A Platform of Distributed Speech Recognition for the European Portuguese Language

João Miranda, João P. Neto

L^2F - Spoken Language Systems Lab / INESC-ID,
Instituto Superior Técnico / Technical University of Lisbon
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
{jrsm,Joao.Neto}@l2f.inesc-id.pt

Abstract. In this paper we present a Distributed Speech Recognition system for the European Portuguese based on the *Audimus* system, that can be used in embedded systems such as PDAs. The obtained system has no significant degradation in what concerns word error rates or increased latency in processing, and only a small increase in word error rate when the audio is recorded using the microphone in the device.

Key words: Automatic Speech Recognition, Distributed Speech Recognition, Embedded Systems

1 Introduction

Audimus [1, 2] is an Automatic Speech Recognition (ASR) system, tailored for the Portuguese Language. It is used for a variety of tasks, being parameterized by models that are adapted to each specific task. Since *Audimus* is prepared to work with tasks which involve large vocabularies, some of its models are very large and processing them requires large amounts of memory and processing power, so it has been designed to run on computers that can accomodate these requirements.

The existence and increasing popularity of a large number of embedded devices, such as PDAs and cell phones, combined with the difficulties of entering data using traditional input devices such as keyboards (due to their small size), makes it desirable to have a speech interface running on those devices. However, they have resources that are very limited when compared to desktop workstations. For example, their processors are usually RISC processors, which besides being much slower than workstation CPUs, often lack a floating point processing unit, so that arithmetic calculations must be rewritten as calculations involving only integers. Also, while it is not uncommon for modern workstations to have a few gigabytes of memory, the devices being targeted rarely have more than a few tens of megabytes available for processing.

One way to reduce the processing load on the embedded device would be to transfer all the processing to the server, in what is known as Network Speech Recognition (NSR), but for many networks the bandwidth is reduced and therefore it becomes necessary to employ a low bitrate coding, thereby reducing the

quality of the speech received at the server. These problems are overcome by Distributed Speech Recognition (DSR)[3, 4] systems, which process the signal at the client device, in order to reduce the dimensionality of the set of features that must be transmitted - basically, the speech recognition system is divided into two parts:

- the front-end, which runs at the client (i.e., the embedded device) and performs the feature extraction and acoustic model calculation
- the back-end, which runs at the server and executes the most time and memory consuming module of the system - the decoder.

In this work, we intended to port the *Audimus* system to embedded devices. We have large vocabulary (over 100K words) models for several tasks in the European Portuguese Language, and wanted to be able to use these models in the resulting speech recognition system. To fulfill these requirements, we decided to adopt the DSR paradigm.

In the next section, the state-of-art in Speech Recognition techniques is briefly described, with a particular emphasis on the *Audimus* system. In the third section, the architecture of the implemented DSR system is described and, in the fourth section, some of the implementation options are discussed. Finally, in the fifth section, the results obtained with the system are presented.

2 Current Speech Recognition System

Figure 1 shows the architecture of a generic speech recognition system.

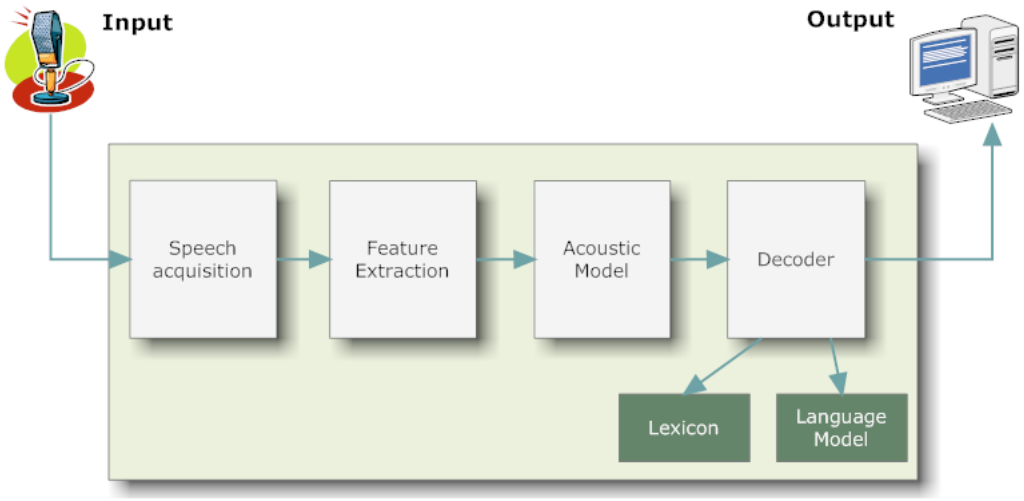

Fig. 1. A generic ASR system architecture as a cascade of processing blocks

State-of-art speech recognition systems generally employ some combination of these components. The speech acquisition module performs the capture of the speech that is to be recognized; it can be done in several ways, using a simple microphone, a microphone array, or a multimedia file.

The feature extraction component performs a form of transformation on the speech signal, producing a reduced set of features that are intended to better represent it. It is implemented using signal processing techniques, that essentially try to remove the signal information that does not help to identify the words being spoken. In particular, *Audimus* has a rich set of components, that calculate ETSI, MFCC, PLP, and RASTA features [2], which are based on different algorithms.

The acoustic model estimates a set of *posteriori* probabilities: for each phone, the probability of it having been generated by the observations (that are the output of the feature extraction component). Currently the two main approaches to solve this problem are to use a Gaussian Mixture Model, which assumes that each feature vector is generated by a distribution which is a mixture of Gaussians, and a Multi-Layer Perceptron. *Audimus* uses the latter.

Finally, the decoder is the module that, based on a language model (which is a description of "how likely" is each sequence of words in the language) and on a lexicon, finds the sentence that is the most likely to have been uttered by the speaker. To approach the decoding problem, some systems use decoding strategies based on the Viterbi or stack decoder [5] (also known as the A* algorithm). *Audimus* uses the WFST-based approach to Large Vocabulary Speech Recognition [6].

In addition to the above structure, which is common to almost all state-of-art ASR systems, *Audimus* also has components to segment a speech signal into sentences, and to classify a frame of audio into speech / non-speech [2] (to reduce the load when there is no useful speech to process).

3 System Architecture

The architecture of the Distributed Speech Recognition (DSR) System we defined is depicted in Figure 2.

3.1 Network

The front-end and the back-end of the system are connected through a network, which in our case is a wireless LAN, so that the device can be used freely. This model can be generalized to other types of LANs or even to a service model concept over the Internet. The current paper (and Figure 2) only considers, however, networks where loss, errors or duplication of packets do not occur.

3.2 Application on the Client

In the client, speech acquisition can be done either using the device's embedded microphone or with a Bluetooth Microphone, which enables higher quality audio

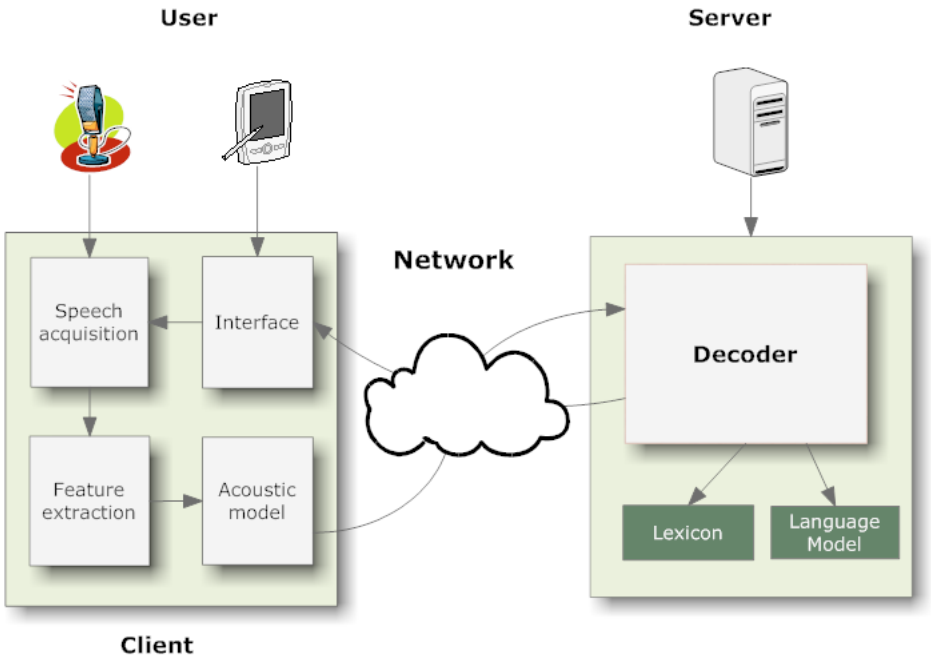

Fig. 2. Architecture of the DSR System

capture, since the speech source is closer to the microphone. We used the PortAudio Library [7] to capture the audio in a system-independent way, of which there is already a version for Windows Mobile, the target OS.

We also developed a simple interface in which the user can specify the parameters of the distributed recognition system (the IP of the server, models - all sizes and types of the *Audimus* system being supported) and the processing the user wants the server to do, speak to a dictation interface, and see the results, as can be seen in Figure 3.

3.3 Server Configuration

Initialization of communication Communication between client and server is initiated by the client, as it is usual in the client-server model. The server must therefore be running before a connection from the client is attempted. The client must provide the server's IP address and the port where it is listening to incoming requests, as can be observed in figure 3. The server accordingly creates a new *Audimus* process to handle the current request.

Model Synchronization As seen in Figure 3, the models used for recognition can be specified before recognition using the interface in the client. The server

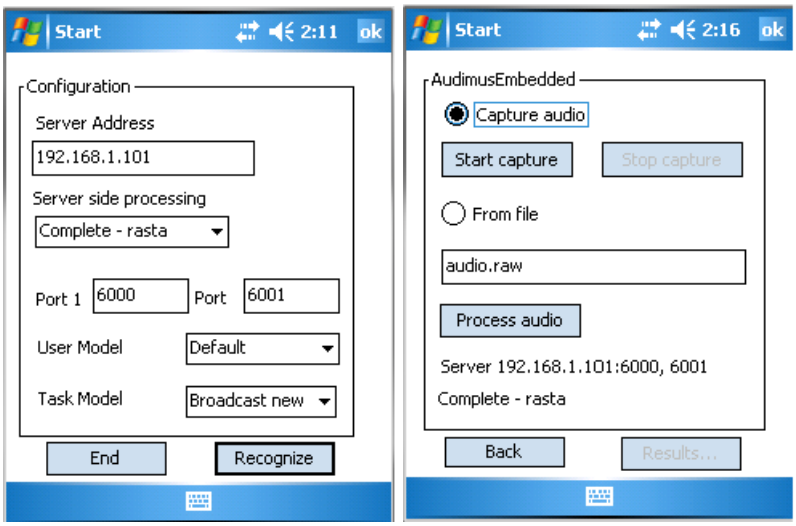

Fig. 3. The configuration screen, on the left, and the main speech recognition screen on the right.

application manages this change by using the *Audimus* API appropriately to change the models at the server.

Data transfer The data is transferred from the client to the server across a normal TCP/IP connection, as seen above. There is also a parallel control connection that enables asynchrounous control of the recognition process.

Types of available models The available models, for the European Portuguese, are language models, pronunciation dictionaries (lexica), and acoustic models.

Acoustic models come in the form of a file containing the neural network weights (since *Audimus* uses MLPs for acoustic modelling), and they can be adapted to a user or set of users or be generic.

Lexica and langage models are, in *Audimus*, modelled by WFSTs. Language models can represent a contained task, where they can be used to obtain very good performance by restricting the possible sentences considerably, or larger tasks, with many different words, where, even compressed, they can grow to sizes of hundreds of megabytes.

4 Portability to an embedded system

In this section we focus, at a lower level, on the portability issues that were solved in order to obtain a working system.

We chose to port the PLP component as the feature extraction module. This was motivated by the fact that we intend to recognize clean speech, not speech significantly distorted by noise. As the acoustic model component, we selected the *ForwardMLP* component, which performs forwarding on a MLP model. We decided to port the acoustic model also, since we want to progress into a totally embedded system, where all the processing (including the decoding) is performed at the client.

We also ported *Audimus*' *MacroComponent* system, to make the specification of different pipelines of components more flexible. In doing so, we had to port the ZThreads Library [8], over which the *MacroComponent* system is built, to the device.

The devices we were targeting with our system have limited processing capabilities, in particular, they do not usually possess floating-point processing units. Emulating the inexistent floating-point units through software incurs a slowdown of about 10x, which renders it unacceptable for applications that are processing intensive, such as the computation of PLP features, needed for speech recognition. The solution is the use of fixed point computations in all arithmetic operations. The memory limitations of the processor are also relevant in the acoustic processing stage, because some acoustic models occupy non-negligible amounts of space.

4.1 PLP Component

The main issues found while porting the PLP component to the embedded device are described below. Some of the issues considered in this section apply to other components as well.

Computation of the FFT The FFT is a central component in the computation of PLP coefficients, 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.

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 [9], where a single bit selects one of two possible exponents. This ensures that the range is enough to cover the squared magnitude spectrum, while mantaining

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%.

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.

Further optimisation 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 ForwardMLP Component

In the ForwardMLP component, it is necessary to compute the output of a MultiLayer Perceptron. To that effect, we need 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. We also need to calculate the activation function, which is usually the sigmoid function, but that is easily done as a table lookup. This computation fits naturally within the framework of matrix multiplication, and the current implementation 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. It was, therefore, necessary to improve the baseline ($O(n^3)$) matrix multiplication algorithm directly with the following optimizations:

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. Instead, we partition the first and second matrices into blocks, and multiply them "blockwise" (i.e. the elements of the higher level multiplication operation will be matrices themselves).

The number of operations executed by the algorithm will be the same as before, but if we choose the size of the block so that it fits in the cache of the processor, the number of cache misses will be much lower, causing the algorithm to run much faster.

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) and speed (since our target processor can muliply four 16-bit values in one clock cycle). Also, as the input of our network consists not only of the current frame but also of the 3 that precede and follow it, 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.

5 Results

The PC used for our tests was a 2.4 Ghz Core Duo with 2 GB of RAM. The target device used in the tests was a 520 Mhz XScale, with 64 MB of RAM and 256 MB ROM running Windows Mobile 5.0. The network between the PC and device was a Wi-Fi 802.11b network.

We selected the radiology task to perform our tests, using an adapted language model existent in the laboratory, consisting of 13161 words. The test set was split into five tests, totalling 215 sentences and 2292 words. Each test contained one or more full radiology reports. The differences in recognition quality for each set were measured using two different acoustic models: the generic acoustic model, and the speaker dependent acoustic model. Also, we considered how different configurations - a) audio acqusition and processing in the PC, b) acquisition in the device and complete processing in the PC, c) distributed system with feature extraction in the device, d) distributed system that also includes MLP processing - would impact recognition speed, by measuring it against our test set. Finally, we also assessed the impact that using the PDA's internal, lower-quality microphone, would have in recognition quality. The results are summarized in the tables below.

There was no significant increase in delay incurred by the network transmission of the data from the device to the PC, as can be observed in columns 1 and 2 of table 1. Additionally, the increase in latency noticed when considering the configurations of the third and fourth column of the table was most likely due to overhead in the creation of the *MacroComponent* architecture in the PDA, which is more significant in the acoustic model case.

Table 2 shows that the use of speaker adapted models almost completely eliminated the errors in the radiology task. Furthermore, there was a small (0.51%) degradation in WER because of rewriting the PLP component in fixed point for the PDA. This is mainly due to the numerically unstable nature of delta

Test	Duration	Time(PC)	Time(PDA to PC)	Time(PLP in PDA)	Time(MLP in PDA)
1	349s	125s	126s	129s	135s
2	484s	161s	163s	167s	184s
3	546s	204s	205s	207s	220s
4	567s	207s	208s	211s	226s
5	540s	188s	190s	193s	210s

Table 1. Duration of each test (column 2) and time 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 (columns 5 and 6) respectively.

Test	Words	WER (generic model)	WER (adapted model)	WER (ad. model, in PDA)
1	317	7.89%	2.52%	3.47%
2	448	11.20%	2.23%	2.90%
3	520	8.85%	1.73%	2.31%
4	521	11.70%	2.50%	2.88%
5	486	9.05%	1.44%	2.06%
Average	458	9.86%	2.05%	2.66%
Total	2292			

Table 2. Word error rates using the generic model, the adapted model, and the adapted model with PLP processing done in the PDA, respectively.

calculation, used in the PLP component, which increases the relative error of the fixed point implementation by about one order of magnitude.

In the 2^{nd} column of table 3, we show the results of trying to directly reuse the speaker adapted model, used in the discussion above, with audio recorded using the PDA's microphone. This caused a large increase in word error rate, so we further adapted the above model using 100 additional sentences, recorded with the PDA's microphone. As a result, the WER dropped to 3.75% (3^{rd} column of table 3), which represents a degradation of less than 2% when compared to the use of a high quality microphone. There was no significant degradation caused by the execution of the PLP component in the PDA (column 4); in one of the testcases, the WER actually decreased. However, also porting the MLP component (column 5) increased the WER by 0.91%, probably due to the reduced precision of the 16 bit matrix representation, discussed in section 4.

6 Conclusions

In this paper we have presented a Distributed Speech Recognition system for the European Portuguese. Our intent to progress, in future work, towards a fully embedded system, where all the processing resides on the device, has motivated the decision of investigating two configurations for the distributed system, with and without the acoustic model component. In both configurations it was possible

Test	Base SA model	Adapted. model	Adapted.model+PLP	Adapted.model+PLP+MLP
1	14.83%	1.89%	2.84%	4.10%
2	19.20%	4.24%	4.24%	4.91%
3	17.12%	3.46%	3.46%	4.03%
4	25.34%	4.61%	4.22%	5.76%
5	13.40%	3.91%	3.91%	4.53%
Total	18.41%	3.75%	3.80%	4.71%

Table 3. Word error rates using the PDA's internal microphone. The 2^{nd} column refers to the acoustic model trained with the high quality microphone, while the 3 last columns refer to the acoustic model trained for the PDA's microphone . In the 4^{th} and 5^{th} columns, the PLP component / PLP + acoustic model, respectively, are executed in the PDA.

to achieve word error rates below 5% , with a speaker and microphone adapted acoustic model, in a 13161 word radiology task.

7 Acknowledgements

This work was funded by PRIME National Project TECNOVOZ number 03/165.

References

1. Meinedo, H., Caseiro, D. A., Neto, J. P., Trancoso, I.: AUDIMUS.MEDIA: a Broadcast News speech recognition system for the European Portuguese language. In: PROPOR'2003, Faro, Portugal (2003).
2. Meinedo, H., Neto, J. P.: Automatic speech annotation and transcription in a broadcast news task, In ISCA Workshop on Multilingual Spoken Document Retrieval, Macau, China (2003).
3. Ramabadran, T., Sorin, A., McLaughlin, M., Chazan, D., Pearce, D., Hoory, R.: The ETSI Distributed Speech Recognition (DSR) standard server-side speech reconstruction, In: IBM Research Report (2003).
4. Suk, S. Y., Jung, H. Y., Makino, S., Chung, H. Y.: Distributed Speech Recognition System for PDA in Wireless Network Environment. In: Proceedings of SPECOM'2004, St. Petersburg, Russia (2004).
5. Paul, Douglas B., An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with Stochastic Language Model. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP-92), San Francisco, USA (1992).
6. Caseiro, D., Trancoso, I.: A Specialized On-the-Fly Algorithm for Lexicon and Language Model Composition. IEEE Transactions on Audio, Speech and Language Processing, vol. 14, pp. 1281-1291 (2006)
7. PortAudio - portable cross-platform API `http://www.portaudio.com`
8. ZThreads - A platform-independent, multi-threading and synchronization library for C++ `http://zthread.sourceforge.net/`
9. Ewe, C. T., Cheung, P. Y. K., Constantinides, G. A.: An Efficient Alternative to Floating Point Computation. In: Lecture Notes In Computer Science, LNCS, vol. 3203, pp. 200–208. Springer, Heidelberg (2004).