Speaker-independent embedded speech recognition using Hidden Markov Models

Mariano Marufo da Silva, Diego A. Evin, and Sebastián Verrastro

Abstract— **This paper describes the implementation of a real-time, speaker-independent isolated speech recognition system using Hidden Markov Models on a 32-bit ARM Cortex-M4F microcontroller. We introduce the theory, the requirements and details of the embedded implementation. The evaluation was made using a multi-speaker isolated-digit corpus of Argentinian Spanish, and its performance in terms of accuracy, speed and required memory was compared against a baseline Dynamic Time Warping recognizer, implemented previously on the same architecture. Test results show that the proposed HMM system outperforms the baseline system, and exhibits a recognition accuracy of 96.21% under a clean acoustic environment.**

Index Terms— **Automatic speech recognition; Command and control systems; Embedded software; Hidden Markov models; Microcontrollers; Real-time systems**

I. Introduction

EVEN though Automatic Speech Recognition (ASR) on embedded platforms has reached performance levels consistent with the requirements of commercial applications, in most cases this performance is achieved at the expense of transferring the complexity of the problem to remote servers. For some applications this represents a problem, since the system requires an internet connection and its speed is strongly limited by the available bandwidth. Furthermore, there are privacy concerns related to data transfers. In this paper we detail a solution which implements the recognition directly on the device.

Hidden Markov Models (HMMs) are one of the most successful approaches to automatic speech recognition. They represent the intrinsic variability of speech signals as well as the structure of spoken language in an integrated and consistent statistical framework [1]. Speech signals are inherently variable and even when people speak the same word, the acoustic signals are not identical. The probability measure represented by HMMs is an essential component of a speech recognition system that follows the statistical pattern recognition approach, and has its root in Bayes' decision theory.

While ASR systems based on HMMs are very common in the literature, the number of works describing implementations on embedded platforms is very limited. This is even more evident if we consider the number of such applications for the Spanish language, developed on existing devices in the consumer market.

In this paper, we describe the implementation of a real-time, speaker-independent isolated speech recognition algorithm using HMMs on a 32-bit ARM Cortex-M4F microcontroller.

Manuscript sent October 9, 2016; accepted November 1, 2016.

M. Marufo da Silva (corresponding author) is with Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Buenos Aires, Argentina (e-mail: mmarufodasilva@est.frba.utn.edu.ar).

D. A. Evin is with Laboratorio de Investigaciones Sensoriales-INIGEM, CONICET-UBA, Buenos Aires, Argentina (e-mail: diegoevin@gmail.com).

S. Verrastro is with Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Buenos Aires, Argentina (e-mail: sebastian_verrastro@yahoo.com.ar).

978-1-5090-2938-9/16/$31.00 ©2016 IEEE

II. Hidden Markov Models

A. Isolated Word Recognition

The characteristics of the pronounced words are represented using the observation vectors sequence O [2]:

$$O = o_1, \dots, o_T \tag{1}$$

where each o_t is the observation vector corresponding to time instant t and T is the total number of observation vectors.

The goal in automatic recognition of isolated words is to solve:

$$arg \max_{1 \le i \le V} \left(P(w_i|O) \right) \tag{2}$$

where V is the number of words in the dictionary and each w_i represents one of them. Applying Bayes' theorem we obtain:

$$P(w_i|O) = \frac{P(O|w_i).P(w_i)}{P(O)} \tag{3}$$

This means that for a set of equal probabilities $P(w_i)$, the most likely word to have been pronounced only depends on $P(O|w_i)$. Due to the wide range of values that the sequence O can take, it is not possible to compute directly $P(o_1, \dots, o_T\,|w_i)$. For this reason, the generation of words is modeled using a parametric model.

B. Word Models with HMMs

A Markov model is a finite state machine in which the transition from one state to another is performed according to a probability. Depending on the state in each time unit, it generates a new observation vector from a probability density output function. An HMM is a double stochastic process: there is an underlying stochastic process that generates the state sequence that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols [2], [3]. Speech production can be represented by these models and in particular, in the isolated speech recognition problem, each word from the system vocabulary is represented by its own HMM [4], [5].

A complete description of an HMM (figure 1) requires the specification of N (number of states in the model), and the probability matrices A, B and π, for which the nomenclature $\lambda = (A, B, \pi)$ is usually used:

- $A = \{a_{ij}\}$ is the state transition probability matrix, where (q_t is the state in which the model is at instant t):

$$a_{ij} = P\big(q_{t+1} = S_j | q_t = S_i\big), 1 \leq i \quad j \leq N \tag{4}$$

$$a_{ij} \geq 0, \forall\, i, j \tag{5}$$

$$\textstyle\sum_{j=1}^{N} a_{ij} = 1, \forall\, i \tag{6}$$

- $B = \{b_j(o_t)\}$ is a matrix with the observation symbol probability distribution, where $b_j(o_t)$ is the emission probability of observation o_t provided the model is in state S_j at instant t. Since we worked with observations of continuous signals, continuous probability density functions were used. Among those functions, the most commonly used are the Gaussian probability density functions with mean vector μ and covariance matrix *Σ* [6], [7], [8].
- $\pi = \{\pi_i\}$ is the initial state probability matrix, where:

$$\pi_i = P(q_1 = S_i), 1 \leq i \leq N \tag{7}$$

In order to solve (2) through (3), the following was assumed:

$$P(O|\lambda_i) = P(O|w_i) \tag{8}$$

The use of HMMs for automatic recognition of isolated words entails two processes:

- Training process (figure 2): there is a training set consisting of several repetitions (different sequences of observation vectors) for each of the V words in the dictionary. For each word, a model λ must be created to represent it, that is, parameters A, B and π of the model must be estimated so that they maximize its likelihood in relation to the word it represents, using the available repetitions of observation vectors sequences corresponding to each word w. This stage is implemented using an algorithm called Baum–Welch [2], [3].
- Recognition process (figure 3): each model's conditional probability $P(O|\lambda)$ (also known as *posterior* probability or *likelihood*) of generating the observation sequence $O = o_1, \ldots, o_T$ corresponding to the word to be recognized must be calculated, and then the model with the highest probability (*maximum likelihood*) is chosen:

$$i^* = arg \max_{1\leq i\leq V}\big(P(O|\lambda_i)\big) \tag{9}$$

- At this stage, an algorithm called *forward* procedure is used to calculate probabilities.

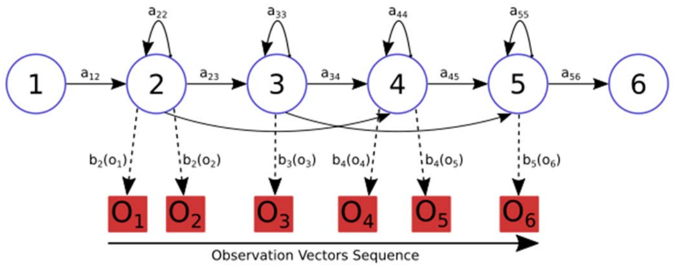

Fig. 1. Representation example of an HMM.

C. *Recognition: forward procedure*

Given model λ, the *forward* variable is defined as the probability of it being in state S_i at instant t and having seen the partial sequence of observation vectors $o_1\, o_2 \ldots\, o_t$:

$$\alpha_t(i) = P(o_1\, o_2 \ldots\, o_t, q_t = S_i|\lambda) \tag{10}$$

The *forward* variable values are calculated at each instant and for each state as follows:

$$\alpha_1(i) = \pi_i . b_i(O_1), 1 \leq i \leq N \tag{11}$$

$$\alpha_{t+1}(j) = \big[\textstyle\sum_{i=1}^{N} \alpha_t(i). a_{ij}\big]. b_j(O_{t+1}), 1 \leq t \leq T-1 \quad 1 \leq j \leq N \tag{12}$$

The last step consists in calculating the *likelihood* $P(O|\lambda)$ as the sum of the *forward* variables at the last instant T:

$$P(O|\lambda) = \textstyle\sum_{i=1}^{N} \alpha_T(i) \tag{13}$$

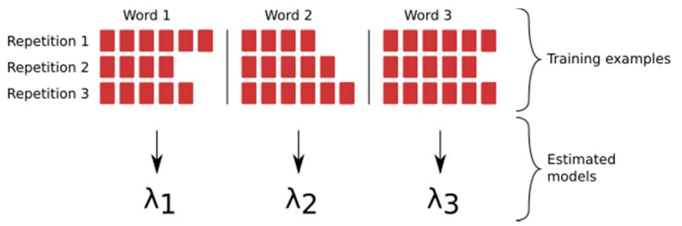

Fig. 2. Training process [2].

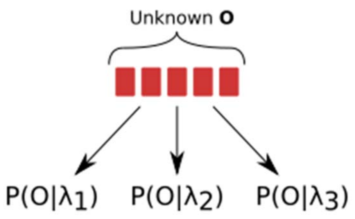

Fig. 3. Recognition process [2].

III. Embedded Implementation

A. Recognition Algorithm

Considering the practical application of the ASR system, it was decided that the training would be done using a PC and that only the front-end and the recognition algorithm would run on the microcontroller.

Due to speech signal redundancies, and inspired by the human auditory processing, most of the ASR systems rely on some type of short time spectral information, extracted in a step called the front-end. In this work, we use Mel-Frequency Cepstral Coefficients (MFCCs), one of the most popular spectral features in the literature [9]. For the task of MFCC parameter extraction, the implementation of the front-end reported in [10] was used, which also computes the energy of the signal, as well as the delta (velocity) and the delta-delta (acceleration) coefficients in order to represent the dynamic characteristics of speech [11], producing a feature vector of 39 parameters in total. To train the HMMs, the HTK toolkit was used [2], while the algorithm corresponding to the *forward* procedure was implemented to run on the microcontroller.

For this last implementation, the Bakis or left-right model [12] was used. The vector of initial probabilities π was set to have a probability 1 of beginning at the first state (and 0 of beginning in any other), and the matrix A was designed so that:

$$a_{ij} = 0, \mathrm{j} < \mathrm{i} \tag{14}$$

$$a_{ij} = 0, \mathrm{j} > \mathrm{i} + 1 \tag{15}$$

Matrix B was defined to have two dimensions, whose sizes correspond to the number of states in the model and the length of the sequence of observation vectors.

In addition, in this kind of Markov models, each state has different Gaussian mixtures associated (which might be regarded as "sub-states"). It has been shown that an HMM state with a probability density function given by a mixture of Gaussians is equivalent to a model of multiple states, each with a single Gaussian function [8].

This is equivalent to first calculating a number of matrices equal to the number of mixtures in the state, and then combining them to form the matrix B. To do so, a coefficients vector c_{im} is added to the model for each state, whose values add up to 1, in order to make a weighted combination of the different sub-states that comprise each state of the model.

Then, for each mixture of each state of each HMM model, and for each instant t of the observation vectors sequence, the value of the probability density function is calculated so that, all together, they form the output probability matrix B, as follows:

$$B_{i,t}(O, \mu, \Sigma) = \sum_{m=1}^{M} c_{im}. f(o_t, \mu_{im}, \Sigma_{im}), \quad \begin{matrix} 1 \leq \mathrm{i} \leq \mathrm{N} \\ 1 \leq \mathrm{t} \leq \mathrm{T} \end{matrix} \tag{16}$$

where N is the number of states and M is the number of mixtures. The Gaussian density function f used was the Multivariate Mixture Gaussian Distribution [11], [13], which is expressed as:

$$f(o_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} exp\left[-\frac{1}{2}(o_t - \mu)'\Sigma^{-1}(o_t - \mu)\right] \tag{17}$$

In this expression, the variables o_t and μ are vectors with length d (d being the number of coefficients used) and Σ, a dxd matrix.

Once the matrix B is calculated, the *forward* procedure is applied in order to determine the likelihood $P(O|\lambda)$ for each model.

B. Scaling

From the analysis of the calculations implemented in the *forward* procedure, it can be observed that each term in the sum corresponds to a product over t, of values from matrices A and B. Since the numbers in these matrices represent probabilities and are therefore in the [0-1] interval, as t increases, the result of the product exponentially tends to 0. This is a problem that, if not solved, generates *underflow* in many of the calculations of the *forward* procedure (especially if working with a single-precision Floating Point Unit, as is the case in the microcontroller adopted), which results in a lower percentage of recognition accuracy. The solution to this problem is to incorporate a scaling procedure [14], which works as follows: for each instant t, once the corresponding values of the *forward* variable $\alpha_t(i)$ are computed for all the N states, they are multiplied by a scaling coefficient calculated as indicated below:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{18}$$

Therefore, for a certain instant *t*, the normalized *forward* variable is calculated as:

$$\hat{\alpha}_t(i) = \alpha_t(i). c_t = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \tag{19}$$

While this procedure allows us to work within the microcontroller's dynamic range, it requires a different calculation for the *likelihood* $P(O|\lambda)$, since the values of $\hat{\alpha}_T(i)$ cannot be added up for the N states because now they are scaled values. That probability is calculated as the inverse of the product of the T scaling coefficients.

$$\prod_{t=1}^{T} c_t . \sum_{i=1}^{N} \alpha_T(i) = \prod_{t=1}^{T} c_t . P(O|\lambda) = 1 \tag{20}$$

$$P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t} \tag{21}$$

However, such probability could still fall outside the available dynamic range and, thus, rather than calculating the probability, its logarithm or *log-likelihood* is used instead:

$$log[P(O|\lambda)] = -\sum_{t=1}^{T} \log(c_t) \tag{22}$$

C. Implementation with logarithmic probabilities

While the scaling procedure is necessary to avoid falling outside the available dynamic range, unfortunately, it is not enough, since the probabilities to estimate the matrix B can often fall out of the valid range as well. Therefore, the complete algorithm was implemented using only logarithmic probabilities.

The main problem when trying to use the logarithm throughout the algorithm was to apply it to the sum of the values. In those cases, the following approximation was used [15], [16]:

$$log(\sum f(x)) \approx log\left(max(f(x))\right) \tag{23}$$

$$log\left(max(f(x))\right) = max\left(log(f(x))\right) \tag{24}$$

Therefore:

$$log(\sum f(x)) \approx max\left(log(f(x))\right) \tag{25}$$

The main points of the complete algorithm implemented logarithmically are presented below:

- Gaussian probability density function:

$$\log(f(o_t, \mu, \Sigma)) =$$

$$= log\left(\frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} exp\left(-\frac{1}{2}(o_t - \mu)'\Sigma^{-1}(o_t - \mu)\right)\right) \tag{26}$$

$$= -\frac{1}{2}(o_t - \mu)'\Sigma^{-1}(o_t - \mu) - log\left((2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}\right) \tag{27}$$

- Output probability matrix B:

$$log\left(B_{i,t}(O, \mu, \Sigma)\right) = log(\textstyle\sum_{m=1}^{M} c_{im}.f(o_t, \mu_{im}, \Sigma_{im})) \tag{28}$$

$$= max\left(log(c_{im}.f(o_t, \mu_{im}, \Sigma_{im}))\right) \tag{29}$$

$$= max\left(log(c_{im}) + log(f(o_t, \mu_{im}, \Sigma_{im}))\right) \tag{30}$$

- Scaling procedure:

$$log(c_t) = log\left(\frac{1}{\sum_{i=1}^{N}\alpha_t(i)}\right) \tag{31}$$

$$= -log(\textstyle\sum_{i=1}^{N}\alpha_t(i)) = -max\left(log(\alpha_t(i))\right) \tag{32}$$

$$log(\hat{\alpha}_t(i)) = log(\alpha_t(i).c_t), 1 \leq \mathrm{i} \leq \mathrm{N} \tag{33}$$

$$= log(\alpha_t(i)) + log(c_t), 1 \leq \mathrm{i} \leq \mathrm{N} \tag{34}$$

$$= log(\alpha_t(i)) - max\left(log(\alpha_t(i))\right), 1 \leq \mathrm{i} \leq \mathrm{N} \tag{35}$$

- *Forward* procedure:

$$log(\alpha_1(i)) = log(\pi_i.b_i(O_1)), 1 \leq \mathrm{i} \leq \mathrm{N} \tag{36}$$

$$= log(\pi_i) + log(b_i(O_1)), 1 \leq i \leq N \tag{37}$$

$$log(\alpha_{t+1}(j)) = log\left(\left(\textstyle\sum_{i=1}^{N}\hat{\alpha}_t(i).a_{ij}\right).b_j(O_{t+1})\right) \tag{38}$$

$$= max\left(log(\hat{\alpha}_t(i)) + log(a_{ij})\right) + log\left(b_j(O_{t+1})\right), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N \tag{39}$$

$$log[P(O|\lambda)] = -\textstyle\sum_{t=1}^{T}\log(c_t) \tag{40}$$

D. Temporal Optimization

In order to obtain the best real-time factor, the variables containing the $log\left((2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}\right)$ results (logarithm of the denominator of the Gaussian probability density function) and of Σ^{-1} (inverse of the covariance matrix) were stored in memory in order to avoid calculating them during the recognition phase. Furthermore, matrix Σ is no longer required.

In addition, to save memory, only significant values were stored for matrices A and Σ^{-1}. Since matrix A contains only 2 diagonals of non-zero values, and matrix Σ^{-1} is a diagonal matrix, this saving has the additional advantage that calculations where they are involved require fewer operations, which reduces recognition time.

E. Memory optimization

Both Matrix α and B from the *forward* procedure have two dimensions and the same sizes. This characteristic, along with the sequences used to perform the calculations in such procedure, allowed us the implementation of a single matrix (of the same size as well) to represent both of them, thus requiring half of the memory. All the RAM memory used in the algorithm corresponds to local variables (stored on the stack) except for the aforementioned integrated matrix, which is allocated dynamically (stored on the heap) since its size depends on the length of the word to be recognized.

In addition, the implemented *forward* procedure does not use the whole transition matrix A but only its two non-zero diagonals to optimize the use of memory. Then, the equation $\sum_{i=1}^{N}\hat{\alpha}_t(i).a_{ij}$, corresponding to the multiplications of the whole matrix A with vector $\hat{\alpha}$, is performed in two steps: one multiplication for each diagonal, and the sum of both products.

Finally, in the calculation of the Gaussian probability density function, just the diagonal of the inverse of the covariance matrix Σ^{-1} is used. This is one of the main advantages of MFCC coefficients, which are usually uncorrelated, and therefore diagonal covariance matrices can be used [2]. Then, the equation $-\frac{1}{2}(o_t - \mu)'\Sigma^{-1}(o_t - \mu)$, corresponding to the calculation of the function exponent is implemented through a multiplication and the accumulation of the components corresponding to each value of the diagonal.

IV. Test Results

The process of training and testing of the HMM models was performed using a database of 11 words in Spanish (numbers “zero” to “ten”) with the following characteristics:

- Data for the training process: 11 speakers (7 male, 4 female), 99 pronunciations for each speaker (3 repetitions of each number at 3 different speeds), with a total of 1089 audios.
- Data for the testing process: 8 speakers (5 male, 3 female), 33 pronunciations for each speaker (3 repetitions of each number), with a total of 264 audios.
- All audios for the training and testing process were recorded at 16KHz, mono and 16 bits.
- The test was performed in an anechoic chamber to minimize the influence of the different types and levels of noise.

First, the training data were used to build various models with different numbers of states and Gaussian mixtures using the HTK tool [2]. Since the total flash memory available in the microcontroller was limited to 1MB, the criterion considered was that the total size of the models should not exceed 500KB, so that remains enough internal ROM memory for the operating system and its applications.

Then, the test was performed using the recognition algorithm implemented in the microcontroller. Models with 8, 16 and 24 states were tested. The results obtained are shown in Table I.

While, as can be observed, the best results were obtained using HMM models with 16 states and 4 mixtures in each state, the final model selected was the model with 16 states and 2 mixtures, because even when the performance is slightly lower than the best model, it requires half the memory that the former.

V. Comparison

To have a relative evaluation of performance for the proposed system, it was compared against a system running on the same platform but based on Dynamic Time Warping (DTW) [10]. DTW is a template matching technique that makes use of dynamic programming to perform time alignment between two utterances in order to derive a meaningful comparison of their similarities [1], [17], [18].

While employing more than one prototype or template by candidate word could improve the recognition rates of the DTW-based recognizer, especially in the case of speaker-independent systems [19], that alternative also increases the memory requirements and processing time of the recognizer. In [20], alternatives for the development of only one prototype for word using various examples of such words were analyzed, and no significant improvements were found in comparison to the alternative of choosing any randomly selected example as the prototype for each word. For the purposes of comparison with the case of the DTW-based recognizer, we selected randomly a pattern for each word to be recognized.

In the comparison, performance was assessed in terms of accuracy (rated with the WER), speed (measured using the real time factor) and required memory for storing the models (HMM) or templates (DTW).

The real time factor (RTF) is defined as [21]:

$$RTF = \frac{Recognition\ Time}{Utterance\ Length} \tag{41}$$

Unlike HMM, where the size of the models does not depend on the duration of the words of the dictionary used but on the number of states and mixtures chosen, in the case of DTW, the size of the templates linearly depends on the duration of the words pronounced. As the size can vary even for different pronunciations of the same word, the durations of the recorded pronunciations were averaged and the size of the templates was calculated based on that value (0.69 seconds).

Table II shows a comparison of the results for HMM and DTW. The HMM analysis was performed only for the speaker-independent case because the number of training examples available for each subject wasn’t enough to train speaker’s dependent systems. However, the DTW analysis was performed both for the independent case and for the speaker-dependent case.

While the DTW-based recognizer exhibited a very good performance in the speaker-dependent case, it shows a clear degradation in the speaker-independent scenario. On the other side, the HMM-based speaker-independent system showed a

TABLE I
Recognition performance in terms of Word Error Rate (WER), and size required for different number of states and mixtures per state

States	Mixtures	WER [%]	Models size [KB]
8	2	11.36	57
	4	5.30	113
	8	6.44	226
	16	6.44	451
16	**2**	**3.79**	**114**
	4	3.41	228
	8	4.17	452
24	2	4.92	171
	4	4.17	340

TABLE II
Recognition performance comparison in terms of WER and RTF for the proposed and baseline systems

Algorithm	WER [%]	RTF	Models/templates size [KB]
HMM (speaker-independent)	3.79	0.37	114
DTW (speaker-dependent)	2.27	0.54 [a]	111 [b]
DTW (speaker-independent)	34.78		

[a] Mean value of various measured repetitions.

[b] For templates 0.69 seconds long.

performance similar to the speaker dependent DTW system, but without imposing the requirement of having samples of the user's speech to build the recognizer.

Finally, considering the real time factor, the HMM system outperform the baseline, requiring approximately 30% less time to perform the recognition; while the memory requirement is similar in both implementations.

VI. Conclusions

This paper describes the implementation of a real-time, speaker-independent isolated speech recognition algorithm using HMMs on a 32-bit ARM Cortex-M4F microcontroller. The evaluation was made using a multi-speaker isolated-digit corpus composed of 11 words of the Argentinian Spanish spoken language, and it was tested against a DTW implementation running under the same platform.

The results obtained with the proposed system showed a high level of recognition, acceptable memory requirements for low or medium size vocabulary-isolated word recognition, and compatible with real time requirements.

The comparison relative to a DTW baseline system showed comparable RTF measures and memory requirements for both algorithms. In the case of speaker-independent recognition, performance in terms of WER was much better for HMM than for the baseline. On the other hand, for speaker-dependent recognition, DTW has a level of recognition similar to HMM, with the advantage of not requiring a large database to train its models.

VII. Future Work

Future work will cover the development of new method for isolated word recognition under the same platform. This method is under development and is based on a Support Vector Machine (SVM) - HMM hybrid system, with the aim of combining the discriminative power of SVM algorithms, with the temporal modeling properties of HMMs [22].

Acknowledgment

The authors would like to thank the collaboration of members of the Laboratorio de Investigaciones Sensoriales (INIGEM, CONICET-UBA) for the creation of the database used in this work.

References

[1] B. H. Juang and L. R. Rabiner, "Automatic speech recognition–a brief history of the technology development," Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara, 1, 2005.

[2] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, P. Woodland, P. (2006). The HTK book. Cambridge University Engineering Department.

[3] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE 77.2 , 1989, 257-286.

[4] L. R. Rabiner and B. Juang, Fundamentals of speech recognition, First. Prentice-Hall International, Inc., 1993.

[5] D. Jurafsky and J. Martin, Speech & language processing, Second. 2009.

[6] L. Liporace, "Maximum likelihood estimation for multivariate observations of Markov sources," Information Theory, IEEE Transactions on28.5 (1982): 729-734.

[7] B-H. Juang, "Maximum-Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains," AT&T technical journal 64.6 (1985): 1235-1249.

[8] B-H. Juang, S. E. Levinson, and M. Mohan Sondhi, "Maximum likelihood estimation for multivariate mixture observations of markov chains (corresp.)," Information Theory, IEEE Transactions on 32.2 (1986): 307-309.

[9] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition," Speech Communication 54, no. 4, 2012, 543-565.

[10] A. G. Alvarez, D. A. Evin and S. Verrastro, "Implementation of a Speech Recognition System in a DSC," in press.

[11] Xuedong Huang , Alex Acero , Raj Reddy , Hsiao-Wuen Hon, Spoken Language Processing: A Guide to Theory, Algorithm, and System Development, Prentice Hall PTR, Upper Saddle River, NJ, 2001

[12] R. Bakis, "Continuous speech recognition via centisecond acoustic states," The Journal of the Acoustical Society of America 59.S1 (1976): S97-S97.

[13] J-L. Gauvain and C-H. Lee, "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," IEEE transactions on speech and audio processing 2.2 (1994): 291-298.

[14] S.E. Levinson, L.R. Rabiner and M.M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," The Bell System Technical Journal 62.4 (1983): 1035-1074.

[15] S. Astrov (2007). Optimization of algorithms for large vocabulary isolated word recognition in embedded devices (Doctoral dissertation, Technische Universität München).

[16] P. Paramonov and N. Sutula, "Simplified scoring methods for HMM-based speech recognition," Soft Computing (2014): 1-6.

[17] M. Müller, "Dynamic time warping," in Information retrieval for music and motion, 2007, p. 318.

[18] S. Salvador and P. Chan, "FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space," Intell. Data Anal., vol. 11, no. 5, 2007.

[19] J. Wilpon and L. Rabiner, "A modified K-means clustering algorithm for use in isolated work recognition," IEEE Transactions on Acoustics, Speech, and Signal Processing 33.3 (1985): 587-594.

[20] M.S. Barakat, C.H. Ritz and D.A. Stirling, "An improved template-based approach to keyword spotting applied to the spoken content of user generated video blogs," 2012 IEEE International Conference on Multimedia and Expo. IEEE, 2012.

[21] Yu, D. and Li D. Automatic Speech Recognition. Springer, 2012.

[22] A. Ganapathiraju, J. E. Hamaker and J. Picone, "Applications of support vector machines to speech recognition," Signal Processing, IEEE Transactions on, 52(8), 2004, 2348-2355.