DEVELOPMENT AND IMPROVEMENT OF A REAL-TIME ASR SYSTEM FOR ISOLATED DIGITS IN SPANISH OVER THE TELEPHONE LINE

Ricardo de Córdoba, Xavier Menéndez-Pidal, Javier Macías-Guarasa, Ascensión Gallardo, José M. Pardo
Grupo de Tecnología del Habla, Dpto. Ingeniería Electrónica, ETSIT Madrid, UPM
Ciudad Universitaria s/n. Madrid 28040. Spain
e-mail: cordoba@die.upm.es

ABSTRACT

We present the development and characteristics of a basic ASR system for isolated digits in Spanish, used over the telephone line. Initially we will introduce our first idea, a basic discrete system, and then we will see the improvements we made to increase the recognition rate at a low CPU cost (always considering its practical implementation as a real time system). The most remarkable advances were obtained with:
1) Semicontinuous modelling. It is a more precise modelling, although more time consuming.
2) End-pointing with a Neural network.
3) One pass decoding with noise models.
The intention of both 2 and 3 is to alleviate the effects of a wrong end-pointing.
4) Parametrization using perceptual filters in frequency and filtering in the time domain (RASTA-PLP). We wanted to decrease the effect of telephonic noise in our system.

1. EXPERIMENTAL SET-UP

Our database consists of 1320 repetitions of the ten digits and a break command in Spanish (that makes an average of 120 reps./word). It was captured over the telephone line with different volunteers, covering male and female, and a wide range of ages. We had much more male speakers than female (about 50% more), so our recognition rate for female was much worse. In some pilot experiments we got error rates of 2% for male, but 5.5% for female. We will try to balance our database in the near future.

The overall real-time system has been used as information center for students who wanted to know their college califications by phone. We do not have recognition rate figures for this application, as we did not have enough time to verify all the calls.

To make our tests we split the database into six balanced parts (same number of repetitions per digit in each one), train with five parts and test with the remaining one. The main purpose of this approach is to increase the significance of our tests. We make this with the six parts and add the scores obtained in all of them.

We considered two types of end-pointing for our system:

1) *Automatic end-pointing*, based on energy levels. The begin and end frame of a word is obtained following given thresholds of energy.
2) *Manual end-pointing*: manually verified data.
Obviously, the first one gives the performance of our system in real conditions. The second one is just interesting for comparison purposes.

2. BASE SYSTEM

To begin with, we used discrete HMM word models. We made experiments in three directions to tune our system: find the best set of parameters and the way to group them in different codebooks, the number of centroids in each codebook and the right number of states per model.

2.1. Set of parameters

We use 10 MFCC Cepstrum and the total energy. We include the first and second derivate of these parameters, as they are quite important in speaker independent systems, giving a total of 33 parameters. We group them into three different codebooks, the first one with MFCC and energy, the second with delta-MFCC an delta-energy (first derivative) and the third with the acceleration parameters (second derivatives). Previous experiments in our labs showed high correlation between MFCC and energy (same with their deltas), obtaining good results with this association.

The recognition rate increased a lot with the second and third codebook of parameters. In our first experiments (not to be taken as final figures) we got about 85% - 93% - 96% using one, two or three codebooks respectively.

2.2. Quantification

We tested different number of centroids for each codebook as a trade-off between recognition rate and cost (memory and processing time). We can point out that we did not get our best figures with 256 centroids for each codebook. It was better using 128 for the first one, probably because the dynamic range of these parameters is smaller, and 256 for the second and the third one.

2.3. Number of states

First, we tried using a fixed number of states per model, but we got better results using a variable number of states per model. We made the number of states proportional to the average duration in frames of the model, which we obtained from the manually labelled files. We got our best results with an average of 17 states/model:

96.1% (manual end-pointing), 95.1% (automatic)

We can point out that the recognition rate for the training set (recognition with files used in training) was 99.6%, which showed that, probably, with a larger database the results would be better. That will be in our next release.

3. IMPROVEMENTS

3.1. Semicontinuous Modelling [1][2]

We model the VQ codebook as a family of finite mixture probability density functions (the distributions overlap, so there is no partitioning). Each codeword of the codebook is represented by one Gaussian pdf with a mean vector and a variance. As initial estimate, we take the discrete centroids as mean vectors of the gaussians. Then we calculate, with the training speech material, the corresponding variances to obtain first estimates of the basic modelling mixtures.

In the HMM model, the pdf produced by a vector x in- state i can be written as:

$$b_i(x) = \sum_{v_j \in \eta(x)}^{M} f(x/v_j)\, b_i(j)$$

where $b_i(j)$ is the HMM probability, $f(x/v_j)$ is the value of the Gaussian pdf for vector x given codeword v_j, and 0(x) is the set of VQ codewords for the most significant values of $f(x/v_j)$ (M). We used M=4, which improves the performance and reduces computation time.

We got a significant improvement in recognition rate:
98.2% (manual), 97.1% (automatic)
with the same number of states and centroids as before.

The drawback is the increase in processing time in our real-time system. To reduce this time we have tested some strategies to override some computing:

* Originally, we had to compute the Gaussian pdf $f(x/v_j)$ for all codewords, and select the M best codewords. For each codeword, we had to make computing for the eleven components of our vector of parameters, which took quite a lot of time. Now, we follow the next steps:
1) For each codeword, we compute a simplified Gaussian considering only P parameters.
2) We select the best T/F codewords, where T is the total of codewords and F is a prunning factor (ranging from 2 to 7 in our experiments).
3) We complete the Gaussian computing only for the codewords selected.
4) We select the M best in the last set.

The results were really good. The final M selected codewords usually were the same as the original, and, as we did the complete computing for them, the weights were the same.

Increasing the value of F and decreasing P, the time reduction is more significant. We achieved similar recognition rates, tested over a wide set of experiments, using F up to 6 or 7 and P=4. In these conditions, we got up to 50% time reduction with almost no loss in accuracy.

* We did some pruning in our Viterbi routine too, making the computing in each frame only for the best state and a set of adjacent states (as a function of the total number of states). We got significant reduction time with almost no loss in accuracy.

3.2. End-pointing with a Neural network [3]

As we can see in the results, we had another problem remaining: performance was much worse with our automatic end-pointing than with the manual one. We have tried two ways to solve this problem, using a Neural network and/or using one-pass decoding with noise models (see 3.3).

We use a TDNN with one output, giving signal or no signal. It is a three-layer network, with one neuron in the second and third layers, and N (variable) in the first one.

- Topology: The TDNN neural arquitecture has been chosen to provide time invariance and output stabilibty in discriminating silent as well as speech segments. The neural network is trained using stocastic gradient descent method to speed up the training procedure. A unique output node is used to discriminate speech and non-speech labels, setting the desired output to 0 when a silent segment is presented to the network and to 1 when a speech segment is trained. A global input width of 100 ms experimentally provides good accuracy. The width of the contextual windows in the TDNN, is 30 ms in the first layer and 80 ms in the second one. This last configuration makes the recognition task experimentally robust averaging the output layer three shifted examples of the second layer and using 8/12/16 hidden cells in the first TDNN layer. Once the TDNN has been trained, a frame-by-frame accoustic string is produced for each input pattern. For ouput values less than 0.5, the central input frame is classified as a silent segment, and for output values greater than 0.5 the central input frame has been assigned as speech segment.

- Training examples: We train the NN picking from each word of the training set 14 examples of 8 frames each, 7 representing noise and 7 signal. We tested other configurations with different number of examples and frames in each example, obtaining similar results. We take the same number of examples for noise and signal to keep our database balanced. We always take examples from the boundaries of both signal and noise, because what we really want is to discriminate the boundaries of noise and signal. In many cases there is some overlapping between the samples, because the word is too short, and does not have 7 * 8 signal frames.

- Training: with our network parameters, convergence was quick, but never got a minimum. After a few iterations, the average error decreased very little, so we stopped after 40 iterations. We made some experiments using the weights obtained with more training, but got no improvement.

- Adjustments to the output given by the NN: we filtered the output, to have steady series of 1's, and did some postprocessing when we had more than one series, first to decide which one was best and then if two series should be joined or just discard one of them. All decisions were based on the output values of the net (floating point value), looking for overall majority, and tested over the whole database.

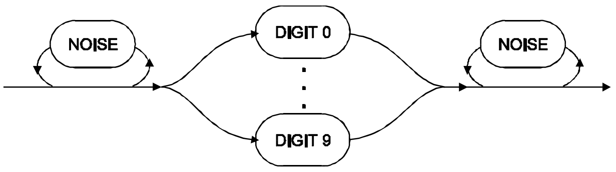
- Experiments: we made experiments changing the number of neurons in the first layer and the number of examples used from each word. We present the results, considering the end-pointing accuracy (two values, one for begin and the other for end) and the recognition rate, in the following table. We compare with manual and automatic end-pointing.

System	**Exampl-neurons**	**Average error**	**Gross erros**	**Discrete results**	**SC results**
Manual				96.1	98.2
Autom.		4.47-4.41	113-80	95.1	97.1
Neural	7 - 8	3.39-4.59	26-62	95.8	98.2
Neural	7 - 12	3.47-4.93	26-71	95.9	98.2
Neural	7 - 16	3.31-4.76	24-63	95.9	98.0

Conclusions:
- Much better results than those obtained with automatic end-pointing.
- The performance is not dependent on the number of neurons and the number of examples used in training (results not shown for this).
- Almost same results as manual, but now automatically.

3.3. One pass decoding with noise models [4][5]

A completely different approach to override the wrong automatic end-pointing is to use noise models. We train two different noise models with three states each using noise frames (out of boundaries of manual end-pointing) and, in recognition, given the automatic end-pointing, we consider 15 additional frames at each side of the word. We use the One-pass algorithm with the following network:



It solves two problems:
1) If there are losses with the automatic end-pointing (beginning end-point too late or ending end-point too soon), as we consider 15 additional frames, the loss would be ignored or at least smaller.
2) If the end-pointing is good, or includes noise, there is no problem because the additional frames are absorbed by the noise models, as experience shows.
The only problem with this approach is the increase in CPU time, as we consider more models and more frames.

We got a slight improvement allowing some freedom in the transitions for the noise models (let them begin or end in the middle state).
Results: considering manual end-pointing there is no improvement, as the digit models are the same, but now there is no significant difference with automatic end-pointing. Only a couple of crashes in the automatic system cannot be recovered by the one-pass. We present here the results for the semicontinuous system:

98.3% (manual), 98.2% (autom.), 98.4% (neural net).

showing no significant differences between these end-pointings. Observe that this approach can be combined with end-pointing with a neural network for more reliability, but it is not necessary if we have CPU time restrictions. This is why we did not use it for the following experiments.

3.4. Parametrization using perceptual filters in frequency and filtering in the time domain

In our parametrization, to compute the energy in each band of frequency, we used triangular filters with overlapping, which gave good results in previous systems of our laboratory [6]. We considered new ideas for noisy environments. First, we used perceptual filters, similar to those proposed in [7], but still using MFCC. The idea was to keep the philosophy of our parametrization, but taking advantage of the auditory model used in PLP instead of our triangular filters. The steps we follow are: Hamming window, preemphasis, FFT, compute energy in each band (critical-band spectrum) and compute MFCC. Now, we use the filters and weights from the auditory model to compute the energy used in MFCC computation. See results in Table 1 for "Perceptual MFCC", (1) means discrete system and (2) semicontinuous one-pass. We can point out that the error rate decrease is close to 20% in the discrete system (automatic).

Then we consider filtering in time the values of the energy in each frequency channel, before computing the MFCC, using a band-pass filter with spectral zero at the zero frequency, as proposed in [8] (Rasta-PLP). The purpose is to reduce the influence of the communication channel to the spectral values, as any constant or slowly-varying component in each frequency channel is suppressed, so the spectral estimate is less sensitive to slow variations. This way we minimize the effect of the telephonic channel, obtaining similar spectral estimates for different channels. We followed the next steps:

1) We compute the energy in each filter the same way as before, and keep the logarithm. This way, the supressed constant spectral components come from convolutive factors due to the channel.

2) Apply the following band-pass filtering to each frequency channel:

$$H(z) = 0.1 * \frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{z^{-4}\,(1 - Kz^{-1})}$$

The purpose of filtering low frequencies is to alleviate the effect of the convolutional noise, supressing linear distortions which appear as an aditive constant in the log spectrum, and filtering high frequencies smooths the fast spectral changes due to analysis procedures. We made experiments for different values of K, and got our best results with K=0.94.

3) Compute the MFCC using these filtered critical-band values.

The experimental results are shown in Table 1 ("Rasta MFCC") (as before, (1) is discrete and (2) semicontinuous one-pass). They show a significant improvement in our discrete system, giving and additional error rate reduction of 22% for the discrete system (automatic).

We tried setting a threshold for the filtered values (no negative values), but got no improvement.

We mixed 50% of the critical-band spectrum (PLP) and 50% of the rasta-filtered spectrum, but results were a little worse.

System \ End-pointing	**Manual**	**Auto-matic**	**Neural Network**
Discrete (1)	96.1	95.1	96.1
Semicontinuous (SC)	98.2	97.1	98.2
SC - One-pass (2)	98.3	98.2	98.4
Perceptual MFCC (1)	97.4	96.0	
Perceptual MFCC (2)	98.5	98.5	
Rasta MFCC (1)	97.7	96.9	
Rasta MFCC (2)	98.	98.6	

Table 1. Summary of results.

4. CONCLUSIONS

The results show that we are close to the top in the SC systems, as many mistakes can be considered impossible to avoid. We will increase the size of the database in order to obtain new figures showing more differences in those systems. Both 3.2 and 3.3 are good approaches to override a bad energy-based end-pointing.

The Rasta perceptual MFCC gives a good improvement for the discrete systems, showing that they are a good choice in our noisy environment and show great adaptation to telephone lines.

The error rates reduced a lot, up to 37% for the discrete system, and up to 71% if we compare our last semicontinuous system with the first discrete.

5. REFERENCES

[1] J. Ferreiros, R. de Córdoba, M.H. Savoji, J.M. Pardo, *"Continuous speech HMM training system: Applications to speech recognition and phonetic label alignment"*. Granada, June 1993, NATO ASI "New advances and trends in speech recognition and coding"
[2] X.D. Huang, Y. Ariki, M.A. Jack, *"HMM for speech recognition"*, Edinburgh Univ. Press.
[3] X. Menéndez-Pidal, J. Ferreiros, R. de Córdoba, J.M. Pardo, *"Recent work in hybrid neural network and HMM systems in CSR tasks"*, Vol. 3, 1515-1518, ICSLP-94.
[4] H. Ney, *"The use of a One-stage dynamic programming algorithm for connected word recognition"*, IEEE Transactions on ASSP, April 1984.
[5] J. Macías-Guarasa, M.A. Leandro, J. Colás, A. Villegas, S. Aguilera y J.M. Pardo, "*On the Development of a Dictation Machine for Spanish: DIVO"*. ICSLP-94.
[6] R. de Córdoba, J.M. Pardo, J. Colás, "Improving and optimizing speaker independent, 1000 words speech recognition in Spanish", ICSLP-92, pp. 161-164.
[7] H. Hermansky, *"Perceptual linear predictive (PLP) analysis of speech"*, JASA, pp. 1738-1752, 1990.
[8] H. Hermansky, N. Morgan, A. Bayya, P. Kohn, *"Rasta-PLP speech analysis technique"*, IEEE ICASSP'92, pp. 121-124.