Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

An Efficient Distributed Speech Recognition Front-End Implementation using a Motorola Star Core 140-Based Platform

Corneliu Burileanu[1], Vladimir Popescu

Abstract – An efficient implementation of a Distributed Speech Recognition (DSR) on a Digital Signal Processor (DSP) platform is shown. Although complying with the European Telecommunications Institute (ETSI) DSR standard, feature extraction and data compression enhancements are added, in order to address three issues: computational speed of the signal analysis process, reliability of the representative signal features, and low channel bandwidth utilization, when transmitting the speech features. A series of tests and simulations are provided, in order to show the functionality of the proposed system.

Keywords: Distributed Speech Recognition, Feature Extraction, Vector Quantization

I. INTRODUCTION

Portable devices (such as PDAs- Personal Digital Assistant and mobile devices) have become very popular nowadays. Since the size of these devices is becoming smaller and smaller, their manipulation by traditional means (i.e., keyboard, touch-screen) can be cumbersome. Therefore, the idea of using speech recognition to control portable devices is natural. However, these devices are fitted with low computing power processing units, and with low amounts of memory also. This is why speech recognition, which is a computationally very demanding task cannot be implemented on such devices in a straightforward manner. Previous trials of speech recognition system implementations on DSPs (which fit mobile devices) have been pursued [2], but the results were rather poor: connected word recognition of a small vocabulary (100 words). But in order to interact with portable devices in a more natural way, this is not enough: continuous speech recognition of a large vocabulary (1000 words) is much more appealing. In order to accomplish this task, the idea of Distributed Speech Recognition has emerged a few years ago; it eventually attained a standardized description, in 2003 [1]. However, the standard specifies only parts of the entire DSR system: the feature extraction block (called front-end), which includes a speech parametrization module, a voice activity detection (VAD) module, and a pitch extraction module. The standard also specifies the techniques for voice reconstruction, at the server side (called back-end). Although several implementations of the ETSI DSR standard are known, some issues are still intensively studied: the robustness of the DSR system to the channel perturbations and variabilities, more specifically, to additive distortions (channel noise) and convolutive distortions (channel frequency response). These issues can be basically addressed in two ways: robustness of the back-end speech recognition engine itself, and specific signal processing techniques: echo cancellation, spectral equalization.

Our paper proposes an alternative approach: improving the DSR system performance by careful speech analysis (parametrization) and vector quantization (VQ) of the feature vectors obtained. The ETSI DSR standard also specifies a quantization procedure, but in an unsupervised manner. Instead, we propose a vector quantizer designed using a modified version of the Linde-Buzo-Gray (LBG) algorithm. The modification from the original LBG algorithm is so as to reduce the computational power required, at no accuracy cost.

The paper is structured as follows: Section II gives an overview of the DSR system architecture, emphasizing the feature extraction part; Section III brings insights on the front-end implementation, on a Motorola Star Core 140 core; Section IV shows some results from simulations of the proposed system, specifying the practical framework used and testing policy; Section V concludes the paper and asserts future developments.

II. SYSTEM ARCHITECTURE

A. *Overview*

The proposed DSR system consists, as stated before, of two parts: the actual system, placed on the server, which encloses a back-end for the recognition task

[1] Facultatea de Electronică , Telecomunicaţii şi Tehnologia Informaţiei, Universitatea "Politehnica" Bucureşti, Bd. I. Maniu Nr.1-3, Bucureşti, e-mail cburileanu@messnet.pub.ro

and a front-end also, for the signal processing needed to train the system, and a front-end placed remotely, on the client device, needed for system testing. From these, the former is implemented on a standard Personal Computer (PC), and the latter, on a DSP-based platform (fitted with a Motorola SC140 Core). A block diagram of the DSR system is given in Figure 1.

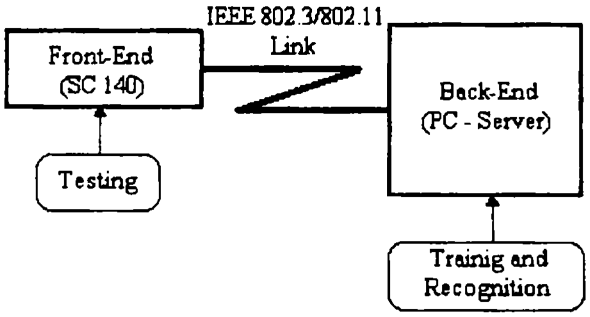

Fig. 1 DSR system architecture

As seen from Fig.1, the back-end and the front-end are connected through an usual IEEE 802.3 (Ethernet) link, or through an IEEE 802.11 (Wireless) Link. We will use the former connection in our experiments.

B. *The Front-End*

The client side of the DSR system is concerned mainly with speech signal acquisition, endpoint detection and feature extraction. These analysis features are accomplished following the ETSI DSR standard [1], but with slight modifications:

- the VAD module uses also the zero cross rate (ZCR) for the speech/silence decision, into an adaptive algorithm, as shown in [3];
- the preemphasis(PE) of the speech signal is performed before framing, as it is the case in the nowadays speech recognition engines [4];
- the feature vectors are considered by taking, besides the Mel Frequency Cepstrum Coefficients (MFCC), the differenced MFCC coefficients also (ΔMFCC);
- the differenced energy (ΔEN) is taken along with the energy (EN), in composing the feature vector of each speech frame.

Using the EN parameter, along with the ZCR, the VAD process simplifies considerably; the algorithm reduces itself to successive comparisons of the EN and ZCR with some fixed, specified thresholds: for the EN, these thresholds are computed from the maximum and mean values of the energy, for a given utterance; for the ZCR, the thresholds are computed from the sum of the ZCRs, and a form of dispersion of this sum [3]. This algorithm alleviates the need for the complex spectral estimation blocks, present in the ETSI standard [1].

The inversion of the PE and the framing processing features is performed for compatibility reasons: most of the continuous speech recognition engines extract the feature vectors in this way [4], [5]. Thus, one creates the premises for the ease of the back-end recognition task, at no computational cost for the front-end.

Considering the first order temporal variations of the mel-cepstral coefficients is particularly suited for the short-term stationary speech signal, since this way one can take into account the coarticulatory effects, between phonemes or even words [4]. Temporal changes in the spectra play an important role in human perception; this is particularly true for speaker independent recognition, where formant slopes are more relevant than the absolute formant locations [4]. We also chose the ΔMFCCs for compatibility reasons: the most speech recognition engines perform this feature extraction for their server-side front-end [4],[5]. The ΔMFCCs are computed with:

$$\Delta MFCC(t) = MFCC(t+\theta) - MFCC(t-\theta) \quad (1)$$

We used also the differenced energy (priorly normalized by subtracting the mean value in a frame, from each energy value in the frame; one thus alleviates the need for the log-energy computation, reducing thus more the cost!), which provides information about relative changes in amplitude or loudness. It is given below:

$$\Delta EN(t) = EN(t+\theta) - EN(t-\theta) \quad (2)$$

In order to further increase the efficiency of the system, a vector quantization (VQ) is performed, using a modified version of the Linde-Buzo-Gray (LBG) algorithm.

We first used the classical LBG algorithm [2], taking as the first centroid, the arithmetical mean of the feature vectors of all the frames of a given utterance, then, choosing a small division parameter, we built the codebook for the feature vectors, representing the frames for a given input utterance.

Then, in order to further reduce the computational power, we adapted a modified version of the LBG algorithm, in which, by interverting the optimization loop with the division loop, one practically optimizes the codebook only when it reaches its final dimension [2]. A flowchart of the modified LBG algorithm is shown in Figure 2 [2].

The feature extraction also includes a decision block, used in order to map the feature vector to the corresponding codebook, so as to minimize a certain distance between the codeword and the feature vector.

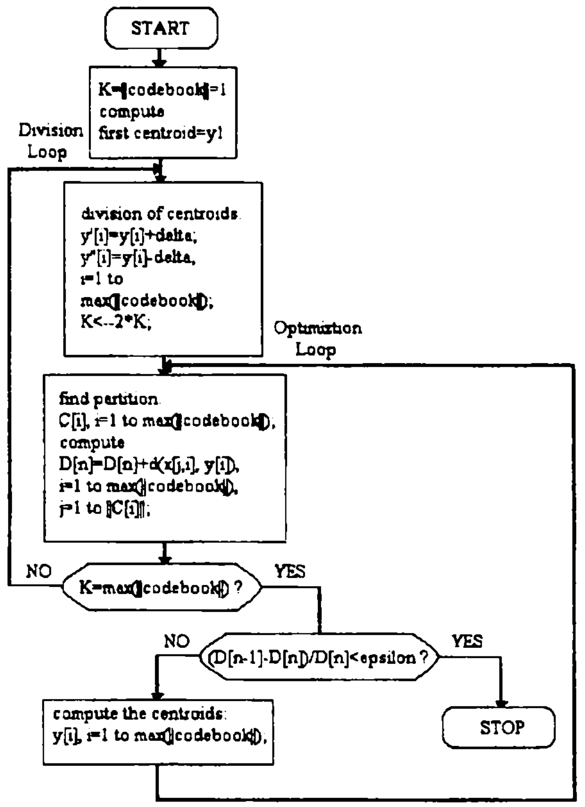

Fig.2 The modified version of the LBG al orithm

Due to the computational simplicity, we have chosen the Euclidean distance, also used in the state-of-the-art speech recognition applications:

$$d(x, y[j]) = \left[\sum_{k=1}^{K} |x_k - y_k[j]|^2\right]^{\frac{1}{2}} \qquad (3)$$

For each frame of speech, not one but several codebooks are used to replace the input vector. Since each input vector is a vector of symbols, the recognition algorithms at the back-end are producing multiple symbols for each frame. By assuming that multiple output observations are independent, the output probability of emitting multiple symbols can be computed as the product of the probabilities of producing each symbol. Thus, the coefficients can be divided into distinct sets, each set being quantized into a separate codebook. For our system, following previous approaches [4], three codebooks were created, each with 256 centroids. These codebooks were generated from: 1) the MFCC coefficients, 2) the differenced MFCC coefficients, ΔMFCC, and 3) an *equally* weighted combination of energy (EN) and differenced energy (ΔEN).

The multiple codebook approach has a distinct advantage over the single codebook approach specified in the ETSI DSR standard – namely, reduced quantization error. If too many features are used in VQ, the distortion will be very large, which means the observed vectors will match their corresponding prototype vectors poorly. Multiple codebooks reduce the distortion by partitioning the feature space into several smaller subspaces.

Another advantage of multiple codebooks is the large increase in the dynamic range and precision of the resulting parameters. With three codebooks, there are 256^3 possible parameter combinations using just 256×3 parameters. With such an increase in precision comes the ability to make finer distinctions.

However, the independence assumption with multiple codebooks is inaccurate. Also, more memory and time are needed with multiple codebooks. But the modified version of the LBG algorithm can compensate partly this increment in computation time and memory requirements.

As a conclusion to what was stated in this subsection, the block diagram of the proposed front-end is shown in Figure 3.

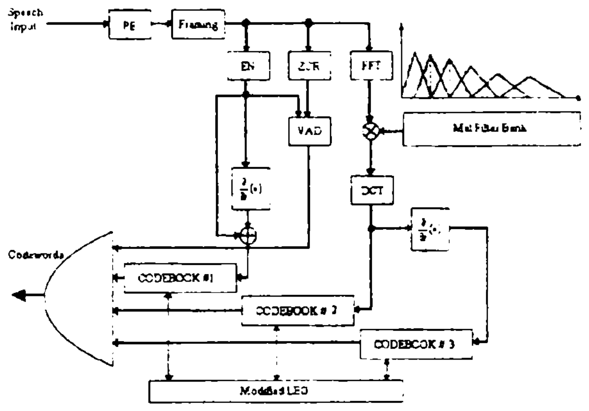

Fig. 3 The front-end block diagram

For further enhancement of the resulting front-end, a Hamming window can be added, following the framing block.

Besides these specifications, with the exception of the Pitch extraction module (which is not implemented at present), the proposed front-end follows the ETSI DSR standard [1].

C. *The Back-End*

In order to ensure compatibility with actual speech recognition engines and even software products, we used the Carnegie Mellon University's SPHINX-II speech recognition engine [5] for our DSR system back-end.

The SPHINX-II system has a phonemic-HMM three-stage Viterbi decoder and is designed for building applications of small, medium and large vocabulary speaker-independent continuous speech recognition, being one of the state-of-the art products in its field [7], [8].

In order to integrate the SPHINX system, which is a standalone recognition engine, with our DSR system, we had to adapt the input/output Application Programming Interface (API), so as to process the (fixed-point) code vectors, instead of its own (floating-point) code vectors. The API of the SPHINX-II system is described in detail in [8]. Thus,

the modified version of SPHINX provides only training (this is why we kept its own feature extraction block) and actual recognition, which is based on three knowledge sources: 1) the vocabulary (for the English language at the moment), 2) the (three-state triphone) HMM acoustic models, and 3) the grammar. For our tests we chose a finite state word-pair grammar (FSG) (which is simple to implement, and offers higher perplexity than an n-gram language model).
In fact, the architecture of SPHINX-II is more complex, but this is not our concern yet. A simplified architectural overview of the SPHINX-II system is given in Figure 4, adapted from [7].

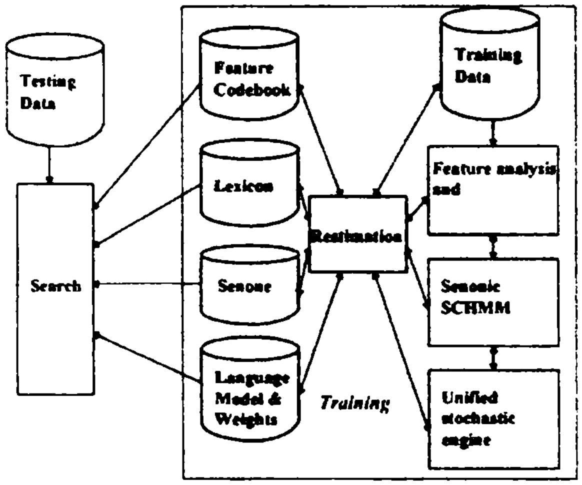

Fig. 4 SPHINX-II system block diagram

In our system, the "Testing" block is replaced by a "Communication" block, which encloses the reception of the (coded) features from the client device.

III. MOTOROLA SC 140 IMPLEMENTATION

The architecture of the Motorola Star Core 140 DSP core is presented in detail in [9]; a brief overview of the processor is given also in [10].
For our implementation, we used CodeWarrior 2.52 as the Integrated Development Environment (IDE) for the front-end implementation. Furthermore, we simulated its behavior using the Motorola SC100 Sim Star Core Simulator. Thus we could perform some benchmarks concerning the execution speed and the memory requirements. This way, we depicted the following issues:

- The fixed-point implementation involves possible overflow when computing parameters like EN or even MFCC, therefore scaling methods should be used.

- The computation of the FFT involves extensive memory usage, therefore attention should be paid when configuring the memory layout.

- The parallel architecture (4 Arithmetic-Logic Units – ALUs and 2 Address Generation Units – AGUs) permits the parallelization of the codebooks generation process, and also of the energy and zero cross rate computation.

In the fixed-point implementation, many scaling techniques can be used:
- the normalization through division, with respect to the maximum value of a certain feature, considered over a window of analysis (Hamming-window weighted or not);
- the normalization through division, with respect to the summation of all the values of a certain feature, considered over a window of analysis (Hamming-window weighted or not);
- the normalization through subtraction, with respect to the mean value of a certain feature, considered over a window of analysis (Hamming-window weighted or not).

From all these, we preferred the normalization through subtraction of the mean value of the features within a frame.
Our choice is motivated by the fact that in this way one performs mostly additions (for the computation of the mean value) and subtractions (for the actual normalization), which are more robust to possible overflow or underflow, on a fixed-point platform. The only division is performed when computing the mean value, but this can be eased by choosing a number of features within a frame which is a power of two (e.g. 32 features per frame); the division becomes in this case a right shift by a number of positions equal to the respective power of two.
As for the memory required to compute the FFT, one can depict a memory layout in which to place the stack segment very high in the 16-bit address space, and furthermore, the code segment is placed right below the stack segment, reserving as much space for the code as it is needed. And then, the rest of the memory remains basically for the data segment.
The space required by the code segment can be determined by simulating the code execution on an Intel x86-based PC platform (e.g. using Microsoft Visual C++), and looking (with a debugger) at the memory consumed by the actual code.
A possible memory layout is shown in Figure 5.

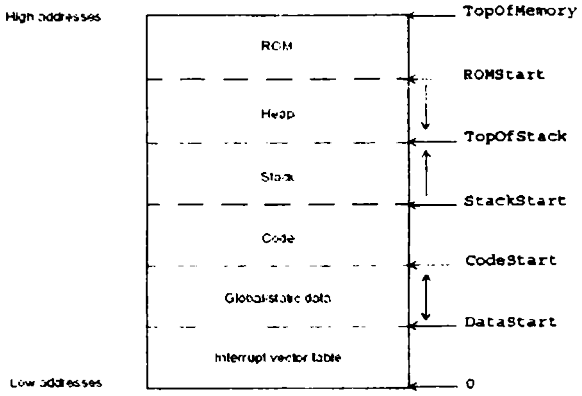

Fig. 5 SC 140 Core memory layout

As for the parallelization of the tasks, a naïve approach is to use standard optimization techniques [9] for all the computational routines. In this way, one accomplishes still a sequential codebook computation flow, which is not efficient. This is why we propose a parallelization at the main routine level, which calls the codebook generation routine three times, in parallel, using as inputs the already-computed feature vectors. This way, although achieving a lower performance for a single codebook computation, the overall performance is better, since one has all the three codebooks generated at the same time.

IV. EXPERIMENTS

At the moment when our experiments were performed, we simulated our front-end using Microsoft Visual C++ and building a project enclosing the routines described below. Using this environment, we could also simulate (syntactically) the SC 140 Core (including the prototype function definitions for the intrinsic routines associated with SC 140 [9]. This way, starting from the wave (.wav) files, we generated the feature vectors and the codebooks (with the code words) in corresponding files (.dat). These .dat files were used as input to a Simulink model, with which we simulated channel distortions (additive and convolutive). The outputs of the Simulink model (which are also some .dat files), were used as input to the (already trained) SPHINX-II System. As for the recognition back-end, we used an adapted application provided by CMU, in order to launch the Viterbi decoder using the .dat files (provided by the Simulink model) as codeword inputs. In the Simulink channel simulation, a Gaussian noise can be added, along with an analog Butterworth 12-order bandpass filter, in order to simulate the convolutional perturbations. The model is shown in Figure 6.

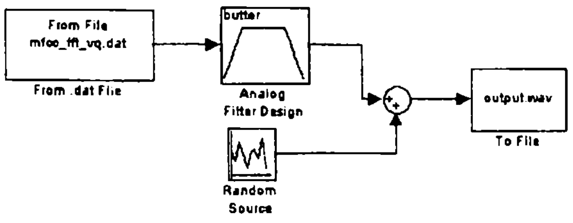

Fig. 6 The channel simulation

The performance evaluations show the following:

- The modified LBG algorithm brings an about 32% decrement in computing time, along with only 6% increment in error rate, as compared to the original LBG algorithm.
- The memory requirements for the feature vectors are about 77 KB each (for a 6-second speech input). After VQ, the memory requirements go to about 20 KB, for all the three codebooks.
- The processing time (however, on a 1.47-GHZ AMD Athlon XP Processor) is real-time, less than 10 seconds.
- The code size for the entire front-end application is about 240 KB.
- The recognition performance, for a FSG grammar, provided by the SPHINX-II developers at CMU, which provided also the about 1000-word vocabulary and phonetic transcriptions, along with the acoustic HMM training, is close to the one obtained using the baseline SPHINX-II system: around 68%, for our system, as compared to 76%, for the self-contained SPHINX-II system.

It is worth mentioning that these relative low recognition rates are due to the poor-quality desktop microphone used, as compared to the microphones used for SPHINX-II training, which was performed at CMU.

V. CONCLUSIONS AND FURTHER DEVELOPMENTS

A novel front-end designed was proposed, starting from the ETSI DSR standard. Our system emphasized the idea of circumventing additional signal processing associated with the DSR task: echo cancellation, spectrum equalization etc, by a careful signal analysis and feature extraction. Thus, besides the energy and ceptsral coefficients, we used their first order variations also, providing more robustness to environmental conditions variation.

In order to further improve system performance, we used a vector quantization approach, but in a supervised manner. considering multiple codebook building. The use of more than one codebook per feature vector, at the cost of minor increment of memory requirements, offers better error protection, at a real-time computation time.

The Motorola Star Core 140 architecture chosen allows us to parallelize the process of codebook generation, along with the one of the feature vectors generation, which offers us real-time performance.

As further developments, we intend to use the front-end implementation on the MSC8101 ADS Development Board [10], which encloses an audio codec, along with an IEEE 802.3 Ethernet connection, besides the actual SC 140 DSP core. Thus, we plan to connect the MSC8101 ADS to a PC, using an Ethernet connection, and to repeat the previously performed tests within this new framework.

REFERENCES

[1] ETSI ES 202 211 v1.1.1 (2003-08), "*ETSI Distributed Speech Recognition Standard*", www.etsi.org, 2003

[2] L. Bojan, *Contribuţii la analiza semnalului vocal*, PhD Thesis, University "Politehnica" of Bucharest, 1997

[3] M. Ionita, *Strategii de recunoaştere a semnalului vocal*, PhD Thesis, University "Politehnica" of Bucharest, 2003

[4] K.-F. Lee, H.-W. Hon, R. Reddy, "An Overview of the SPHINX Speech Recognition System", *IEEE Trans. On Acoustics, Speech and Signal Processing*, Vol. 38, No. 1, Jan. 1990

[5] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, R. Rosenfeld, *The SPHINX-II Recognition System: An Overview*, CMU-CS-92-112, Carnegie Mellon University, 1992;

[6] M.-K. Ravishankar, *Efficient Algorithms for Speech Recognition*, PhD Thesis, Carnegie Mellon University, 1996

[7] R. Singh, *Subphonetic modeling for Continuous Speech Recognition*, PhD Thesis, Carnegie Mellon University, 1997

[8]M.-K. Ravishankar, K. A. Lenzo, *Sphinx-II User Guide*, Carnegie Mellon University, 2004

[9] Motorola, Inc. *SC 140 DSP Core Reference Manual*, Rev. 3, 2001

[10] D. Burileanu, A. Feciorु, D. Ion, M. Stoica, C. Ilaş, "An Optimized TTS System Implementation Using a Motorola Star Core SC140-Based Processor", *Proc. of ICASSP2004*, Montreal, Canada, 2004