. Dunn D.A. Reynolds, T.F. Quatieri. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing, M.I.T, Lincoln Laboratory,Lexington, Massachusetts*, 2000.

[9] R.C. Rose D.A. Reynolds. Robust text-independent speaker identification using gaussian mixture speaker models. *Lincoln Laboratory, MIT, Lexington, MA*, 1995.

[10] D.Reynolds. Universal background models. *MIT Lincoln Laboratory, USA*, 2001.

[11] S. Sridharan J. Pelecanos. Feature warping for robust speaker verification. *ISCA Workshop on Speaker Recognition*, 2001.

[12] S. Meignier J.F. Bonastre, F. Wils. Alize, a free toolkit for speaker recognition. *LIA / CNRS, Universite Avignon*, 2005.

[13] H. Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, 45, No.4, 2005.

[14] B.H. Juang L.R. Rabiner. Fundamentals of speech recognition. *Prentice Hall, Englewood Clis, New Jersey*, 1993.

[15] B.H. Juang L.R. Rabiner. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3, 1996.

[16] H. Lloyd-Thomas R. Auckenthaler, M. Carey. Score normalization for text-independent speaker verification systems. *Digital Signal Processing*, 10, 2000.

[17] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *ATT Bell Lab, Murray Hill, New Jersey*, 1989.

[18] V. Tiwari. Mfcc and its applications in speaker recognition. *International Journal on Emerging Technologies*, 2012.

[19] D.A. Reynolds W.M. Campbell, D.E. Sturim. Support vector machines using gmm supervectors for speaker verification. *MIT Lincoln Laboratory*, 2006.

[20] K.H. Pun Y.S. Moon, C.C. Leung. Fixed-point gmm-based speaker verification over mobile embedded system. *Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin*, 2003.

Appendix A

SD Formatting code

Script definition for formatting SD card:

```
#! /bin/sh

export LC_ALL=C

if [ $# -ne 1 ]; then
        echo "Usage: $0 <drive>"
        exit 1;
fi

DRIVE=$1

dd if=/dev/zero of=$DRIVE bs=1024 count=1024

SIZE=`fdisk -l $DRIVE | grep Disk | grep bytes | awk '{print $5}'`

echo DISK SIZE - $SIZE bytes

CYLINDERS=`echo $SIZE/255/63/512 | bc`

echo CYLINDERS - $CYLINDERS

{
echo ,9,0x0C,*
echo ,,,-
} | sfdisk -D -H 255 -S 63 -C $CYLINDERS $DRIVE

sleep 1

if [ -x `which kpartx` ]; then
        kpartx -a ${DRIVE}
fi

PARTITION1=${DRIVE}1
if [ ! -b ${PARTITION1} ]; then
        PARTITION1=${DRIVE}p1
fi

DRIVE_NAME=`basename $DRIVE` DEV_DIR=`dirname $DRIVE`

if [ ! -b ${PARTITION1} ]; then
        PARTITION1=$DEV_DIR/mapper/${DRIVE_NAME}p1
fi

PARTITION2=${DRIVE}2
if [ ! -b ${PARTITION2} ]; then
        PARTITION2=${DRIVE}p2
fi

```

```
49 if [ ! -b ${PARTITION2} ]; then
50         PARTITION2=$DEV_DIR/mapper/${DRIVE_NAME}p2
51 fi
52
53 #make partitions.
54
55 if [ -b ${PARTITION1} ]; then
56         umount ${PARTITION1}
57         mkfs.vfat -F 32 -n "boot" ${PARTITION1}
58 else
59         echo "Cant find boot partition in /dev"
60 fi
61 if [ -b ${PARITION2} ]; then
62         umount ${PARTITION2}
63         mke2fs -j -L "Angstrom" ${PARTITION2}
64 else
65         echo "Cant find rootfs partition in /dev"
66 fi
```

Appendix B

Optimal ALIZE / LIA_RAL parameters

Configuration file for TrainTarget

```
1  *** TrainTarget Configuration File
2  ***
3  distribType                    GD
4  mixtureDistribCount            64
5  maxLLK                         200
6  minLLK                         -200
7  bigEndian                      false
8  saveMixtureFileFormat          XML
9  loadMixtureFileFormat          XML
10 loadFeatureFileFormat          SPRO4
11 featureServerBufferSize        ALL_FEATURES
12 loadMixtureFileExtension       .gmm
13 saveMixtureFileExtension       .gmm
14 loadFeatureFileExtension       .tmp.prm
15 featureFilesPath               data/ELSDSRdatabase/features/
16 mixtureFilesPath               data/ELSDSRdatabase/mixtures/
17 labelFilesPath                 data/ELSDSRdatabase/labels/
18 lstPath                        data/ELSDSRdatabase/lists/
19 baggedFrameProbability         1
20 // mixtureServer
21 nbTrainFinalIt                 1
22 nbTrainIt                      1
23 labelSelectedFrames            speech
24 useIdForSelectedFrame          false
25 normalizeModel                 false
26 // targetIdList                data/ELSDSRdatabase/lists/trainFAML.lst
27 inputWorldFilename             wld //
28 alpha                          0.75
29 MAPAlgo                       MAPOccDep
30 MAPRegFactorMean               14
31 featureServerMask              0-15,17-33
32 vectSize                       33
33 frameLength                    0.01
34 debug                          true
35 verbose                        false
36 meanAdapt                      true
```

Configuration file for TrainWorld

```
1  *** TrainWorld Configuration File
2  ***
3  distribType                    GD
4  mixtureDistribCount            64
5  maxLLK                         200
```

```
6  minLLK                          -200
7  bigEndian                       false
8  saveMixtureFileFormat           XML
9  loadMixtureFileFormat           XML
10 loadFeatureFileFormat           SPRO4
11 featureServerBufferSize         ALL_FEATURES
12 loadMixtureFileExtension        .gmm
13 saveMixtureFileExtension        .gmm
14 loadFeatureFileExtension        .tmp.prm
15 featureFilesPath                data/ELSDSRdatabase/features/
16 mixtureFilesPath                data/ELSDSRdatabase/mixtures/
17 labelFilesPath                  data/ELSDSRdatabase/labels/
18 lstPath                         data/ELSDSRdatabase/lists/
19 baggedFrameProbability          0.05
20 baggedFrameProbabilityInit      0.1
21 labelSelectedFrames             speech
22 addDefaultLabel                 true
23 defaultLabel                    speech
24 normalizeModel                  false
25 featureServerMask               0-15,17-33
26 vectSize                        33
27 frameLength                     0.01
28 // inputFeatureFilename         ../../../data/ELSDSRdatabase/lists/UBMlist.lst
29 fileInit                        false
30 // inputWorldFilename           wld
31 initVarianceCeiling             10
32 initVarianceFlooring            1
33 finalVarianceFlooring           0.5
34 finalVarianceCeiling            5
35 nbTrainIt                       20
36 nbTrainFinalIt                  20
37 outputWorldFilename             wld
38 debug                           true
39 verbose                         true
```

Configuration file for ComputeTest

```
1  *** ComputeTest Config File
2  ***
3  distribType                     GD
4  loadMixtureFileExtension        .gmm
5  // saveMixtureFileExtension     .gmm
6  loadFeatureFileExtension        .tmp.prm
7  mixtureDistribCount             64
8  maxLLK                          200
9  minLLK                          -200
10 bigEndian                       false
11 saveMixtureFileFormat           XML
12 loadMixtureFileFormat           XML
13 loadFeatureFileFormat           SPRO4
14 featureServerBufferSize         ALL_FEATURES
15 featureFilesPath                data/ELSDSRdatabase/features/
16 mixtureFilesPath                data/ELSDSRdatabase/mixtures/
17 labelSelectedFrames             speech
18 labelFilesPath                  data/ELSDSRdatabase/labels/
19 frameLength                     0.01
20 segmentalMode             segmentLLR
21 topDistribsCount                10
22 computeLLKWithTopDistribs       COMPLETE
23 //ndxFilename                   data/ELSDSRdatabase/lists/aitor.lst
24 //worldModelName               wld
25 // outputFilename               data/ELSDSRdatabase/res/testing.res
26 gender                          F
27 debug                           true
28 verbose                         false
29 featureServerMask               0-15,17-33
30 vectSize                        33
31 inputWorldFilename              wld
```

Configuration file for NormFeat

```
1  *** NormFeat config File
2  ***
3  mode                              norm
4  debug                             false
5  verbose                           true
6  bigEndian                         false
7  loadFeatureFileFormat             SPRO4
8  saveFeatureFileFormat             SPRO4
9  loadFeatureFileExtension          .tmp.prm
10 saveFeatureFileExtension          .norm.prm
11 featureServerBufferSize           ALL_FEATURES
12 sampleRate                        100
13 saveFeatureFileSPro3DataKind      FBANK
14 //inputFeatureFilename            data/ELSDSRdatabase/lists/testList.lst
15 labelSelectedFrames               speech
16 segmentalMode                     false
17 writeAllFeatures                  false
18 labelFilesPath                    data/ELSDSRdatabase/labels/
19 frameLength                       0.01
20 defaultLabel                      speech
21 addDefaultLabel                   speech
22 vectSize                          34
23 featureServerMode                 FEATURE_WRITABLE
24 featureFilesPath                  data/ELSDSRdatabase/features/
25 featureServerMemAlloc             50000000
```

Configuration file for EnergyDetector

```
1  *** EnergyDetector Config File
2  ***
3  loadFeatureFileExtension          .enr.tmp.prm
4  // saveFeatureFileExtension       .norm.prm
5  minLLK                            -200
6  maxLLK                            200
7  bigEndian                         false
8  loadFeatureFileFormat             SPRO4
9  saveFeatureFileFormat             SPRO4
10 saveFeatureFileSPro3DataKind      FBANK
11 featureServerBufferSize           ALL_FEATURES
12 featureFilesPath                  data/ELSDSRdatabase/features/
13 mixtureFilesPath                  data/ELSDSRdatabase/mixtures/
14 lstPath                           data/ELSDSRdatabase/lists/
15 // inputFeatureFilename           testList.lst
16 labelOutputFrames                 speech
17 labelSelectedFrames               male
18 addDefaultLabel                   true
19 defaultLabel                      male
20 saveLabelFileExtension            .lbl
21 labelFilesPath                    data/ELSDSRdatabase/labels/
22 frameLength                       0.01
23 writeAllFeatures                  true
24 segmentalMode                     file
25 nbTrainIt                         8
26 varianceFlooring                  0.5
27 varianceCeiling                   10
28 alpha                             0.0
29 mixtureDistribCount               3
30 featureServerMask                 16
31 vectSize                          1
32 baggedFrameProbabilityInit        1.0
33 debug                             true
34 verbose                           false
```