INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Speech Recognition system for mobile devices

João Tiago Rodrigues de Sousa Miranda

Dissertation for obtaining the Master's Degree in

Information Systems and Computer Engineering

Jury

October 2008

Acknowledgements

I wish to thank my advisor, Professor João Paulo Neto, for his invaluable ideas and suggestions.

I also want to thank all the members of L^2F in general, and in particular those who provided me with an understanding of *Audimus*' internals, support and tools necessary for the completion of this work.

Finally, I would like to acknowledge the support of my family and friends throughout the process of writing this thesis.

Lisbon, October 13, 2008

João Tiago Rodrigues de Sousa Miranda

Abstract

In this thesis, an Automatic Speech Recognition system for large vocabulary (more than 10000 words) tasks is presented. This system is intended to run on resource-constrained mobile devices such as PDAs and is based on *Audimus*, a hybrid Hidden Markov Model / Multi Layer Perceptron based speech recognizer, tailored for the European Portuguese language. The system's main goal is to ease interaction with these devices by adding a speech interface.

The present thesis describes the progressive porting of the speech recognition system from the desktop computer to the mobile device. The main components of the system (feature extraction, acoustic model, and decoder) were progressively ported to the mobile device, producing intermediate Distributed Speech Recognition systems. To overcome the limitations of the devices, it was necessary to choose appropriate algorithms and to balance the speed and memory use of the system with its performance in terms of word error rates (WER). The final system had the same characteristics as the baseline system.

The obtained system was tested in a 520 MHz Pocket PC with 64 MB of RAM running Windows Mobile 6.0. The tests used a continuous, 13,161-word radiology task with existing language models. The baseline system, tested with audio recorded with the PDA's internal microphone and with a speaker and microphone adapted acoustic model, achieved an average WER of 3.75%. The two intermediate distributed systems had WER of 3.80% and 4.71%, respectively, while the final embedded system presented an average WER of 7.77%, running at 0.71x RT.

Resumo

Nesta tese apresenta-se um sistema de reconhecimento de fala para tarefas com grandes vocabulários (mais de 10000 palavras). Este trabalho baseia-se no *Audimus*, um reconhecedor de fala para o Português Europeu. O sistema desenvolvido tem como alvo dispositivos de baixos recursos computacionais como PDAs, e como principal objectivo facilitar a interacção entre humanos e dispositivos ao adicionar uma vertente de fala.

Esta tese descreve a progressiva adaptação do sistema do computador para o dispositivo móvel. Os seus principais componentes (extracção de características, modelo acústico, e descodificador) foram progressivamente transferidos para o dispositivo, obtendo-se um conjunto de sistemas distribuídos intermédios. Para tal, foi necessário ultrapassar as limitações de recursos dos dispositivos alvo, através de uma optimização cuidada dos algoritmos, e de *tradeoffs* entre os erros de transcrição do sistema e o seu tempo de execução. O sistema final mantém as caracteristicas do sistema inicial.

O sistema obtido foi testado num PDA com um processador a 520 MHz e 64 MB de RAM, executando o Windows Mobile 6.0. Utilizou-se uma tarefa de radiologia, com 13161 palavras, contínua e adaptada ao falante. Com áudio gravado com o microfone do PDA e um modelo acústico adaptado ao falante e ao microfone, o sistema inicial (no PC) apresenta uma taxa de erro de 3,75%, ao passo que os dois sistemas distribuídos intermédios têm 3,80% e 4,71%, respectivamente, e o sistema final apresenta um erro de 7,77%, executando numa fracção (0,71) do tempo real.

Palavras Chave
Keywords

Palavras Chave

Reconhecimento de Fala Automático

Reconhecimento de Fala Distribuído

Sistemas Embebidos

Perceptrão Multicamada

Transdutores Pesados de Estados Finitos

Keywords

Automatic Speech Recognition

Distributed Speech Recognition

Embedded Systems

Multilayer Perceptron

Weighted Finite State Transducers

Table of Contents

List of Figures

List of Tables

List of abbreviations

ARM - Advanced RISC Machines

ASR - Automatic Speech Recognition

BLAS - Basic Linear Algebra Systems

DFT - Discrete time Fourier Transform

DSP - Digital Signal Processor

DSR - Distributed Speech Recognition

FFT - Fast Fourier Transform

L^{2}F - Spoken Language Systems Laboratory

MLP - Multilayer Perceptron

NSR - Network Speech Recognition

PDA - Personal Digital Assistant

PLP - Perceptual Linear Prediction

WER - Word Error Rate

XML - Extensible Markup Language

1 Introduction

In recent years, mobile devices such as mobile phones and PDAs have acquired great importance in our lives, because they enable their users to be integrated with the world everywhere and at any time. The ease of interaction with these devices is a very important factor, but their size makes it very difficult to design effective input devices. In this respect, speech recognition offers an alternative to the traditional, unmanageable input devices such as small keyboards or stylus pens.

Automatic Speech Recognition (ASR) systems attempt to transcribe human speech into words. Speech recognition is a complex task, and ASR systems usually integrate knowledge from many different sources, usually including models of the acceptable sentences in the language, as well as word pronunciation models, building up a large search space that combines this knowledge. ASR systems are complex enough for modern workstations, where they sometimes occupy hundreds of megabytes of RAM, let alone for mobile devices such as PDAs. For this reason, traditionally, many ASR systems for low-resource devices have been limited to small-vocabulary tasks, such as the recognition of a list of names for phone dialing.

Mobile devices have not followed Moore's law in the last decade - doubling their speed every 18 months - mainly due to problems with energy consumption, which is limited due to the need to conserve battery life. Their processing capabilities, however, have been slowly but steadily increasing: a typical PDA currently (as of 2008) has between 32 and 128 MB of RAM, with processors with speeds in excess of 400 MHz, that sometimes include specialized instructions to accelerate multimedia processing . This opens up a new set of processing-intensive applications, like video and audio processing, games, or speech recognition, that were previously very difficult to implement in these devices. These new applications enrich user experience and lift some of the barriers previously hampering the widespread adoption of mobile devices.

Besides mobile phones and PDAs, many other applications can benefit from low-resource speech recognition systems, such as robots that can listen and respond to a user, car navigation systems and computer game consoles, for instance.

1.1 Goals

The current work's primary goal is to port an Automatic Speech Recognition System, *Audimus* (Meinedo et al., 2003), to mobile devices such as PDAs or mobile phones. This implies adapting the underlying algorithms in order to keep performance as high as possible, in terms of error rates and task sizes (the target being to be able to use large vocabulary models in the embedded system, of about ten thousand words), considering the resource constraints of these devices. The obtained system has to be able to work at near real time or, preferably, under real time, for the largest task it is supposed to handle. The completed system will use all the available memory in the device - in contrast with some systems which attempt to have minimal memory footprint - so that it is possible to take advantage of models of maximal size.

To realize this goal, it was decided to port the *Audimus* system by transferring each of its composing blocks sequentially (feature extraction, acoustic model, and finally decoder). In order to achieve this, it is necessary to distribute the processing between the system that is currently working and the system that is being built in the mobile device. In the beginning, the processing is all done in the desktop computer, being progressively moved to the device. In the end, the system as a whole resides in the mobile device.

The systems that arise from these intermediate steps are known as Distributed Speech Recognition (DSR) systems. These are the intermediate or secondary goals of the thesis, which were considered less important than its primary goals. Some of the problems that must be addressed if DSR systems are to work in realistic networks were therefore not considered in this thesis; the networks were chosen so as to simplify the problem to be addressed.

1.2 Target device and operating system

This section describes the target device and its operating system, in order to motivate the main problems that this thesis purports to solve.

1.2.1 Target device

The target device is a 520 MHz PocketPC, with 64 MB of RAM and 128 MB ROM running Windows Mobile 6.0. It is equipped with a processor (PXA270x) (Intel, 2004) of the XScale family, with a core based on the ARM920T. This is a microprocessor with no integrated floating-point capabilities, but which includes some DSP (Digital Signal Processing) capabilities in the Wireless MMX extensions (Paver et al., 2003). The PXA270x also does not possess a division instruction - this is common in embedded microprocessors in order to save chip area and energy consumption.

As said above, the processor of the target device has no integrated floating-point capabilities. Emulating the inexistent floating-point units through software incurs a slowdown of about 10x, which renders them unacceptable for applications that are processing intensive in terms of arithmetic operations. This is the case of most components of the *Audimus* system. The solution is the use of fixed point computations in all arithmetic operations. Library functions that rely on floating point operations must be reimplemented to remove that dependency.

Also, the 100 MHz memory bus is a limiting factor in terms of bandwidth, since only 100 million words (at best) can be read from the main memory in one clock cycle, but the memory latency is perhaps a more constraining factor, since a word takes several dozens of CPU cycles to arrive, counted from the clock cycle in which it is requested.

1.2.2 Operating system

The operating system, Windows Mobile 6.0, is based on Windows CE 5.2's kernel, meaning that it has several limitations that are not present in its desktop version . The aspects that are the most relevant for the work in this thesis include:

- A hard limit of 32 to the number of processes and threads that can be active at each time - the thread, not the process, is the basic execution unit. *Audimus* uses several threads, in particular, at least one for each component of the system. This limit can thus be a problem, in special if a sufficiently large number of other processes is used simultaneously in the system.
- More importantly, there is a limit of 32 MB of usable address space (virtual memory space) per process. This includes process code, data and loaded libraries. This problem is aggravated by the fact that DLLs must be loaded at the same address across all processes, meaning that DLLs loaded for one process "pollute" the address space of other processes. In practice, there is often about 14-18 MB of virtual memory space available at program startup. In this case, it also becomes a significant problem because the target device has 64 MB of RAM, of which about 30 MB are free for use by user programs at startup. Windows Mobile 7, based on Windows CE 6.0, solves the problem by offering each process a 2 GB address space, but it will only be available by the end of 2009.
- The number of usable handles, which include sockets, files, global synchronization objects (mutexes, semaphores, critical sections), and other resources, is also limited to 512. *Audimus* uses threads and therefore synchronization objects intensively, but as long as it is contained in a single process, the use of global synchronization objects can be mostly avoided; the use of temporary files must also be controlled so that it does not exhaust the system resources.

1.3 The network

The distributed system functions across an ideal network which does not lose, corrupt, or reorder packets. Networks using TCP/IP transport are, from the application's viewpoint, close enough to these ideal properties, and were therefore used for this thesis. Also, it is assumed that the network that is used has enough bandwidth to support the transmission of all these features without problems. A Wireless 802.11g network was chosen, which at 54 Mbps, has more than enough bandwidth to enable the transmission of speech sampled at 44 KHz, with 16-bit resolution. Finally, the network considered is assumed to have low latency, to make real time recognition possible. In realistic networks, it is difficult to guarantee a hard upper bound on latency.

These assumptions are formulated so that it is possible to avoid focusing on packet loss mitigation, feature quantization, and error correction techniques that are outside the scope of this thesis, as mentioned in section 1.1.

1.4 Thesis Overview

The thesis consists of seven chapters, organized in the following way:

In the second chapter, the state of the art in Automatic Speech Recognition (ASR) is presented. After a brief introduction to ASR systems in general, the algorithms and techniques that have been used to implement Distributed Speech Recognition (DSR) and Embedded Speech Recognition systems are surveyed. The advantages and disadvantages of each of these two different approaches are also compared.

The third chapter presents the *Audimus* ASR system, focusing on its different applications. It then describes *Audimus*' main components, with special emphasis on those which need to be ported to achieve the goal of this thesis, serving as a basis for the two subsequent chapters.

The following two chapters (chapters 4 and 5) are organized in a way that reflects the progressive porting of the system to the mobile device; the order in which the components are presented is therefore the same as the order in which they were ported to the mobile device.

The fourth chapter is centered on the distributed speech recognition systems that were produced as intermediate system prototypes. It is concerned essentially with explaining the changes that had to be made to the feature extraction and acoustic model computation algorithms in order to make them work efficiently in the target device.

The fifth chapter analyses the final embedded system that was obtained. In particular, it describes the porting of the last and most resource consuming component of the system, the decoder, that integrates all the knowledge sources available (acoustic model, lexicon, language model, for instance) and finds the sentence that is the most likely to have been uttered by the speaker.

Both the fourth and the fifth chapters also include the results of the tests that were performed to evaluate the quality of the distributed and embedded systems produced. In the fourth chapter, the testing procedure, which is the same for all tests, is described. Then, in chapters 4 and 5, the results pertaining to each of the chapters are presented and discussed.

In the sixth and final chapter of this thesis, some conclusions are drawn from the work in this thesis and the results obtained in previous chapters. Possible future improvements to the present work are also laid out.

2 State of the Art

2.1 An Introduction to Speech Recognition

2.1.1 Introduction

Speech recognition can be defined as a process through which a computer system acquires a speech signal produced by a human speaker, possibly distorted by noise in the environment, and tries to reconstruct the utterance produced by the speaker. Humans are excellent at understanding other human speakers; for machines, the problem is harder because they do not (usually) have access to or understand clues such as facial expressions, gestures, and discourse context people use to extract meaning from a voice signal.

The problem of perfectly recognizing an unrestricted sentence said by an arbitrary speaker in a noisy environment is well beyond the capabilities of current systems, because problems such as background noise or different speaker pronunciations have not yet been solved satisfactorily. Also, many systems force their users to speak in an unnatural way to achieve good recognition results; that is, they are not well suited for spontaneous speech.

2.1.2 Classification of ASR systems

Speech recognition systems can be classified based on the complexity of the recognition task they are supposed to undertake, which in turn can be defined by a set of orthogonal attributes (Young, S., 1995). Some of the most important are:

2.1.2.1 Continuous/Discrete Systems

In continuous systems, in contrast with discrete or isolated-word systems, the speaker is not required to pause between words, making it harder to segment the signal into words.

2.1.2.2 Vocabulary Size

Systems can also be divided based on the size of the vocabulary to be recognized. Naturally, larger vocabulary sizes can be used for richer and more complex tasks, but they also give the recognizer a larger space to search through, which complicates the problem. For instance, a voice dialing system may require a vocabulary containing just the ten digits (0-9) and a few control words (a very small vocabulary). Medium sized vocabularies containing a few thousand words can be appropriate for some tasks in a limited domain, for instance, a system for controlling appliances in a car. However, a system that transcribes the broadcast news must consider a much larger vocabulary (tens or hundreds of thousands of words), since the words spoken come from many different domains and issues. Intuitively, the latter system will attain the largest word error rate (ratio between words incorrectly transcribed and the total number of words), all other conditions being equal, since for each spoken word there are more words to potentially be 'confused' with.

2.1.2.3 Speaker Independence

It is possible to distinguish the systems on whether or not they are speaker independent, that is, on whether they can, or not, achieve good recognition quality with speakers they have not been trained with before. This is important because many systems need to be prepared to understand arbitrary speakers, for example, systems that sell tickets for cinema, etc, while others may be trained to work with only one or a few speakers, as may be the case of systems designed to work in mobile devices such as PDAs. Speaker dependent systems may achieve higher recognition rates and be able to use smaller models, because they are better adapted to the speaker they model.

Systems are usually speaker adapted in a process which involves the user reading a number of predefined sentences. The system then aligns these sentences with the acoustic observations, using them to train the acoustic models in the system.

2.1.3 Applications

Speech recognition systems can be used in a vast number of applications, a few of which are described next.

2.1.3.1 Spoken Dialog Systems

Systems that engage in a conversation with an user in spoken natural language are an important research area. These systems enable users to do away with the need to remember a complex interface, that would be inadequate for many users and in many situations. Dialog systems can be classified based on who directs the conversation - the user, the system, or both - and, on another dimension, on the range of vocabulary they support (McTear, 2002).

2.1.3.2 Dictation Systems

In dictation systems, an user dictates a text to the computer using spoken language instead of typing it in the keyboard. Usually, these systems have large vocabulary models. While currently, these systems are inferior for experienced keyboard users, who can type faster than they speak without introducing as many errors as a dictation system would, they can be invaluable for novice computer users or in small embedded devices where the input devices are of inferior quality.

2.1.3.3 Broadcast News Recognition

The automatic transcription of broadcast news is useful, for instance, to the hearing impaired, or to search for news that have appeared in the past, but it requires large vocabulary models to be effective. *Audimus* has been used for Broadcast News Recognition (Meinedo, H. and Neto, J. P., 2003) .

2.1.3.4 Speech-to-Speech Translation

These systems require a natural language processing and a text-to-speech module, and can be of invaluable help when people that speak different languages need to interact without an interpreter. They are more computationally demanding than other systems because they use three complex modules. An interesting example of an (embedded) two-way speech-to-speech system appears in (Hsiao et al., 2006), which was tested by American forces in a war scenario in Iraq.

2.1.4 Current Speech Recognition Techniques

In the current section, the basics behind current state-of-the-art speech recognition systems are described. A more detailed overview can be found in (Young, S., 1995). Speech recognition systems can be seen as being made of several modules which act sequentially on the speech signal, in order to transform it into a discrete sequence of words. A reasonable approach to solving this would be finding the sequence of words which maximizes the probability of the observations having been generated by it; in other words, we wish to find the word sequence W such that the probability $P(O|W)$ is maximal over all possible word sequences. Calculating this probability is not straightforward, but using Bayes' theorem it is possible to write:

$$P(O|W) = \frac{P(W|O)P(W)}{P(O)} \tag{2.1}$$

For each word sequence W, we can obtain estimates of the probability $P(W)$ by using, for instance, a language model. We can also obtain estimates of the probability $P(O|W)$ by using an acoustic model. Since we are maximizing for a constant observation, we can drop the $P(O)$ dividing factor; this will not affect the word sequence chosen.

The speech signal is usually first divided into several short, normally overlapping, windows of between 5-20 milliseconds, and these windows are subsequently analyzed with signal processing techniques such as Fourier Transforms which are aimed at 'extracting' a low-dimensionality set of features from the speech signal, that better represent it by abstracting from noise and speaker variability, among other factors. Accordingly, the output of this initial processing phase is a number of feature vectors, one for each window of speech. This allows us to work with a much more compact representation of the observations.

The speech signal is now usually modeled as a Hidden Markov Model (HMM), which is basically a set of states which emit observations, along with transition probabilities among the states and emission probabilities. An introduction to HMMs and their use in speech recognition can be found in (Rabiner, L.R., 2000). In this model, the probability of being in one state x_i depends exclusively on the state x_{i-1}; also, the probability of emitting a given observation depends only on the current state. In an HMM model, states are hidden in that one only has access to the observations - that is, it is not possible to know which sequence of states generated a given set of observations. In the particular case of speech recognition, the observations are the feature vectors and states represent, for example, the phones that were spoken. Many systems use tri-phones as the states of the HMM because the acoustic realization of a phone is context-dependent, that is, depends on the phones that precede and follow it.

The emission probabilities are calculated using an acoustic model. The acoustic model can be realized as a classifier, such as a Multi-Layer Perceptron, as in *Audimus*, that maps the vectors obtained in the feature extraction step to probabilities, leading to the hybrid HMM-MLP approach (Boulard & Morgan, 1994).The other usual way to implement the acoustic model is through the use of a Gaussian Mixture Model (GMM) (Young, S., 1998). Gaussian Mixture Models model the probability distribution as a combination of Gaussian distributions, being therefore most effective when the probability distribution is a smooth (continuous) function.

Language models assign a probability to each word sequence, in order to filter out word sequences that are not acceptable in the language being recognized. Most current systems employ n-gram models, which use a context consisting of the $n-1$ most recent words to condition the probability of the current word . The probability of a word sequence in this model can be computed using equation 2.2:

$$P(w_1 \dots w_j) = P(w_1)P(w_2|w_1)\dots P(w_{n-1}|w_1 \dots w_{n-2}) \prod_{i=n}^{j} P(w_i|w_{i-n+1} \dots w_{i-1}) \qquad (2.2)$$

N-gram models encode useful local language constraints, but fail to capture deeper, long distance syntactic and semantic interactions. They are preferred to other alternatives, however, because the training process is relatively simple (essentially counting in the training set) and evaluation is very efficient, as it can be done directly via equation 2.2. An alternative sometimes used in simpler systems (for example, command recognition systems) is to use language models that directly enumerate all the possible sentences.

Considering again the recognition problem as posed above, and since the model is stochastic, we need to find the best path through the HMM that was constructed, that is, the path which is the most likely to have generated the observations seen. In order to accomplish that, speech recognition systems often use variations of algorithms such as the Viterbi algorithm (Ryan & Nudd, 1993) (a dynamic programming algorithm built on the assumption that the best path over i observations must contain a best path over $i-1$ observations) or the A* algorithm (decoders built using this algorithm are also known as stack decoders (Paul, 1992)).

Many current systems, of which *Audimus* is an example, approach the search space modeling problem by using Weighted Finite State Transducers (WFSTs) (Mohri et al., 2002). WFSTs are finite state automata, in which the transitions have been augmented with an output symbol (to be output when the WFST follows the transition) and a weight. In spite of only being able to describe weighted mappings between regular languages, it is possible to express both the acoustic and language (n-gram) models as WFSTs. WFSTs can benefit from most of the tools and operations available in finite state automata (composition, inversion, minimization, determinization, etc.), to represent all the search space in a single WFST. Transducer composition can be done online, using specialized algorithms that approximate determinization and minimization, thus eliminating the need of storing/loading a huge transducer into memory. Additionally, the time taken to explore the transducer and thus the overall decoding time can be limited at the expense of accuracy, using all the techniques available in traditional decoders, such as beam search. But the key advantage of the use of WFSTs is perhaps their flexibility: it is possible to integrate information to the recognition process from many sources without changing the decoder, by simply composing with a transducer that represents the information to integrate. This contrasts with traditional approaches in which different sources of information are combined in an ad-hoc way and new code has to be written each time a different information source is added to the system.

2.2 Distributed Speech Recognition

Because of their smaller size, weight, and limited power consumption when compared to desktop computers, mobile devices have smaller memory sizes, reduced processing speeds and/or instruction sets. As a result, the software and operating systems used in these applications usually also has less functionality. Also, speech recognition applications often require large amounts of memory and processing time to achieve good results. One of the ways to do this, as seen in the next section, is to adapt the algorithms to these settings with reduced resources, even if that results in worse ASR performances. An alternative to this are systems based in the client-server model, where the mobile device is the client of a service provided in the network.

An architecture that transfers all the processing to the servers in the network, known as Network Speech Recognition (NSR) (Kiss, 2000) could be the immediate solution. The device would then just acquire the speech signal and send it to the server for processing. Finally, the server would send back the result of the recognition. On the positive side, this architecture enables seamless upgrades to the system (from the user viewpoint), and enables the embedded device to do away with almost all computation. But, on the negative side, wireless network connections normally have limited bandwidth, which forces the use of low bit-rate codecs, substantially degrading the recognition rates.

The above-mentioned disadvantages have led to the development of systems where the processing is split between the client and the server: Distributed Speech Recognition (DSR)(Zhang et al., 2000) systems . This is done by having the client execute the feature extraction blocks (also known as the front-end), therefore acquiring a set of coefficients it then sends to the server. It now has to execute the search portion of the ASR task, using its acoustic and language models (the server executes the back-end). When the server has terminated this part of the process it will send the result of the recognition process back to the client, just like in the fully server-based architecture described above.

The fact that only a set of coefficients per window of speech is necessary to represent the speech signal, which can be compressed more easily than the speech signal itself, reduces the information that needs to be sent across the network. Also, the feature extraction blocks use but a small fraction of the total processing time. So, when the remaining blocks are transferred to the server, the strain on the embedded device is reduced, possibly freeing it for other tasks. The main disadvantage of NSR systems - the excessive use of network bandwidth which results in worse recognition quality - is thus mitigated by DSR systems.

In light of this, there are several problems (apart from noise robust feature extraction in the device, which is addressed in the next section) that need to be addressed before building a DSR system. These are considered in the next subsection.

2.2.1 Issues to consider before building a DSR system

2.2.1.1 Compressing the feature vectors

It must be decided how to compress the feature vectors that are sent across the network in order to further reduce the amount of information to be transmitted; this involves finding quantization techniques that allow the reduction of the number of bits sent per feature without significantly impacting recognition quality. To do this, one can use scalar quantizers. Scalar quantizers treat each component of the feature vector individually, and quantize its values in a uniform or non-uniform way. An alternative is to use vector quantizers (VQ), that assign a code to each region of a feature vector space. Vector quantizers achieve better performance rates when compared to scalar quantizers, but they are not bit rate scalable - that is, they do not adapt to different bit rates gracefully. Also, because of the computational effort required to accurately quantize the feature vector space, it is necessary to explore suboptimal vector quantization techniques such as product-code vector quantization (Digalakis et al., 1998). Here, the feature space is partitioned into subspaces, which are then vector quantized independently, thereby reducing the size of each codebook. Other, more sophisticated techniques combine vector quantization with Gaussian Mixture Models (Hedelin & Skoglund, 2000) to achieve better performance.

2.2.1.2 Error control and mitigation

It is necessary to decide how to control the errors in the transmission, which are inevitable in some channels. To detect the occurrence of errors, standard error detection codes as Cyclic Redundancy Checks (CRCs) are often used, providing good protection in most channels, as in (*ETSI ES 201 108 v1.1.2 distributed speech recognition (front-end feature extraction)*, 2000). The client can also protect the transmitted features through the use of forward error correcting codes, which can be used by the server to correct errors that may have occurred, when combined with interleaving, that addresses burst errors. Alternatively, or in addition, server-based error concealment techniques may be considered (Tan et al., 2004). These include interpolation-based techniques, statistical methods and soft feature techniques. These methods attempt to replace the missing features with some value that can reasonably be expected to be close to the actual value, based on information of the received features. Finally, ASR-based methods (methods that are integrated with the recognition process itself, which assigns less importance to feature vectors that have values that are not known) may also be used. The advantages of these server-based methods are that they do not require either modifications to the DSR client or additional bandwidth use (as the error detection and correction codes do). Of course, they lead to degraded performance relative to client-based methods, since the missing information is only approximated.

It is also important to mention here that some systems (e.g. phones) use speech coding, in order to transmit speech over a telephony or IP network as efficiently as possible. The speech is then reconstructed in the other side as closely as possible to the original by using the features received across the network. The features generated by the speech coding process can then be adapted to be used in the speech recognition process, but they are normally extracted at a low frequency and need to be interpolated to keep DSR performance at acceptable levels (Fabregas & Alcaim, 2007). This can make it possible to avoid porting the feature extraction module to the device, which besides sparing programmer time also frees the device's resources for other tasks.

In addition to that, many devices' processors include Digital Signal Processing (DSP) extensions or coprocessors. Usually, these extensions consist of special instructions that can perform saturated add or multiply operations (that avoid overflow by setting the result to the largest or smallest representable integer), or faster arithmetic operations (for instance, a full-width multiply-accumulate instruction that runs in one clock cycle). These capabilities facilitate the implementation of many signal processing algorithms, both by requiring less time to complete them and by optimizing power consumption in the device.

In this work, however, since the production of a distributed system was not the primary goal, it was assumed that the network channel used to transmit information did not lose or corrupt any information, and had enough bandwidth, so error correction and feature quantization techniques were not considered in greater detail. Also, the existence of a speech coding module was not assumed, since this would unnecessarily reduce the range of target devices, some of which do not possess this module.

2.3 State of the Art in Embedded Speech Recognition

The need of achieving embedded speech recognition is largely motivated by the phone / PDA market, driven by instant messaging or dictation applications, for instance. There exist many embedded ASR systems: Nuance's VoCom (*Nuance Vocon*, n.d.), IBM's ViaVoice (*IBM Embedded ViaVoice*, n.d.), or Asahi Kasei's VORERO (*Asahi Kasei VORERO*, n.d.), only to mention a few. These systems are software based solutions that run on a wide range of devices, supporting vocabulary sizes that are only limited by the device's memory. Aside from the mobile phone / PDA market, there is an important section of automotive embedded ASR systems, such as Mercedes' and Daimler-Chrysler's Linguatronic (Heisterkamp, 2001). This is a speaker independent ASR system with between a few hundred and a few thousand words, that enables the user to interact with the car totally hands-free. The computer game market is also interested in embedded ASR systems, as integrating speech recognition in these can offer users new gaming experiences; while the consoles often have more powerful processors than PDAs, computer games are very processing-demanding and the applications required are different. Konami's Lifeline (*Konami's LifeLine*, n.d.), a video game with ASR capabilities built on ScanSoft's speech recognition system, is a good example, recognizing a vocabulary of a few thousand words and phrases.

When trying to build new embedded ASR systems or porting existing ones, one must consider the limitations of the target devices. In particular, the limited capacities of the CPUs of the target devices, mainly in what concerns speed, DSP extensions and the (usual) inability to perform floating point operations, must be taken into account. The last two features are mostly relevant to the feature extraction step. In addition to that, one must look into the limited RAM sizes and perhaps equally importantly, the reduced memory speeds. In fact, since processors in embedded systems often have smaller caches, and memory hierarchies that are not very deep (usually not more than a cache), when compared to their desktop equivalents, memory speed can quickly become a bottleneck. Memory sizes and speeds are mostly a limiting factor of the size of the models (and dictionaries) that can be used in the ASR process. Embedded ASR systems also face limitations imposed by the operating system, namely restrictions on the number of processes or sockets they can create or on the resources they can hold simultaneously. These restrictions are often stricter than the ones imposed in desktop computers OS's.

In the remainder of this section, we present several techniques that can be employed to overcome the memory and CPU limitations of embedded systems while minimizing the negative impact on the recognition performance of the original ASR system.

These techniques are divided in three broad categories: feature extraction optimizations, acoustic model optimizations, and decoding optimizations. Apart from the optimizations there presented, porting ASR systems to these embedded systems often involves coding at a lower level in some sections of the code, since performance is much more critical here and the compiler doesn't always optimize as expected.

2.3.1 Feature extraction optimizations

The most significant obstacle in this subsection is the rewriting of all the operations to perform fixed-point operations. Choice of fixed-point scaling to use is often non-trivial; it may be necessary to rescale between modules of the feature extraction system, or even inside some of the modules. This, however, has no significant impact in performance if it is done at function boundaries and not after each arithmetic operation. An example where rescaling is often performed every two *butterflies* (iterations) is the Fast Fourier Transform (FFT). This is a special case of the Block Floating Point (BFP) (Mitra & Chakraborty, 2002) concept, where scaling is applied to relatively large blocks of data as a whole. In the rest of this paragraph, other problems commonly encountered during the porting of the feature extraction module are discussed.

2.3.1.1 Computation of the Power Spectrum

Many feature extraction techniques involve the computation of the power spectrum's magnitude. Most applications, in order to avoid performing computationally expensive square root operations, work with this magnitude squared. However, the dynamic range of the resulting values exceeds what can be saved in a 32-bit fixed point register with reasonable accuracy; as a result, essentially three options arise: approximating the square root using some fast, lightweight technique such as a linear combination of the components of the vector to be approximated; working with the logarithm of the power spectrum, which is very appropriate for MFCC coefficients, since their calculation requires the computation of the logarithm anyway (Huggins-Daines et al., 2006); or working with the magnitude squared, by using a dual fixed point format (Ewe et al., 2004), where a single bit selects one of two possible exponents.

2.3.1.2 Relative complexity of arithmetic operations

Most processors of the target devices have slow multiplication operations (when compared to additions and shift operations) and often inexistent division operations, that must be emulated in software, taking up to hundreds of cycles. As a result, it is of crucial importance to replace division operations, whenever possible, by multiplications with the inverse or table lookups. In some cases, it may also be possible to replace some of the multiplications or divisions with shifts and additions, when their operands are constant values.

2.3.1.3 Computation of Special Functions

Often, special functions (trigonometric or transcendental) need to be calculated. For instance, the FFT and Inverse FFT transforms require the computation of co-sine and sine functions, whereas PLP feature extraction requires the use of cube-root functions and the calculation of MFCC features requires the computation of the logarithm function (Huggins-Daines et al., 2006). In many cases, these can be exactly replaced by table lookups, using tables that are not excessively large (as in the case of the FFT and Discrete Cosine Transform); if not, adding linear interpolation between table entries is usually sufficient to achieve the desired precision.

2.3.2 Acoustic Model Optimizations

2.3.2.1 Gaussian Mixture Models

Gaussian Mixture Models (Young, S., 1998) often occupy considerable amounts of memory. In order to reduce memory footprint, it is possible to quantize the mean and variance vectors of the Gaussian mixture models. This can be accomplished through the use of all quantization techniques described in the previous section, being usually done with Vector Quantization (VQ) (Singh et al., 2003) techniques. In (Huggins-Daines et al., 2006) the use of scalar quantization is proposed, where each of the Gaussian components' mean and variance is quantized independently, using both uniform and nonuniform quantization methods.

Evaluation of Gaussian likelihoods occupies, in some systems, a significant portion of their total running time. To improve Gaussian evaluation speed, several techniques can be employed. The simplest of these can be found noting that, if aggressive parameter quantization is used, then it is possible that we can calculate the probability distribution function using only table lookups. It is also possible to use Gaussian Selection (Knill et al., 1996) based on the fact that, if a vector is an outlier with respect to a given Gaussian distribution, then the probability of having been generated by this distribution is very small: thus it is possible to neglect it by skipping the evaluation of this Gaussian. This can be implemented by assigning a set of Gaussians to each region of the clustered acoustic space, such that only these components are evaluated during likelihood calculation, while the others are simply approximated or ignored.

2.3.2.2 Multi-Layer Perceptrons

The output of the neural network (MLP) is often calculated using techniques based on matrix multiplication, to simulate the signal propagation over the several layers (often only one) that the network is built from. Matrix multiplication is a rather computationally complex problem, whose usual algorithm takes time $O(n^3)$, where n is the dimension of a n x n square matrix. Efficient algorithms exist that take advantage from the memory hierarchy of the system (which, as a general rule, tends to comprise smaller amounts of faster memory and larger amounts of slower memory), by partitioning the matrices into blocks that fit into the cache and performing matrix multiplication on matrices whose elements are the blocks. Other methods try to exploit the relatively stationary nature of the speech signal: they only propagate the signal from one layer to the next if the difference in activation from one time instant to the next is appreciable (Albesano et al., 1996), though this method is only applicable to Multi-Layer Perceptrons; Single-Layer Perceptron designs do not benefit from its use. The neural network weights may also be quantized in order to save space, since they take up considerable amounts of memory. As a side effect, on processors that lack a single-cycle multiply instruction, as is the case with most embedded processors, multiplying smaller words is often faster.

2.3.3 Decoding Optimizations

Many of the decoder optimizations mentioned in this section apply not only to embedded systems, but to normal ASR systems as well.

2.3.3.1 Beam search

A typical way to speed up search in the decoding step of *Viterbi* and *A** algorithms is to maintain only the set of paths which score is above some threshold, which is usually related to the best score found so far. This is easier to do in *Viterbi* than in *A** because the latter is time-asynchronous, and the likelihoods of paths with different lengths must thus be compared.

2.3.3.2 Use of fast match

Another way to speed up search is to use a fast match technique, as described in (Bahl et al., 1993). This is done by using simplified acoustic and/or language models, or a class of heuristics, to avoid considering words that are unlikely to be part of the decoded sentence. By doing this, the search space is reduced, while not degrading the recognition accuracy considerably if the model is chosen appropriately.

2.3.3.3 Multipass Decoding

Related to the fast match idea is the use of multipass decoding schemes (Gosztolya & Kocsor, 2005). These schemes are ways of reducing the search space by performing searches that are increasingly more detailed, but that are performed over smaller search graphs. Effectively, it is possible to use a small language model transducer in the first pass and then to rescore the resulting model with a larger language model (Hetherington, 2007). Multipass decoding has the effect of reducing the size of the largest model kept in memory, thus reducing the required resources, but may also have the negative result of dropping some paths that would score highly according to the larger model.

2.3.3.4 Reducing memory requirements in WFSTs

Encoding finite state transducers requires to store all the transitions between states. In some systems, it might be possible to compress the transducer by taking advantage of its structural specificities, thus saving not only memory space but also computation time (Hetherington, 2007). An additional resource to help save space is the quantization of the FST's transition weights. The packing of the transition weights may even help with the decoding speed (since it boosts cache locality), even considering the extra required bit-packing and unpacking operations.

2.4 Summary

In this chapter, a brief survey of the state of the art of speech recognition systems and their applications was presented.

Typical ASR systems feature a pipelined structure which main components (feature extraction, acoustic model, and decoder) are executed sequentially. The acoustic model usually uses either Multilayer Perceptrons, as in *Audimus*, or Gaussian Mixture Models. Often the decoder uses a search space specified using WFSTs, as in *Audimus*, because they offer advantages in terms of flexibility and speed.

Distributed speech recognition (DSR) systems are a promising way to give mobile devices ASR capabilities. Realistic systems need to consider feature compression and error mitigation to overcome network limitations. Several distributed systems currently exist; a robust front-end has been standardized by ETSI. Fixed point feature extraction is a well studied problem since its main components such as the FFT have long been implemented in fixed point devices for many different applications.

Embedded speech recognition is also desirable because it works everywhere, regardless of any networks being available. Several embedded systems have recently been developed, targeting PDAs, phones, car control systems, among other applications. It is more difficult to find information about techniques for building embedded speech recognition systems in the literature, since only recently devices have gained enough processing power to perform embedded speech recognition in tasks of reasonable size. However, the most resource intensive part of embedded speech recognition is the decoding, so standard techniques for limiting search complexity can be used to limit the time spent in recognition at the expense of performance.

3 The Existing System

In this chapter, *Audimus*, the system that served as a basis for the development of this thesis, is presented, along with its various applications. The system's architecture is then explained, namely its main components (feature extraction component, acoustic model, and decoder), that will need to be ported to the device in this work. The above discussion serves as a basis for the two subsequent chapters, that expand further on this subject by analyzing the different intermediate and final prototypes developed.

3.1 The Audimus ASR System

3.1.1 Introduction

Audimus is a high-performance speech recognition engine for the European Portuguese. It can be used for a wide variety of tasks of different complexity, from simple isolated-digit recognition to complex dictation tasks. It can also work in adverse acoustic environments, attempting to recognize both low-quality telephonic speech and noisy speech.

Some of the systems in which *Audimus* has been integrated include:

- A Broadcast News speech recognition system (figure 3.1). This system transcribes the evening news of the main Portuguese public channel (RTP). It uses a very large vocabulary of over 100000 words, which is updated every day with new words, and can achieve word error rates of about 5-10% in speech read by the pivot. The main difficulties of the system reside in spontaneous speech, which often includes disfluencies, and in outside studio conditions, because there is usually much more background noise in these cases.
- Dialog systems for applications such as banking or virtual personal assistants. In these cases, *Audimus* is the component of the system which performs speech recognition on (usually) telephonic speech, being coupled with *Dixi*, L^2F's speech synthesizer, and a dialog management system. Also, the language model is in these cases limited to a grammar or set of sentences which describe the probable utterances emitted by the user.
- Several dictation systems, for narrow subject applications such as radiology, imagiology, nuclear medicine, in which case the language models have around ten thousand words, or general-

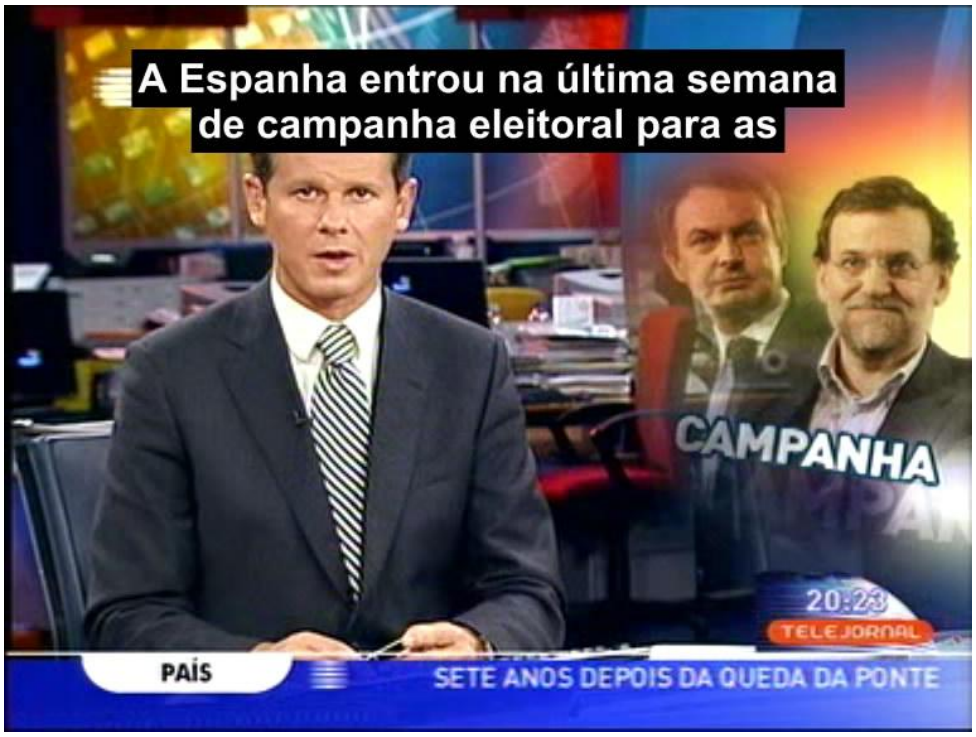
A Espanha entrou na última semana
de campanha eleitoral para as
CAMPANHA
20:23
TELEJORNAL
PAÍS
SETE ANOS DEPOIS DA QUEDA DA PONTE

Figure 3.1: *Audimus* transcribing the RTP evening broadcast news show

purpose dictation systems, with many tens of thousands of words such as the one integrated in the *Microsoft Word* text processor. In the former example, using speaker adapted acoustic models, recognition is almost perfect, achieving around 99% accuracy, whereas in the latter example there is a slightly larger word error rate, in the 2-5% range.

3.1.2 Description of the Audimus system

3.1.2.1 Programming language and development environment

Audimus is written in the C++ programming language. This is one of the programming languages of choice for complex systems with strict efficiency requirements. This is both because its standard ensures that the programmer has complete control over the implications, in terms of efficiency, of language constructs, and because the language has built-in object oriented concepts (encapsulation, inheritance, polymorphism) which help to control complexity when building large systems. Memory management in C++ is also performed by the programmer; this is essential to take advantage of some allocation policies that are made possible by knowledge of the problem domain. For the same reasons, as well as the ready availability of programming tools for embedded devices, C++ was also the language of choice for development in the target device.

Audimus runs in several operating systems, including Windows and Linux. It was written in a way that eases porting to other operating systems, but it is nevertheless easier to port *Audimus* to an OS which is similar to one of the above. This motivated the choice of a version of Windows (Windows Mobile) for development in the device. The Microsoft Visual Studio development environment was used as it is the *de facto* standard for development for Windows Mobile.

3.1.2.2 Audimus' architecture

The architecture of the *Audimus* speech recognition system is shown in figure 3.2.

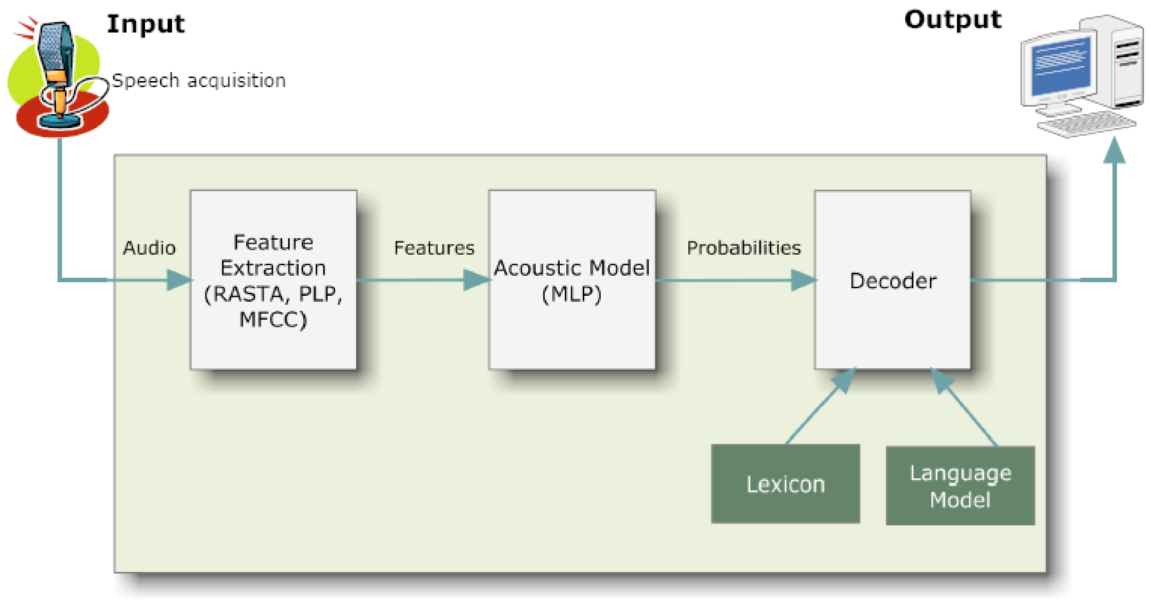

Figure 3.2: *Audimus* as a cascade of processing blocks

The architecture depicted in figure 3.2 is generalized by *Audimus*' *MacroComponent* system, which represents a graph having *components*, the basic unit of the system, as nodes, and connections between components as edges. This enables the flexible specification of different sequences of components, based on an XML file. Porting this system to the target device would, however, be cumbersome, since it would involve porting an XML parsing library. Therefore, the preferred solution was to settle with a text-based configuration file that determined which components were included and which were left out, as well as their parameters.

The *MacroComponent* system uses the producer - consumer paradigm to synchronize pairs of interconnected components, which means that it requires locking to access buffers that are shared among two or more components. The *MacroComponent* system relies on the (*ZThreads - a portable thread library*, n.d.) library to provide these mechanisms, so it was necessary to port the ZThreads system. This system provides a number of high-level synchronization concepts, such as monitors, mutexes or semaphores, that are defined in terms of the primitives of the operating system where ZThreads is compiled. So, porting the system was basically a matter of resolving the slight differences between the thread primitives in Windows Mobile and in the desktop versions of the same operating system.

The PortAudio library (*PortAudio - an Open-Source Cross-Platform Audio API*, n.d.) is used in *Audimus* to capture audio in a device and operating-system independent way. There is no reason not to use this library for the same purpose in Windows Mobile, since a good quality port of PortAudio for this operating system is readily available.

Audimus has a wide variety of feature extraction components, adapted for different conditions that are intended to be recognized. These include modules that compute RASTA, PLP, MFCC, and ETSI features (Meinedo, H. and Neto, J. P., 2003). RASTA features are usually based on PLP features and intend to enhance their performance in noisy conditions via relative spectral filtering. ETSI features are based on MFCCs and designed for robust distributed speech recognition over telecommunication channels. In view of this, PLP was chosen as the feature extraction component, since the goal of this work is to recognize clean speech, not speech significantly distorted by noise, and in the literature MFCCs and PLPs are considered similar in terms of ASR performance.

Audimus is an HMM-MLP hybrid system, combining the temporal modeling capabilities of Hidden Markov Models with the pattern discrimination capabilities of neural networks. Thus, the classifier that was ported was the Multilayer Perceptron existing in *Audimus*.

The previous two components are described in further detail in the next chapter.

Finally, the last component is usually the decoder. This component determines the output of the ASR system by performing a search through a state space. *Audimus* currently uses the WFST approach to search space modeling. WFSTs give a way to decouple the search algorithm from the search space representation. In terms of search algorithm, *Audimus* uses a parallel implementation of the Viterbi algorithm, to take advantage of multiprocessor systems and multiple cores in recent processors. It can also use the A* algorithm, which has the advantage of not committing to the Viterbi hypothesis, but it is generally considerably slower.

The necessary modifications to the decoder are described in chapter five.

Besides these components, *Audimus* employs components to detect speech activity (so that it does not waste resources when there is silence) and to segment the speech signal into sentences, to reduce the strain on the decoder. For that, *Audimus* possesses essentially two components: the endpoint component, which distinguishes speech from silence based on the energy content of the signal and the more complex SNS (speech - non speech) component that uses an MLP classifier to decide whether a segment contains speech. The endpoint component was chosen for this work, for the two main reasons that follow:

- It is easier to port to the target device than the SNS component; the endpoint component is essentially a state machine coupled with a frame energy calculator which averages the squared energy of the samples in the frame, whereas the SNS component is more complex.
- The SNS component requires an extra MLP model to be evaluated at each frame, even when there's silence (albeit this second problem can be overcome by placing an energy detector in front of the SNS component, so that only frames with sufficiently high energy are considered in the SNS component) . This is considerably computationally expensive for a low-resource system.

The main drawback of this choice is that the endpoint component cannot distinguish, for instance, between music and noise: both, above a sufficiently high energy level, are considered speech. In the cases where it arises, this confusion may produce random results and slow down the system, since the decoder will likely have a large number of hypotheses with similar costs to choose from.

4 Building DSR Architectures

The present chapter begins with a description of the initial network speech recognition system. The two distributed systems created as intermediate byproducts of this work are also presented in this chapter, by explaining the changes made to the feature extraction and acoustic components. The performance of these three systems was also tested, in terms of word error rates and execution times; results for each of the systems also appear in this chapter.

4.1 Initial NSR system

Figure 4.1 shows the initial NSR system, used as a basis to port the system to the target device.

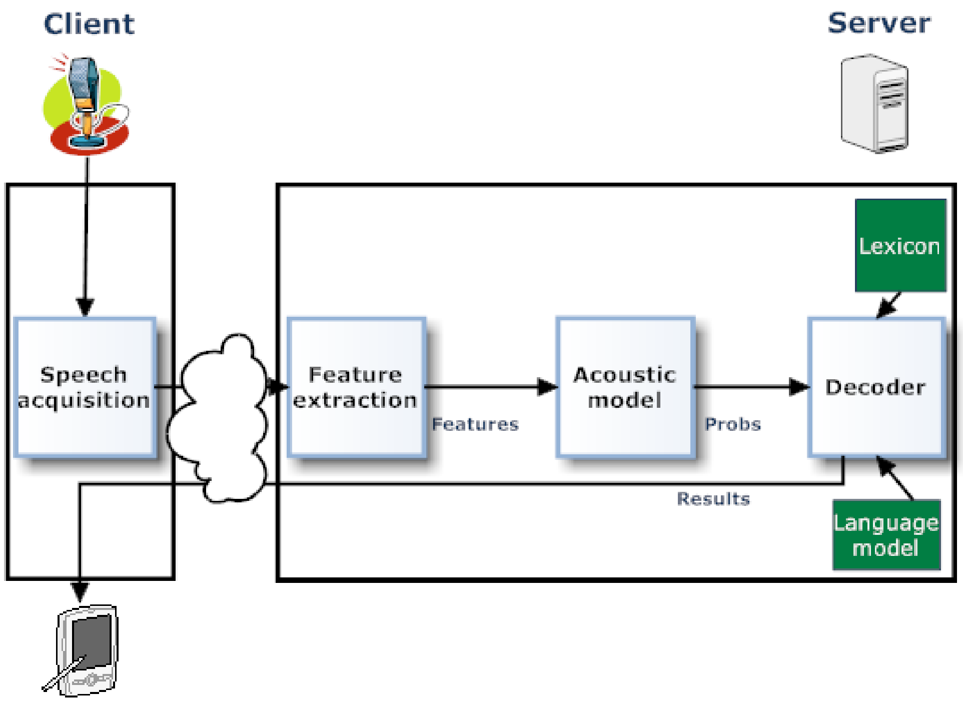

Figure 4.1: The initial NSR architecture of the system

In this figure, all the system's modules are running at the server. The audio is captured in the device and transmitted over the network for processing, and the results of the recognition are sent back to the device across the same network.

Test	Number of Words	Duration
1	317	269s
2	448	306s
3	520	366s
4	521	428s
5	486	332s
Total	2,292	1,701s

Table 4.1: Number of words and duration of each of the five tests.

To create this system, as described in the previous chapter, it was necessary to port the PortAudio and ZThread libraries to the target device. It was also necessary to build the TCP/IP socket communication interfaces in both the target and desktop systems.

4.1.1 Experiments and evaluation of the system

This section presents the experiments that were carried out in order to evaluate the NSR system described above.

Most of the test setup is shared between this and the next chapter, and is therefore not repeated there.

The desktop system used to perform the tests was a 2.4 GHz dual core PC with 2 GB of RAM, running the Windows XP operating system.

4.1.1.1 Test set

The test set consists of 215 sentences drawn from five different tests, totaling 2,292 words. These sentences are from the radiology domain, and each test contains one or more radiology reports. The test set was recorded using two different microphones: a microphone embedded in the device (T1) and a high-quality USB microphone (T2). Both test sets were read by the same speaker, but not simultaneously.

The duration and number of words of each test is shown in table 4.1.

4.1.1.2 Description of the models used

For the tests, three different acoustic models were used, which were:

- Generic model (AM1) - a generic model, trained using a population of different Portuguese speakers from the country's different regions (with an emphasis on the region of Lisbon), and of both genders, that was previously available at L^2F.

- User adapted model (AM2) - a speaker adapted model, trained using 250 sentences with the *Audimus.Dictate* system, using a high-quality microphone.
- User adapted model 2 (AM3) - like the above, this represents a speaker adapted model, but it was trained not only with the sentences recorded above, but also with two sets of 100 sentences. These two sets were made of identical sentences and were recorded simultaneously both with the microphone of the target device and the USB microphone. This microphone has a much lower quality than the headset microphone used to train the sentences in the model above, because it is a small omnidirectional microphone embedded in the PDA.

The lexicon used in the experiments contained 15,190 phonetic transcriptions of 13,141 words for an average of 1.156 transcriptions per word (a few words have more than one transcription). The lexicon was the same throughout the performed tests, because the words were the same in both language models used.

As to the language models, two different language models were used in this work. Both the language models used were n-gram models, where $n = 3$ (i.e, trigram models), trained using a large corpus of normalized radiology texts. The two language models used have the same number of words and differ mainly in their detail; the smaller model is a pruned version of the larger one, where n-grams with low frequency are removed. The larger model (LM1) is approximately 11.0 MB in size (18.2 MB when composed with the lexicon and compressed using the algorithm described in the next chapter), while the smaller model (LM2) is about 5.5 MB in size (9.0 MB when composed with the lexicon and compressed). The radiology task used for the tests already existed at L^2F, so the existing language model was reused as LM1, while the smaller language model (LM2) was built from LM1, by removing the low-frequency n-grams, for this work.

4.1.2 Results

Several tests were designed to measure the performance in terms of WER of this system. This performance is equal to the performance of the baseline system, since all the processing is still done in the PC.

The tests consisted in computing the word error rates for the system, for the two different test sets (with high quality microphone and the microphone embedded in the PDA), using the large language model.

The real time averages are computed by summing the duration of the recognition of each of the tests and then dividing by the total duration of the test set (table 4.1), while the word error rate averages are computed by summing the number of errors in each test and then dividing by the total number of words in the test set (also in table 4.1).

Test	AM1 + USB mic(T1)	AM2 + USB mic(T1)	AM2 + PDA mic(T2)	AM3 + PDA mic(T2)
1	7.89%	2.52%	14.83%	1.89%
2	11.20%	2.23%	19.20%	4.24%
3	8.85%	1.73%	17.12%	3.46%
4	11.70%	2.50%	25.34%	4.61%
5	9.05%	1.44%	13.40%	3.91%
Total	9.86%	2.05%	18.41%	3.75%

Table 4.2: Word error rates for the NSR system. The 2^{nd} and 3^{rd} columns of the table refer to audio recorded with the high quality microphone, while the two last refer to audio from the PDA's internal microphone. Also, in the 2^{nd} column of the table, the generic acoustic model was used; in the 3^{rd} and 4^{th}, the user adapted model was used, and in the 5^{th} column the second user adapted model was used.

Test	Duration	PC	NSR
1	269s / 1.0	125s / 0.46	126s / 0.47
2	306s / 1.0	161s / 0.53	163s / 0.53
3	366s / 1.0	204s / 0.56	205s / 0.56
4	428s / 1.0	207s / 0.48	208s / 0.49
5	322s / 1.0	188s / 0.57	190s / 0.57
Average	1.0	0.52	0.52

Table 4.3: Time taken (and real time factors) to recognize the five tests when using only the PC (column 3), and when transferring the audio from the PDA to the PC (column 4)

Initially, the generic acoustic model lead to an average word error rate of approximately 10%. This model was user adapted with 250 sentences leading to a decrease in the word error rate, which reached 2%. However, when this model was used to recognize speech from the PDA's microphone, the word error rate increased significantly. The second step of adaptation led to a much reduced word error rate, which represents a degradation (in absolute terms) of only 1.70% relative to the audio from the high-quality microphone. Table 4.2 shows these results.

Also, the system's time was measured when the system was entirely in the PC and when it was working as a NSR system, in order to assess the degradation in terms of recognition time caused by the introduction of the network in the system. Table 4.3 presents these results.

From the analysis of table 4.3, it can be seen that this slowdown is not significant (not more than 1-2 seconds). The test network was, as said before, chosen to avoid latency and bandwidth problems; so the small observed degradation in recognition time is mainly due to the initialization of the communication and creation of the *MacroComponent* system.

4.2 First Distributed System

Figure 4.2 shows the first intermediate DSR system that was created as a result of this work.

In this architecture of the system, coefficients computed by the feature extraction component are transmitted across the network, in the client-server direction, instead of speech. The results are still

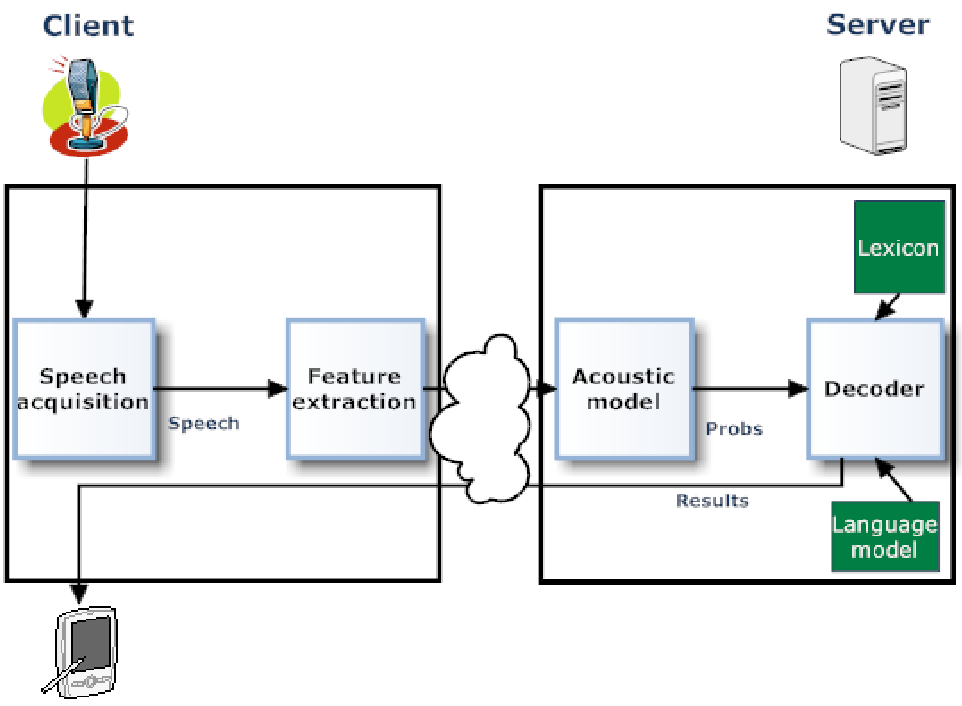

Figure 4.2: The first DSR architecture of the system

returned by the server, in the opposite direction.

To progress towards the goal of this work, it was thus necessary to port the feature extraction component, which as said in chapter 3, is the PLP component, which is one of the feature extraction components of the *Audimus* system. The remainder of this section explains what had to be done to achieve this goal.

4.2.1 PLP Component

The computation of the PLP features starts by computing the FFT. Then the power spectrum is computed, and from it the auditory spectrum . The auditory spectrum is computed from the power spectrum using a set of psychoacoustically based transformations (including, for instance, cube root compression or equal loudness weighting). Finally the LPC coefficients are computed from the auditory spectrum. These calculations use the four basic arithmetic operations (sum, multiplication, subtraction and division) extensively, which must be replaced by their fixed point equivalents, since as has been said before, the target processor cannot perform floating point operations. Also, between many of the subroutines of the component, it is necessary to renormalize the input vector (shift the binary point) so that the range of numbers representable is adequate for the processing done in the subroutine that follows.

The remainder of this subsection refers other, more specific changes that had to be made to the PLP component in order to port it to the target device.

4.2.1.1 Computation of the FFT

The FFT is a central component in the computation of PLP coefficients - and many other feature extraction algorithms - and also represents about 50% of the execution time. The FFT is done in 32-bit fixed point in order to preserve the best possible accuracy. Also, a N-point FFT introduces a gain of N (the output vector has a magnitude which can be up to N times larger). Therefore, to avoid overflow, the input data is shifted right between each two *butterflies* (FFT subroutines). The FFT implementation in *Audimus* was further optimized by replacing calls to trigonometric functions, used to compute the twiddle factors, by table lookups that can be pre-computed.

4.2.1.2 Computation of the Power Spectrum

The power spectrum can be estimated from the complex output of the FFT by calculating the magnitude of each complex number. However, to avoid an expensive square root operation, most applications work with the squared magnitude instead. This increases the range of numbers that must be represented, which complicates a fixed point implementation. The adopted solution was to use a dual fixed point implementation (Ewe et al., 2004), where a single bit selects one of two possible exponents. This ensures that the range is enough to cover the squared magnitude spectrum, while maintaining most of the speed gained by the fixed point approach. One alternative is to use a fast approximation of the magnitude of a vector, but that introduces errors that are up to 10%.

4.2.1.3 Approximation of the cube root function

The auditory spectrum must be equal-loudness weighted and cube-root compressed to account for the characteristics of human audition. The cube-root function must therefore be implemented in fixed-point. One way to solve this problem is to tabulate some of the function's values and then to use linear interpolation to approximate it between those values. The property of the cube-root function f, $f(ax) = \sqrt[3]{ax} = \sqrt[3]{a}\sqrt[3]{x} = f(a)f(x)$, and, in particular, the fact that it is an odd function (i.e., $f(-x) = -f(x)$) mean that we only need to consider the interval between 0 and 1, since any interval between 0 and 2^k can be reduced to the first using a multiplication. We chose to use a table with 256 equally-spaced entries (to avoid using expensive division operations), which leads to an average error of less than 10^{-5}. The resulting implementation was roughly $5 - 10$ times faster than the general-purpose function pow of the C library.

4.2.1.4 Further optimization of operations

In addition to the use of fixed point arithmetic, most ARM processor's division operations are very slow or inexistent, being emulated in software. As a result, whenever possible, division operations (found, for example, in the computation of LPC coefficients from the autoregressive model) were replaced with multiplications by their inverse.

4.2.1.5 Library of transcendental functions

Besides the specific functions mentioned above, a generic library of transcendental functions was implemented. This library was equipped with slower (when compared to the specific cases) but general-purpose fixed-point implementations of transcendental functions such as exponential, logarithm, and trigonometric functions. These functions were useful for computations that were not performed many times and so did not need to be heavily optimized, but that would still be too slow if they used the standard floating-point library.

4.2.2 Experiments and evaluation of the system

In this section, the test setup from the previous system (NSR system) is reused, so that the main changes are the tests performed to the system.

4.2.3 Results

The tests designed in this section were mainly aimed at measuring the degradation of the system in terms of speed and word error rate because of porting the PLP component to the target system.

The language model used for all the computations was still the larger language model.

Table 4.4 shows the word error rates of the system when compared to the baseline system. The acoustic model used was the speaker adapted acoustic model, in the first half of the table, and the second speaker adapted acoustic model, in the second half of the table.

It can be seen from table 4.4 that the word error rate increase in both cases, particularly its absolute value, is limited (less than or equal to 0.51%). This was expected, since the coefficients computed by the fixed point and floating point versions have small discrepancies (values are usually within $10^{-3} - 10^{-2}$ of each other), motivated essentially by truncation and rounding errors, but the largest errors are motivated by the computation of the delta coefficients, which is inherently unstable.

It is also possible to see that in the left half of the table, the relative increase in word error rate is 24.9%; while in the right half it it is only 1.31%. In the first case, the acoustic model is very well adapted

Test	AM2 + USB mic(T1)	AM2 + USB mic(T1)	AM3 + PDA mic(T2)	AM3 + PDA mic(T2)
1	2.52%	3.47%	1.89%	2.84%
2	2.23%	2.90%	4.24%	4.24%
3	1.73%	2.31%	3.46%	3.46%
4	2.50%	2.88%	4.61%	4.22%
5	1.44%	2.06%	3.91%	3.91%
Total	2.05%	2.66%	3.75%	3.80%

Table 4.4: Word error rates for the first DSR system. The left half of the table, that is, the 2^{nd} and 3^{rd} columns of the table, refer to the audio recorded with the USB microphone while the columns 4 and 5 to audio recorded with the PDA's embedded microphone. Columns 3 and 5 refer to the first DSR system, while columns 2 and 4 refer to the NSR system.

Test	Duration	PC	NSR	PLP in PDA
1	269s / 1.0	125s / 0.46	126s / 0.47	129s / 0.48
2	306s / 1.0	161s / 0.53	163s / 0.53	167s / 0.55
3	366s / 1.0	204s / 0.56	205s / 0.56	207s / 0.57
4	428s / 1.0	207s / 0.48	208s / 0.49	211s / 0.49
5	332s / 1.0	188s / 0.57	190s / 0.57	193s / 0.58
Average	1.0	0.52	0.52	0.53

Table 4.5: Time taken (and real time factor) to recognize the five tests when using only the PC (column 3), when transferring the audio from the PDA to the PC (column 4) and when executing the PLP component in the PDA (column 5). Times for the audio recorded with the PDA's microphone

to the acoustic conditions of the high quality microphone, so that a slight change in the coefficients may be significant. In the second case, however, the acoustic model is adapted to a microphone which, being smaller and omnidirectional, captures a larger amount of noise from the environment, so the neural network is less overtrained, meaning that it tolerates a larger amount of noise in the PLP coefficients.

The system's time was compared with the times from the NSR system and the PC-based systems (table 4.5), using speech recorded with the PDA's internal microphone, in order to measure the increase in recognition time.

The average degradation is larger than the difference between PC-only recognition and NSR recognition , mainly because the initial creation of the PLP component is expensive - there are a number of lookup tables, which are computed in floating point before being converted to fixed point. This means that the rest of the embedded system is still running at a small fraction of real time.

4.3 Second Distributed System

Figure 4.3 shows the architecture of the second intermediate system, known as *second distributed system*:

In this architecture - the penultimate before the target embedded system - the client performs both the feature extraction and acoustic model computation portions of the ASR pipeline. The server still sends the results across the network, but in this case the client sends a probability vector. To get one step

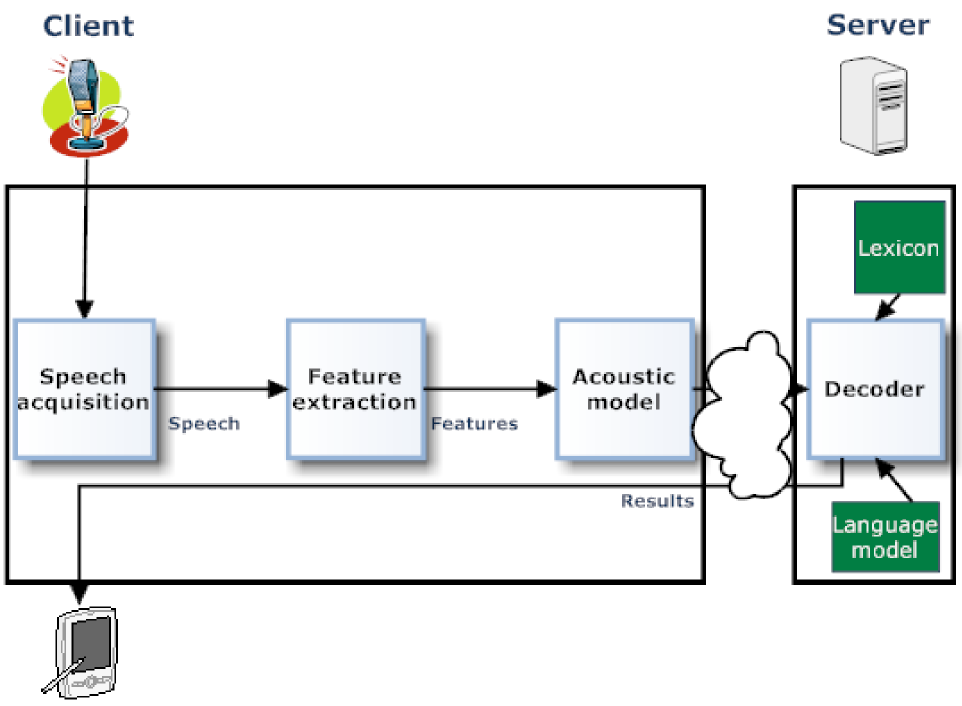

Figure 4.3: The second DSR architecture of the system

closer to the final goal, it is thus necessary to port the acoustic model to the target device.

The acoustic model is in *Audimus*, as said before represented by a Multilayer Perceptron. *Audimus* has a number of components to train Multilayer Perceptrons and classify input vectors, but only the classifier, the *ForwardMLP* component, had to be ported. *ForwardMLP* performs forwarding on an MLP model (i.e., classifies a feature vector returning a vector of probabilities which indicates, for each phoneme, the probability of the feature vector having been produced by that phoneme). The Hidden Markov Models are also part of the acoustic model of each phone, but they are not integrated by the Multilayer Perceptron, but by the WFST decoder, as seen in the next chapter.

4.3.1 ForwardMLP Component

In the ForwardMLP component, it is necessary to compute the output of a Multilayer Perceptron. To that effect, it is necessary to calculate the output of a set of neurons, which is a linear combination of the outputs of the neurons of the layer immediately to the left. This has to be repeated once for each layer in the network; in the case of a single-layer network, it has to be done two times; once from the input layer to the hidden layer, and once from the hidden layer to the output. The activation function, which is usually the sigmoid function, also needs to be evaluated, but that is easily done as a table lookup. The computation of the output of a layer of neurons fits naturally within the framework of matrix multiplication, and the current implementation in *Audimus* uses one of several highly optimized

BLAS (Basic Linear Algebra System) libraries to perform this operation in reasonable time, since the matrices used are, for large networks, very large. Unfortunately, to the best of our knowledge, there is no BLAS system targeting ARM devices. Several possible alternatives and optimizations to the baseline ($O(n^3)$) matrix multiplication algorithm were considered:

- Sub-cubic complexity algorithms are known to exist since the work of Strassen (Strassen, 1969) who developed a $O(n^{2.807})$ algorithm. More recently, exponents for matrix multiplication as low as 2.367 have been achieved (Coppersmith & Winograd, 1982), and it has been conjectured that $O(n^2)$ suffices in general. However, these results are largely theoretical; they are impractical for all sizes of matrices except the largest, since the constant hidden behind the O notation is very large. Also, these algorithms are usually designed for square matrices and adapting them to rectangular matrices requires some effort that will further reduce their efficiency.
- Stochastic matrix multiplication algorithms - some algorithms compute the multiplication of two matrices A (n x m) and B (m x k) by randomly selecting p columns from A and the respective p lines from B, where $p < m$, yielding a result close to the correct result with high probability, and a relative speedup of m/p. The main problem of this approach is that there is still possible that in some cases the result is very far from the correct answer, i.e., the error will stay inside the bounds with high probability. This problem can of course be mitigated by increasing p, but then the advantage of performing less computations is gradually lost.

Another possibility is to propagate significant differences from one layer to the next, but that is only useful in networks with more than one hidden layer, and it was decided to consider networks with just one hidden layer.

The above alternatives to improve the speed of the traditional algorithm were abandoned since it is either very difficult to implement them, with very limited performance gains up to very large matrix sizes (sub-cubic algorithms), or difficult to control the errors produced by the algorithm (stochastic algorithms). It was still necessary to improve the speed of the computations since the *ForwardMLP* component couldn't use up more than a small fraction of the total execution time in the PDA, so it was decided to improve the algorithm's implementation and not the algorithm itself, by reordering the operations (multiplications and sums) that it executes, as explained below.

4.3.1.1 Locality of reference optimizations

The trivial matrix multiplication algorithm uses the processor cache sub-optimally, since to calculate a row of the output matrix, it will read all of the second matrix. This means that none of the entries of the second matrix will be found in the cache and must be retrieved from the slower main memory. Instead of multiplying two matrices in this way, the first and second matrices can be partitioned into blocks, and multiplied "blockwise" (i.e. the traditional matrix multiplication algorithm will be applied as normal, but the elements of the multiplication operation will be matrices, not scalars). This is known as blocked matrix multiplication. The number of arithmetic operations (multiplications and sums) executed by the algorithm will be the same as before. However, if the size of the block is chosen so that it fits in the cache, the number of cache misses will be much lower, causing the algorithm to run much faster; this happens because the latency of the cache is often an order of magnitude larger than the latency of the main memory.

To see this in a different way, let M and N be two square matrices of dimensions $n \times n$, and B a $k \times k$ block, where k divides n, and let $b = n/k$. M and N are thus partitioned into b^2 $k \times k$ blocks. Also, it is assumed that there is one cache level in the target device. Assuming that three blocks (i.e $3k^2$ elements) fit in the cache - so that one multiplication between two blocks followed by one accumulation (i.e, sum of two blocks) can be done without any accesses to the main memory - then each multiplication between blocks requires roughly $2k^2$ memory accesses (assuming, to simplify, that none of the blocks is found in the cache). To compute each of the b^2 blocks of the result, it is necessary to multiply b pairs of blocks, so about $2k^2b^3$ memory accesses are required. It follows that about $2k^2(n/k)^3 = 2n^3/k$ memory accesses are required to perform the matrix multiplication. In contrast to this is the traditional version of the multiplication algorithm that uses three nested loops. To compute one line of the result using that algorithm, it is necessary to read the entire second matrix. Since it does not usually fit in the cache, this means that for every line n^2 accesses will be made to main memory, resulting in a total of more than $n^3 + n^2$ main memory accesses. For a reasonable value of k, $k = 64$, this makes considerably more memory accesses than the blocked version of the algorithm and is therefore slower.

4.3.1.2 Reduction of the network's memory footprint

By quantizing the matrices and all input values to 16 bits, it is possible to reduce the size of the neural network considerably, albeit at a small cost in precision. This not only saves memory but also improves the algorithm's locality (because more data can be made to fit in the cache). It also saves speed, since the target processor is capable of multiplying four 16-bit numbers in one clock cycle using WMMX extensions (which comprise a set of instructions capable of operating on vectors of integers at a time, thus accelerating the processing of the data). It was found, however, that the degree of precision attained by the 16-bit implementation alone was unacceptable, so a block floating point technique was adopted instead. Each set of eight 16 bit values would thus have an associated 8-bit exponent. This did not impact the speed of the multiplication much, since the shift instruction required is fast in most CPUs and, in particular, in ARM processors. Also, as the input of the network consists not only of the current frame but also of the 3 that precede and follow it (so that acoustic correlation with previous frames can be modeled, since HMMs, due to their independence assumptions, are unable to directly model this dependency), each feature frame appears seven times in the first matrix of the first multiplication. This fact was explored to further reduce size by storing each feature frame only once.

4.3.1.3 Coding the matrix multiplication routine in assembly language

The matrix multiplication algorithm above, directly coded in C++, does not achieve the required speed, mainly due to limitations of the compiler in finding the best translation of high level code to apparently unrelated machine instructions. The best instructions are however relatively easy to find for a human programmer, if one takes into account the processor pipeline delays and the result latencies of each instruction. This can be done by performing loop unrolling and loading values into registers ahead of time to maximize throughput. Thus, as a solution to this problem, the entire routine was hand coded in carefully optimized assembly language of the target processor. This solution has drawbacks in terms of flexibility - since the function becomes tied to a particular architecture. Also, the routine is optimized for certain matrix sizes, and becomes less efficient for neural network models of other sizes. But it is necessary to maintain the CPU time spent at acoustic model evaluation at manageable levels. In fact, coding the matrix multiplication routine in assembly language involved considerable effort, due to three main reasons: the need to learn the assembly language of the target device, the relative complexity of the algorithm and the debugging difficulties caused by the lack of support offered by the development environment to assembly programming. However, the benefits were considerable insomuch as testing revealed the assembly algorithm to be approximately 58% faster than the algorithm written in C++.

4.3.1.4 Other optimizations

Like in the feature extraction component, it was necessary to replace all floating point operations with fixed point operations. The activation function of the neural network - the sigmoid function, which is in this case the standard logistic function:

$$f(t) = \frac{1}{1+e^{-t}} \tag{4.1}$$

could be evaluated by reusing the library of transcendental functions produced for the feature extraction component but, since it did not require very high precision, it was implemented via a simple lookup table. This lookup table was improved by storing only half of the values, which is made possible by simply noting the function's symmetry around the point $(0, 0.5)$.

Test	Duration	PC	NSR	PLP+MLP in PDA
1	269s / 1.0	125s / 0.46	126s / 0.47	135s / 0.50
2	306s / 1.0	161s / 0.53	163s / 0.53	184s / 0.60
3	366s / 1.0	204s / 0.56	205s / 0.56	220s / 0.60
4	428s / 1.0	207s / 0.48	208s / 0.49	226s / 0.53
5	332s / 1.0	188s / 0.57	190s / 0.57	210s / 0.63
Average	1.0	0.52	0.52	0.57

Table 4.6: Time (and respective real time factors) taken to recognize the five tests when using only the PC (column 3), when transferring the audio from the PDA to the PC (column 4) and when executing the PLP component and the PLP and MLP components in the PDA (column 5) respectively.

4.3.1.5 Conclusions

The ForwardMLP component, after all the computations had been rewritten in fixed point (but before the optimizations to the matrix multiplication algorithm), was still running near real time. Such a high running time for this component alone would be unacceptable since the decoder, the most time consuming part of the system, would not have CPU time left to run while keeping the system running under real time. The matrix multiplication optimizations performed (both high-level and low level, by coding the routine in assembly) were determinant for the final system, since they brought the running time of the component down to approximately 0.20 xRT, which is a much more manageable fraction of real time which leaves more time available for the decoder.

4.3.2 Experiments and evaluation of the system

This section presents the evaluation of the second distributed system.

4.3.2.1 Tests performed

In these tests the large model (LM1) was used as a language model. Only the speaker dependent model, adapted to the PDA's internal microphone (AM3), was used in these tests, as only audio recorded with this microphone was used.

The time required to process each of the tests using the second distributed system configuration was recorded and compared with the times for other configurations (NSR configuration, for example).

In addition to this, word error rates for the same configuration (second distributed system) were also measured and compared with other configurations.

Test	AM2 + USB mic(T1)	AM2 + USB mic(T1)	AM3 + PDA mic(T2)	AM3 + PDA mic(T2)
1	2.52%	3.47%	1.89%	4.10%
2	2.23%	3.57%	4.24%	4.91%
3	1.73%	2.69%	3.46%	4.03%
4	2.50%	3.45%	4.61%	5.76%
5	1.44%	2.47%	3.91%	4.53%
Average	2.05%	3.10%	3.75%	4.71%

Table 4.7: Word error rates for the second DSR system. The left half of the table, that is, the 2^{nd} and 3^{rd} columns of the table, refer to the audio recorded with the USB microphone while the columns 4 and 5 to audio recorded with the PDA's embedded microphone. Columns 3 and 5 refer to the second DSR system, while columns 2 and 4 refer to the NSR system.

4.3.2.2 Presentation and analysis of results

Table 4.6 indicates that also porting the MLP component led to a slight increase in total recognition time. This does not mean that the combined execution of the endpoint, the PLP and the *ForwardMLP* components is slower than the decoder running in the PC; but it means that the reduced precision of the 16-bit calculations, besides increasing word error rate as explained below, makes a difference in the vector of probabilities which is propagated to the decoder, which then expands more arcs and nodes therefore becoming slightly slower.

As it can be seen in table 4.7, porting both the PLP and MLP components resulted in a relative WER increase of 25.6% and an absolute WER increase of only 0.96% (when transcribing audio recorded in the device). This increase in WER can mostly be attributed to the porting of the MLP component: the absolute increase in WER from the configuration where only the PLP had been ported was 0.91%. The reason for this more significant increase lies in the matrix multiplication algorithm. Since 16 bits of precision were used in all the computations, even considering that the softmax function applied tends to "smooth out" errors, the differences accumulated were in some cases enough to change the decoding decision from a correct to an incorrect one. In the case of audio recorded with the high quality USB microphone, the absolute WER increase is larger, but the MLP component contributes less than the PLP component for this degradation.

5

Final Embedded System

Figure 5.1 presents the architecture of the final embedded system that was built in this work.

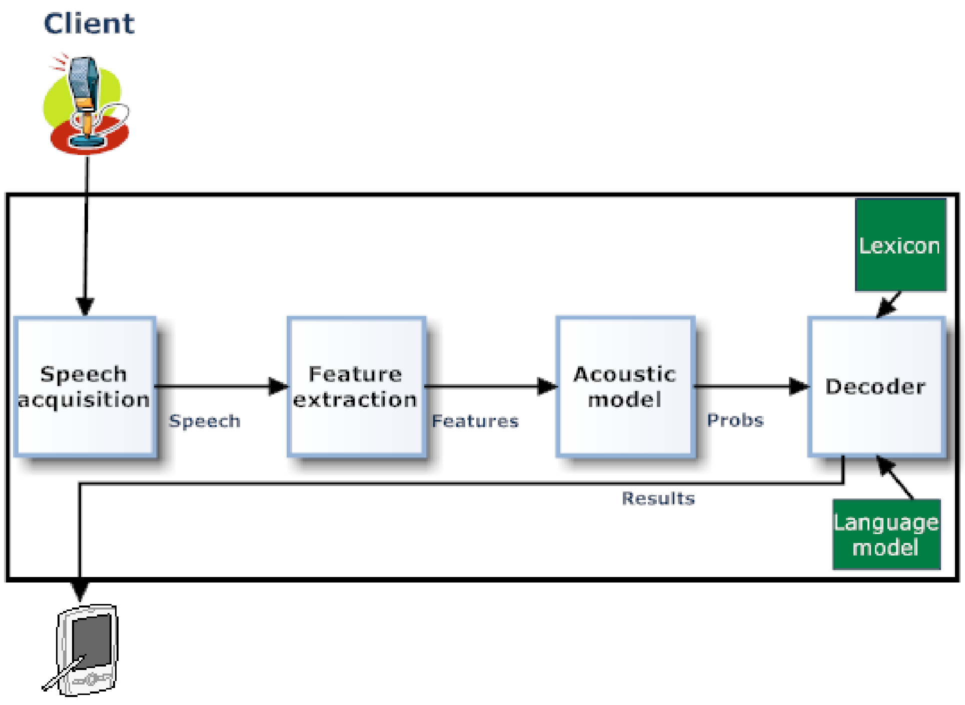

Figure 5.1: The final embedded system

In this system, the speech recognition process (feature extraction, acoustic model, and decoder) is done entirely in the device, so the network and server are no longer necessary.

The current chapter is thus centered on porting the decoder to the device, which completes the porting of the system to the target device. It starts by introducing the existing decoder and its main algorithm, as well as some resource modeling techniques commonly used that justify some specific decisions taken in the optimization stage. In the second section, the optimizations that had to be implemented in order to successfully port the decoder are described.

5.1 The existing decoder

The WFST decoder in Audimus is able to integrate the lexicon, the language model and the subword unit HMMs in several different ways. This means that the user can ask it to work with a search space that is the integration of the language model with the lexicon only, or with a search space that is the composition of the language model, the lexicon and the HMMs. If the phone HMMs are expected to have always the same structures (for instance, in many cases, all the phones are represented by a linear sequence of states, to enforce the phone's minimum duration, with a loop in the last state), it is, for reasons of speed, usually preferable to expand an arc that represents the HMM conceptually in the decoder. On another level, the user can also choose between using a pre-compiled network and one which is built on-the-fly, i.e., the parts of the search space WFST are built from the components when they are necessary. Both approaches have advantages and drawbacks: the static composition approach is faster in terms of CPU cycles consumed than the dynamic composition approach, but the latter usually uses considerably less memory and is more flexible, since it is much easier to change the underlying models dynamically.

In this work, the recognition network is pre-compiled into one large WFST. While memory is surely a scarce resource in the target devices, it is, in this problem, not the most tight constraint - i.e., CPU resources would run out first, as long as some kind of model compression was implemented - as described in the section below. Also, the goal was not to change models on a sentence-by-sentence basis, but to work with a dictation task. Finally, the complexity and overhead of the memory management algorithms associated with dynamic WFST composition (there are many memory allocations followed by garbage collections) makes them inappropriate for use in the target device.

5.1.1 WFST modeling of the lexicon and language model

In this subsection, the modeling of two different knowledge sources in the decoder (the lexicon and the language model) is explained. This is important because some of the characteristics of these models will be used to optimize the compression of the language model with the lexicon, as described later in this chapter. A much more complete discussion of this can be found in Diamantino Caseiro's PhD thesis (Caseiro, 2003).

- **Lexicon** - in *Audimus*, lexica are unweighted transducers from phones to words. This means that all pronunciations of a given word are assigned the same weight. Lexica can be represented as trees, whose leafs are the words in the lexicon and nodes represent prefixes of sets of words, or as graphs where nodes no longer represent prefixes, but suffixes of sets of words. The representation is not very relevant as long as the composition algorithm can produce a fully optimized transducer in both cases.

- **Language model** - in this discussion, only n-gram language models will be considered. N-gram models in *Audimus* are represented as WFSTs by assigning a state to each possible combination of $n-1$ context symbols. Between each pair of states $(a0, a1, \ldots, a_{n-1})$ and $(a1, a2, \ldots, a_n)$ there is an edge labeled with word a_n, with weight equal to the conditional probability $P(a0, a1, \ldots, a_n | a0, a1, \ldots, a_{n-1})$ estimated via counting in the training set (if the n-gram has not been observed in the training set, the edge does not exist). There are also *backoff* edges labeled ϵ from $(a0, a1, \ldots, a_{n-1})$ to $(a0, a1, \ldots, a_{n-2})$ with weight equal to the backoff probability which is computed by some discounting method.

5.1.2 The decoding algorithm

Audimus uses a multithreaded Viterbi decoder, to take advantage of multicore and multiprocessor systems. It is a token-passing implementation of the Viterbi algorithm, augmented, as mentioned above, with the capability to expand the HMMs representing each phone during search. The target devices are usually unable to take advantage of more than one thread of control, since they do not possess multiple cores, so the decoder was simplified to eliminate the overhead of maintaining multiple threads. To control the use of computational and memory resources, the decoder uses standard beam pruning and histogram pruning techniques. These techniques were also employed in the target device, to control the use of CPU at the expense of increasing the WER.

To further increase speed of execution to keep it under real time, and to reduce memory use and consumption, it is possible to reduce the detail of the models used. This can be done in two different ways: via n-gram pruning methods such as Stolcke pruning (Stolcke, 1998), which removes the n-grams that increase the relative entropy less, or by reducing the size of the vocabulary which, as a side effect, eliminates all n-grams containing the pruned words and may cause OOV (out-of-vocabulary) errors. The main goal of these techniques is to find the best possible balance between WER, model size and recognition time, as a last resource after all algorithmic techniques have been explored.

5.2 Main optimizations to the decoder

5.2.1 Compression of the composition of the lexicon with the language model

The composition of the lexicon with the language model usually originates a large transducer, even after being minimized by removing redundant paths. Transducer minimization returns the smallest FST that represents the same transduction between languages as the original FST. However, with the compression method presented herein, it is often possible to reduce the size of the resulting transducer to an acceptable size - below twice the size of the language model transducer. Since decompression of the model is done on-the-fly when the decoder asks for the arcs that originate from a given state in the automaton, the compression algorithm defined must strike a balance between speed of execution and ease of decompression.

The output of this compression algorithm has some redundancy left, in the sense that it could be further compressed using standard algorithms such as Huffman coding or arithmetic coding, in order to encode some fields in the least possible number of bits. However, these algorithms would consume considerable amounts of CPU time, even when decoding only at runtime, for what would likely be only a marginal reward in terms of reduction of the transducer's size.

To store the automaton in memory or in disk, it is necessary to store its initial state, its final states, and its transitions. Storing the initial and final states is trivial (the final states are stored in sorted order so that it is possible to do a binary search to verify if a given state is final). The transitions are first sorted by origin state, in order to support the interface that the compressed transducer must implement, which main function receives a state number and outputs an iterator representing the outgoing transitions from this state. It is also necessary to build an indexing table to quickly find the address of a state in the automaton, due to the variable size of each transition (explained in the next paragraph), which makes it impossible to directly compute the memory address of the state from its number. This indexing table cannot have the same number of entries as the number of states in the transducer, since it would be too large. Instead, the states in the transducer are divided into blocks whose size (i.e., number of states) is a power of two, and these blocks are aligned to an integer number of bytes. The table only stores the addresses of the blocks. To find a state, a lookup is performed in the indexing table which returns the address of the state's corresponding block. Then the states before the desired state in the block are skipped until it is found. This operation, henceforth known as a *state skip*, requires looking into the state's content to determine the appropriate number of bits to skip in each case, and thus represents a compromise between speed and memory usage.

5.2.1.1 Packing the fields of the transitions

As said before, it is necessary to store the transitions in the automaton as compactly as possible. Each transition is defined by its origin and target states, its weight, the input label and the output label. The transitions are sorted by origin state, and as the sequence of states is continuous, it is only necessary to store the number of outgoing arcs for each state. To encode the destination state, the fact that in many cases, its number has a small difference from the origin state, is used. In this case only the difference to the origin state is output, using a smaller number of bits than would be used to encode the full state number. To encode the input label, since it is a phone and there are 40 different phones in the system, 6 bits were used. The output label is the index of a word, so $\lceil \log (numWords) \rceil$ bits are used to represent this word. A large number of transitions have output label ϵ, since only a few edges produce an output word. Finally, the weight is usually stored as a 32 bit float, but it is possible to linearly quantize it to 16 bits without significantly impacting the decoder. It is also useful to note that many transitions in the automaton have weight equal to zero. All of these properties are explored to reduce the average number of bits required per transition and thus the size of the transducer.

5.2.1.2 Encoding linear paths in the transducer

A *linear path* is a sequence of states, connected by arcs, in the transducer, such that each state except the first (the initial state of the path) and the last (the final state of the path) only has one incoming and one outgoing arc (transition). Additionally, arcs in linear paths have as output label ϵ and weight equal to zero.

The automata composition algorithms push the weights in the transitions towards the starting state of the transducer as much as possible, to improve pruning efficiency in the decoding algorithm. These algorithms also push the output labels in the same direction, being produced as soon as the identity of a word is determined. This justifies the existence of a large number of linear paths in the resulting transducer: linear paths usually correspond to sequences of phones common to a given set of words starting at a given language model state, so that there is no difference in language model or lexicon weights spread to the initial and final states of this path, and hence, all the edges in the path have weight zero. That is because it is possible to ignore the lexicon weights since, as explained before, the probability of every pronunciation in the lexicon is equal. Also, because there is no branching in linear paths, the language model weight remains constant between the initial and final states of the path.

Since transitions in a linear path are uniquely determined by their input phone, a linear path is therefore very compactly encoded as its length followed by the sequence of phones that defines it. This can be explored to considerably reduce the size of the transducer; the compact encoding of linear paths immediately led to a reduction of 55% in the average size of the composition transducers.

5.2.2 Cache for faster access

The impact of the transducer compression operations on the speed of the decoder, in particular the need to skip through a large number of states, using bit shifting and masking operations, is considerable. As a solution, a cache was designed, in order to reduce the number of times the compressed FST needs to be directly accessed. The decoder in Audimus already has a cache, but it is not as well optimized for the particular case of automaton composition as the cache that was now designed, and performs iterator copying which is unnecessary in this case (i.e, the cache, when storing iterators, copies them from the original FST to avoid coherence problems. That is however unnecessary in this case : the accesses to the search space FST are managed entirely by the cache system).

Essentially, the cache implements a LRU policy (whenever it is full, and so it is necessary to evict an entry, it chooses the one that has been used the least recently). Additionally, it takes the access pattern of the composition WFST into account, in particular that linear paths make up a significant part of the automaton. The decoder accesses the state having the number immediately above the one that was most recently stored. By always storing the last state that was requested, it is thus possible, in about one half of the cases, to respond to the query using a very small amount of computation, by performing only one state skip. This leads to another important observation: considering the pattern of accesses, the cache only needs to contain the first state in each linear path, since the others will, in most cases, be accessed in sequence after the first and their predecessor will already be in the cache. By doing this, it is possible to use a smaller cache for the same hit rate, which reduces the use of memory space and bandwidth.

The resulting cache is very effective in hiding the access latency of the compressed FST, with hit rates that are usually above 90%. This translates to a reduction of about one order of magnitude in the number of accesses to the compressed FST.

5.2.3 The virtual memory problem

Being the most memory intensive section of *Audimus*, the problem described in chapter 3 (lack of virtual memory space due to restrictions of the OS) was experienced when larger models were experimented with. This was problematic since it was intended to use the full physical memory of the device.

The first attempt to solve this problem hinged on the fact that, since one process cannot directly access more than 32 MB of virtual memory space, the program could be divided into several different processes, with separate 32 MB address spaces. This solution is not simple to implement in general, since many programs have complex interdependences among all of their components, which are not connected through a single point. However, this is made easier in the case under consideration due to the "pipelined" nature of the ASR system, where the output of one component (feature vectors, probabilities, etc) is fed directly into the next. Nevertheless, this scheme has important disadvantages: the use of several separate processes consumes a significant amount of system resources, mainly in terms of memory; synchronization among processes is more cumbersome than among threads, since processes do not share the same address space, and finally, and perhaps most importantly, the solution is not completely scalable, since there is a single component - the decoder - that uses the vast majority of the physical (and virtual) memory of the system, so that this component remains the bottleneck if, for example, the model is larger than 32 MB.

This led to the consideration of an alternative solution - based on a shared memory region, common to the address space of all processes. The function *VirtualAlloc*, which allocates virtual pages in a process's address space, can be directed to allocate pages from this shared pool, that do not belong to the 32 MB virtual address space of the process. There are a few drawbacks to this approach, namely, that data stored in a shared region is accessible to all processes and can thus be read by every other process in the system which can be undesirable in terms of security, and that this shared region is only 128 MB in size, so that occupying a large portion of it may make interprocess communication harder. These disadvantages were, however, regarded as minor when compared to the ease of implementation and elegance of the solution, since it would not require any major changes to the code. As a result, this was the adopted solution.

5.2.4 Optimizations in the decoder

Further optimizations of memory usage in the decoder included reanalyzing the data structures of the decoder in order to save space.

The Viterbi decoder employs a *word list* data structure in order to store the required lists of words. Each node of this structure contains, among other information, the corresponding lattice states in case it is necessary to generate a lattice as the result of the search, as well as information used to compute confidence measures (that indicate how confident one is in a given recognition result) . These can be removed from the structures used in the target device. In this and other data structures of the decoder, it is possible to reduce the width of some fields, which are usually small integers, so their range is not fully used. This is not done in the PC version of the code because, in this case, larger amounts of RAM and cache memory are available. This makes the speed of access to these structures a more important factor than the compactness of the data structures; using smaller fields can, in some cases, disrupt the alignment and thus the access speeds.

Other related optimizations in the area include reducing the memory occupied by the tokens by removing the time instant to which they refer. This value is important for alignment tasks and the computation of confidence measures, but these two tasks are not performed in the target system.

The original decoder already performed beam pruning and histogram based pruning to limit search complexity at the expense of some recognition performance. These techniques are parameterizable and as such they did not require any changes, but were used with narrower beams and a smaller maximum number of active states.

Multipass search, in particular, two-pass search, was considered as an option to speed up the decoder, but it dropped one of the fundamental requirements of the system : that it be online, as much as possible. The direct application of a multipass decoding strategy would require that all of the utterance had been read before the second pass could start, causing a significant delay when obtaining the system's output.

5.3 Experiments and evaluation of the system

This section presents the results obtained for the final embedded system. The language and acoustic models used are the same that were described in the previous chapter.

5.3.1 Tests performed

To assess the performance of the final embedded system, a sequence of tests was prepared. These tests were sequenced in such a way as to only change one parameter of the configuration at a time.

In all tests of this system the audio used was recorded with the microphone in the PDA.

The first four tests were executed in the embedded system. The tests all used the large language model (LM1), except for the fourth, which used the small language model (LM2). The first test used the generic acoustic model (AM1), the second test the speaker adapted acoustic model (AM2), and the third and fourth tests the speaker and microphone adapted acoustic model (AM3). This was intended to assess the impact in recognition performance of different acoustic models.

Tests five and six of this set were executed in the NSR system. Both use the speaker and microphone adapted acoustic model (AM3). Test five uses the small language model (LM2) while test six uses the large one (LM1). These tests are intended to compare the baseline word error rates and times with those from the embedded system.

5.3.2 Presentation and analysis of the results

Tables 5.1 and 5.2 present times of recognition and real time factors for the configurations presented in the above subsection.

The two following tables present times for each of the six tests. Table 5.1 presents times for the first four tests, while table 5.2 presents times for the fifth / sixth tests.

Tables 5.3 and 5.4 present error rates for the same configurations as in tables 5.1 and 5.2, respectively.

From these results, and comparing them with the ones from the tables of chapter 4, it is possible to note the following points:

- The average WER of the best embedded configuration (large language model, and adapted acoustic model AM3) is 7.77% with a real time factor of 0.71. This value of WER compares favorably with those in table 4.2, in which processing power is unrestricted: it is lower than the column in which the generic acoustic model is used to recognize clean speech. It is also only 4.03% higher than the value of the best configuration (column 2) of table 5.4, in absolute terms.

Test	AM1 + Large LM	AM2 + Large LM	AM3 + Large LM	AM3 + Small LM
1	250s/0.93	202s/0.75	165s/0.61	191s/0.71
2	334s/1.09	288s/0.94	224s/0.73	278s/0.91
3	432s/1.18	391s/1.07	275s/0.75	359s/0.98
4	459s/1.07	406s/0.95	316s/0.74	394s/0.92
5	335s/1.01	279s/0.84	233s/0.70	285s/0.86
Average	1.06	0.92	0.71	0.89

Table 5.1: Duration and real time factors in the recognition of the four test cases. The acoustic model is the generic acoustic model (AM1) in the first system, the speaker-dependent model (AM2) in the second, and the speaker-dependent model adapted to the embedded device's microphone (AM3) in the third and fourth.

Test	Small LM	Large LM
1	135s/0.50	126s/0.47
2	169s/0.55	163s/0.53
3	217s/0.59	205s/0.56
4	256s/0.60	208s/0.49
5	193s/0.58	190s/0.57
Average	0.57	0.52

Table 5.2: Duration (in seconds) of the recognition of each of the five test cases, and respective real time factor. Both tests were executed in the NSR system, and use the acoustic model adapted to the PDA's microphone (AM3).

Test	AM1 + Large LM	AM2 + Large LM	AM3 + Large LM	AM3 + Small LM
1	18.30%	16.72%	4.10%	6.62%
2	21.65%	16.07%	5.80%	10.04%
3	25.77%	13.65%	6.73%	5.96%
4	40.38%	25.58%	13.65%	14.23%
5	15.43%	13.99%	6.79%	8.64%
Average	25.06%	17.33%	7.77%	9.30%

Table 5.3: Word Error Rates for the same configurations as presented in table 5.1

Test	Small LM	Large LM
1	2.88%	1.89%
2	3.61%	4.24%
3	4.35%	3.46%
4	6.78%	4.61%
5	4.96%	3.91%
Average	4.68%	3.75%

Table 5.4: Word Error Rates for the same configurations as presented in table 5.2

- As it would be expected, the use of acoustic models AM1 and AM2 is not good enough, mainly in terms of word error rates (25.06% and 17.33% respectively). This confirms that the acoustic model needs to be adapted both to the user and microphone in order to obtain good performances.

- The use of the small language model seems to have limited influence on the system's recognition errors; the WER is degraded 1.53%, in absolute terms, in the embedded system. However, using the smaller language model also leads to an *increase* in execution time; the real time factor in the embedded configuration increases from 0.71 to 0.89, and in the NSR configuration this factor increases from 0.52 to 0.57. In other words, this means that the fact that the larger model has less locality is more than compensated for by its better adequation to the speech segment being recognized, leading to a more focused and therefore faster search. Consequently, it seems to be preferable to use more detailed language models if enough memory to store the models is available.

- Test 4 shows the highest overall error rates of the test set, in many cases much higher than those of the remaining tests, reaching the absolute value of 40% in column 2 of table 5.3. In fact, this test seems to contain a set of sentences made up of more complex words. It also contains short phrases that are sometimes discarded by the endpoint component for being too short. The obvious solution would be to increase the threshold for the shortest acceptable speech block in the endpoint component, but that also leads to an increase in the detection of spurious non-speech fragments as speech.

These results show that the largest absolute increase in word error rate of all components occurs when porting the decoder. This would be expected, not because complex fixed point operations are performed in the decoder (since it performs mostly sums), but because the decoder is the most complex part of the system and so aggressive pruning had to be applied to control its time complexity. However, the absolute increase in word error rate in the best configuration, relative to the NSR system, is only about 4% and the final system still runs under real time (0.71 xRT). This means that the results confirm the feasibility of using a large language model in the system, with good results, provided that the acoustic model is speaker adapted.

6 Conclusions

This chapter presents the concluding remarks of this thesis, summarizing the work that was done. It ends by presenting some work that remains to be done in the future.

6.1 Final Remarks

In this work, *Audimus*, described in chapter 3, was progressively ported to a target device, using a radiology task with a vocabulary of 13161 words to test each of the intermediate systems that was produced. The mobile device used in the tests was a 520 MHz PDA with 64 MB of RAM running Windows Mobile 6 and with no floating point capabilities.

The starting point for this work was a NSR system (chapter 4), running over a wireless TCP/IP network. This initial system, running with a generic model and with audio recorded with a high quality microphone, attained a WER of 9.86%. The generic acoustic model was speaker adapted, leading to a WER of 2.05%. Then, an attempt to reuse the acoustic model with audio recorded with the PDA's internal microphone led to a very large increase in WER (to 18.41%). So, the acoustic model was further adapted to the internal microphone in the PDA. Accordingly, the WER dropped to 3.75%, which represents an absolute increase in WER of only 1.70%, when compared with the adapted system running in the PC with the speaker adapted model. Then the PLP component was ported to the target device, as mentioned in subsection 4.2.1, and the word error rate did not increase significantly (only 0.05% in absolute terms). Also, the MLP component was ported to the PDA (subsection 4.3.1), with an absolute increase in WER of 0.91%. Porting these two components did not cause a significant increase in WER. Finally, the decoder was ported to the target system, creating the final embedded system (chapter 5) and giving the most significant absolute increase in WER, from 4.71% to 7.77%. The system ran at 0.71 xRT.

This final embedded system can be compared with the baseline system in the PC. Both systems perform the same recognition task and are online, that is, they output recognition results as soon as it is possible (this is a desirable quality in dictation tasks for it gives the user immediate feedback). When both systems are used with a speaker adapted model recognizing the same audio, the baseline system has a WER of 3.75%, while the embedded system has a WER of only 7.77%. This absolute increase in WER is small when the discrepancies in processing power are factored into consideration, since the PC is an order of magnitude or more faster than the device. In terms of the percentage of time occupied by each component, in the baseline system the PLP component took up about 4% of the total time and the acoustic model about 12% of the execution time, with the decoder using up the remaining 84%. In contrast, in the final system the PLP component occupied 10%, the acoustic model 28%, and the decoder only 62%. This increase in proportion of the PLP and acoustic model components is natural since, even if these components have been optimized for speed, the improvement is not enough to counterbalance the difference of processing speeds between the desktop system and the target device. It also shows why the decoder had to perform more aggressive pruning, justifying the increase in word error rates.

From the above summary of the obtained results, it is possible to conclude that the goals of this thesis were attained, in that *Audimus* was successfully ported to a PDA, achieving good performance in the task used to test it. It is hard to directly compare its performance with state-of-art embedded systems due to the different characteristics of the devices used in each work. However, most of the surveyed systems that used state-of-art systems had vocabularies between 2000-15000 words, worked near or above real time, were speaker adapted, and had word error rates in the 10-15% range. This is comparable to the performance achieved by the system described in this thesis.

It is also important to note that the intermediate systems that were produced can be an interesting alternative to the final embedded system. They present lower word error rates and enable using larger tasks, since they transfer the most complex part of the processing to a computer with enough resources. Given this and despite their disadvantages - the increased latency in response and the requirement that a suitable network be available within reach of the device - distributed systems may well be the best option in the cases where processing power is too limited or the ability to use a very large language model is the determinant factor.

6.2 Future work

In this section, some of the lines that can be explored in future work are described.

6.2.1 Integration of some improvements in the desktop *Audimus* system

The work that has been done, in terms of reducing memory consumption and CPU time used, is mostly independent of the system where it is deployed. This means that it may be possible to employ it to reduce memory consumption in the desktop system, enabling the use of larger tasks. Integrating some of these improvements in the baseline system, such as the compression of the composition of the lexicon with the language model would, nevertheless, require a more careful analysis of how they would scale for larger models and how they would function in the presence of multithreading, especially in the case of the cache, which is not thread safe.

6.2.2 Improving the distributed systems

The task of creating robust distributed systems was left outside the scope of this thesis, since they arised as a byproduct of its main goal: to create a completely embedded system. In this sense, it would be important to investigate:

- other feature extraction components, in special their adequation to more realistic and challenging environments. Background noise, such as people talking, cars running, or room reverberation is common in most realistic environments, and must be addressed if system performance is to degrade gracefully under these circumstances.
- it would also be interesting to apply feature compression and error mitigation techniques in order to adapt the system to some types of communications networks that are commonly used by mobile devices. Many of these networks are congested and therefore lose or corrupt a large number of packets. This requires compressing the feature vectors to make the most of the available bandwidth, as well as minimizing the impact of any packet loss on the recognition performance.

In addition to what was mentioned in the above paragraph, it would also be interesting to look into the possibility of porting and using the SNS component instead of the endpoint component for sentence segmentation and speech detection, to see if the improvement in robustness would be significant.

6.2.3 Porting the system to other OSs and devices

The current system only works under the Windows Mobile operating system. It is, however, anticipated that porting it to other mobile operating systems will be relatively easy, since the system uses few OS-dependent constructs or functions. Porting the system would thus essentially require to port the PortAudio and ZThreads systems to the target OS, which would not be difficult given the fact that most modern operating systems for embedded devices support threads.

Also, the system is currently dependent on the device's processor; in particular, the efficient version of the matrix multiplication algorithm of the *ForwardMLP* component will only work on processors featuring the Wireless MMX extensions (there is a less efficient version, written solely in C, of the matrix multiplication that will work on all processors, but as it stands, this version is about 58% slower than the assembler version, which is unacceptable for most purposes). This can be avoided if compiler technology advances enough so that compilers can understand how to optimize effectively for all target architectures, in particular how to use some vector instructions of the processor effectively. But it would also be interesting to investigate how to automatically generate good quality code for a set of ARM based architectures sharing similar DSP instructions but with different pipelines and instruction scheduling rules.

6.2.4 Confidence measures

The obtained system does not compute confidence measures, since their calculation introduced non-negligible overhead in the system. The computation of confidence measures does not seem to be a priority, since in a dictation application, because the system is online, the user has immediate feedback on whether the text they have spoken has been correctly recognized by the system. However, if the user speaks a large amount of text in a short period of time, they might not remember exactly what they have spoken, so confidence measures might help them not to be confused with the output of the system. Additionally, confidence measures are extremely useful in a command-based system, because they can help the system not to execute commands that have been recognized with confidence below a given threshold.

For these reasons, it would be important to consider, in future work, the introduction of confidence measures into the system.

Bibliography

Albesano, D., Mana, F., & Gemello, R. (1996). Speeding up Neural Network Execution: An Application to Speech Recognition. In *Proceedings of the IEEE Workshop on Neural Networks For Signal Processing VI.* Kyoto, Japan.

Asahi Kasei VORERO. (n.d.). `http://www.asahi-kasei.co.jp/vorero/en/`.

Bahl, L., De Gennaro, S., Gopalakrishnan, P., & Mercer, R. (1993). A fast approximate acoustic match for large vocabulary speech recognition. *IEEE Transactions on Speech and Audio Processing*, *1*(1), 59–67.

Boulard, H., & Morgan, N. (1994). *Connectionist Speech Recognition - A Hybrid Approach.* Massachussets, EUA: Kluwer Academic Publishers.

Caseiro, D. (2003). *Finite-state methods in automatic speech recognition.* PhD Thesis, Instituto Superior Técnico, UTL, Lisboa, Portugal.

Coppersmith, D., & Winograd, S. (1982). On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*(11), 472–492.

Digalakis, V., Neumeyer, L., & Perakakis, M. (1998). Product-Code Vector Quantization of Cepstral Parameters for Speech Recognition over the WWW. In *Proceedings of the ICSLP 1998.* Sydney, Australia.

ETSI ES 201 108 v1.1.2 distributed speech recognition (front-end feature extraction). (2000).

Ewe, C. T., Cheung, P. Y. K., & Constantinides, G. A. (2004). An Efficient Alternative to Floating Point Computation. In *Proceedings of FPL 2004.* Antwerp, Belgium.

Fabregas, V., & Alcaim, A. (2007). Features Interpolation Domain for Distributed Speech Recognition and performance for ITU-T G.723.1 CODEC. In *Proceedings of Interspeech 2007.* Antwerp, Belgium.

Gosztolya, G., & Kocsor, A. (2005). Speeding Up Dynamic Search Methods in Speech Recognition. In *Proceedings of the IEA / AIE 2005.* Bari, Italy.

Hedelin, P., & Skoglund, J. (2000, July). Vector quantization based on Gaussian mixture models. *IEEE Transactions on Speech and Audio Processing*, *8*(4), 385–401.

Heisterkamp, P. (2001). Product Level Speech System for Mercedes-Benz Cars. In *Proceedings of the HLT 2001.* San Diego, USA.

Hetherington, I. (2007). PocketSUMMIT: Small-Footprint Continuous Speech Recognition. In *Proceedings of Interspeech 2007.* Antwerp, Belgium.

Hsiao, R., Venugopal, A., Kohler, T., Zhang, Y., Charoenpornsawat, P., Zollmann, A., et al. (2006). Optimizing components for handheld two-way speech translation for an english-iraqi arabic system. In *Proceedings of the ICSLP 2006.* Pittsburgh, USA.

Huggins-Daines, D., Kuhmar, M. C. A., Black, A. W., Ravishankar, M., & Rudnicky, A. (2006). PocketSPHINX: A Free, Real-Time continuous speech recognition system for hand-held devices. In *Proceedings of the ICASSP 2006.* Toulouse, France.

IBM Embedded ViaVoice. (n.d.). `http://www-306.ibm.com/software/pervasive/embedded_viavoice/`.

Intel. (2004, October). *Intel PXA27x Processor Family Developer's Manual.*

Kiss, I. (2000). A comparison of distributed and network speech recognition for mobile communication systems. In *Proceedings of the ICSLP 2000.* Beijing, China.

Knill, K. M., Gales, M. J. F., & Young, S. J. (1996). Use of Gaussian Selection in Large Vocabulary Continuous Speech Recognition using HMM's. In *Proceedings of the ICSLP 1996.* Philadelphia, USA.

Konami's LifeLine. (n.d.). `http://www.konami.com/lifeline`.

McTear, M. F. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Survey*, *34*(1), 90–169.

Meinedo, H., Caseiro, D. A., Neto, J. P., & Trancoso, I. (2003). AUDIMUS.media: a Broadcast News speech recognition system for the European Portuguese language. In *Proceedings of the PROPOR 2003* . Faro, Portugal.

Meinedo, H. and Neto, J. P. (2003). Automatic speech annotation and transcription in a broadcast news task. In *Proceedings of the ISCA Workshop on Multilingual Spoken Document Retrieval.* Macau, China.

Mitra, A., & Chakraborty, M. (2002). An Efficient Block-Floating-Point Implementation of Fixed Coefficient FIR Digital Filters. In *Proceedings of the National Conference On Communications.* Bombay, India.

Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite state transducers in speech recognition. *Computer Speech and Language*, *16*, 69–88.

Nuance Vocon. (n.d.). `http://www.nuance.com/vocon/`.

Paul, D. B. (1992). An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with Stochastic Language Model. In *Proceedings of the ICASSP 1992* . San Francisco, USA.

Paver, N. C., Aldrich, B. C., & Khan, M. H. (2003). Intel wireless MMX technology: a 64-bit SIMD architecture for mobile multi-media. In *Proceedings of ICASSP 2003.* Hong Kong, China.

PortAudio - an Open-Source Cross-Platform Audio API. (n.d.). `http://www.portaudio.com/`.

Rabiner, L.R. (2000). A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in Speech Recognition* (pp. 267–296). San Francisco, USA: Morgan Kaufmann.

Ryan, M. S., & Nudd, G. R. (1993). The Viterbi algorithm . In *Warwick Research Report RR238.* Conventry, England.

Singh, G., Panda, A., Bhattacharyya, S., & Srikanthan, T. (2003). Vector quantization techniques for GMM based speaker verification. In *Proceedings of the ICASSP 2003.* Hong Kong, China.

Stolcke, A. (1998). Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop 1998.* Landsdowne, USA.

Strassen, V. (1969). Gaussian Elimination is not optimal. *Numer. Mathemat.*(13), 354–356.

Tan, Z.-H., Borge, L., & Dalsgaard, P. (2004). A Comparative Study of Feature-Domain Error Concealment Techniques for Distributed Speech Recognition. In *Proceedings of the ITRW on Robustness Issues in Conversational Interaction.* Norwich, UK.

Young, S. (1995). Large vocabulary continuous speech recognition: A review. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding.* Snowbird, Utah.

Young, S. (1998). Acoustic Modelling for Large Vocabulary Continuous Speech Recognition. In *Proceedings of the NATO Advanced Study Institute Conference on Computational Models of Speech Pattern Processing 1998.* Il Ciocco, Italy.

Zhang, W. Q., He, L., Chow, Y., Yang, R., & Su, Y. (2000). The study on distributed speech recognition system. In *Proceedings of the ICASSP 2000.* Istanbul, Turkey.

ZThreads - a portable thread library. (n.d.). `http://zthread.sourceforge.net/`.