A TMS320C40 based Speech Recognition System for Embedded Applications

Bernhard Obermaier and Bernhard Rinner

Institute for Technical Informatics
Technical University of Graz, AUSTRIA

[obermaier, rinner]@iti.tu-graz.ac.at

.

ABSTRACT:

This paper describes a prototype implementation of a speech recognition system for embedded applications. The recognition system is comprised of a feature extractor and a classifier. The feature extractor is based on a 64-point Fast Fourier Transformation (FFT); the classifier is based on discrete-density Hidden Markov Models (HMM) with a variable codebook size. Training as well as classification are implemented using the Viterbi algorithm. The prototype is implemented on a digital signal processor (DSP) of type TMS320C40 from Texas Instruments. The recognition rate and the performance are experimentally evaluated using a test vocabulary of 20 words.

keywords: **automatic speech recognition, Hidden Markov Models, TMS320C40**

1. INTRODUCTION

Automatic speech recognition (ASR) has been a very active research area for a long time, and much progress has been achieved within the last years. Nowadays, many applications have been extended by ASR techniques to enable an easy and natural way of human computer interaction. ASR applications can be classified into two categories. Large vocabulary systems are capable of understanding a large number of different words. Such large vocabularies are required for dictionary systems where the ASR system ideally recognizes all words from fluently spoken sentences [8]. On the other hand, command and control applications [2] recognize only a few specific words (commands) required to control the application. Examples of command and control applications are speech-controlled telephone information systems and speech-controlled user interfaces.

Command and control ASR is often integrated into systems which are tightly coupled with their environment, e.g., a speech-controlled robot. In such an *embedded system*, speech recognition must be performed with limited resources, like processors and memory. Furthermore, the recognition must be often completed within a limited period of time.

The work presented focuses on the development of a command and control ASR system for embedded applications. Thus, the main objectives of this project are (i) to recognize a limited number of commands, (ii) to enable speech recognition in limited time and with a limited amount of memory, (iii) to demonstrate the recognition online and (iv) to design a modular system such that the individual components can be easily changed and improved.

The remainder of this paper is organized as follows. Section 2 introduces some background of speech recognition. Section 3 presents the implemented TMS320C40-based ASR system, i.e., some design considerations and implementation details are given, and the ASR system is evaluated using experimental data. A brief discussion concludes the paper.

2. AUTOMATIC SPEECH RECOGNITION

2.1 Overview

Figure 1 presents an overview of an ASR system. In general, speech (words) are recognized using the following procedure.

The analog speech signal is captured by a microphone and sampled by an analog to digital converter (ADC). To reduce the high data rate, characteristic features are generated by the feature extractor. These features represent the speech signal in a very compact form. They are used in the final step of an ASR system to determine which word has actually been spoken. The classifier compares the features to models (templates) of the words of the vocabulary. The classification is based on some distance metric between the features and the templates. The templates are generated during the training of the ASR system; a template is generated for each word.

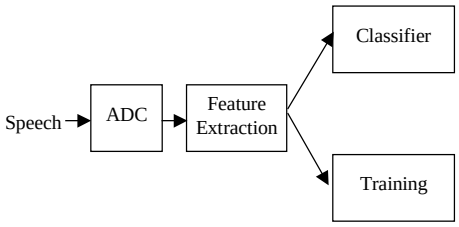

Figure 1: Overview of an ASR system

2.2 Classifier

The classifier is one of the most important components of an ASR system, i.e., it compares the unknown utterance consisting of T feature vectors with the stored templates. We briefly describe the two most commonly-used classifiers, *vector quantization* and *hidden Markov models* [5].

Vector Quantization

The idea of vector quantization (VQ) is to represent each word of the vocabulary by a limited number of feature vectors. Thus, the feature vectors have to be *quantized* using a given number of representative vectors (codebook). After VQ, a feature vector is represented by the codebook vector which has the smallest distance to the original feature vector. An optimal codebook is generated for each word during the training, i.e., the codebook vectors are determined such that the distance between the codebook vectors and the feature vectors is minimized. The distance for T feature vectors is defined by

$$D = \sum_{T} d(x_t, \hat{x}_t) \ ,$$

where x_t is the feature vector and $\hat{x}_t$ is the codebook vector which has the smallest distance to x_t.

The expected distance between the optimal codebook and the corresponding utterance is smaller than the distance to any other codebook. This property leads directly to the following classification method (Figure 2):

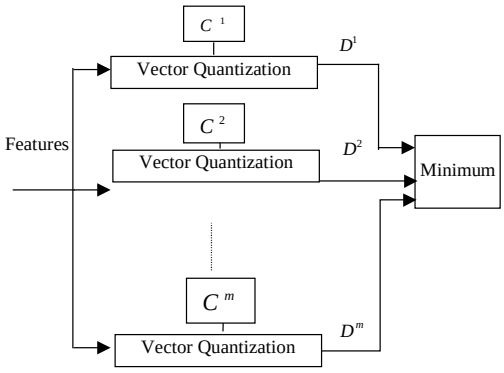

Figure 2 : VQ classifier

The feature vectors of the unknown utterance are quantized using each codebook and the distances D^i to each codebook are calculated. The codebook with the minimum distance determines the classified (recognized) word.

Hidden Markov Model

Hidden Markov models (HMM) are used to describe stochastic processes. The underlying assumption for applying HMMs in ASR is therefore that the speech signal can be well characterized as a parametric random process [6]. An HMM can be informally described as a finite automaton which additionally emits feature vectors at each state. The probability for a transition from state i to state j is defined as a_{ij}; the probability for the emission of feature vector k of a global codebook at state j is

defined as b_{jk}. Hence, the two matrices $A = \{a_{ij}\}$ and $B = \{b_{jk}\}$ define an HMM. A typical HMM consisting of 4 states is depicted in Figure 3. Black arrows represent the transition probabilities and white arrows represent the emission probabilities.

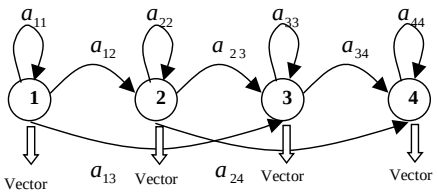

Figure 3: Hidden Markov Model with 4 states

By applying the following procedure, an HMM can also be seen as a generator for feature vectors: start at the initial (leftmost) state and traverse through the HMM using A and emit a feature vector at each state using B. The probability of generating a specific sequence of feature vectors can be calculated by the *Viterbi algorithm* [6].

The training aims at generating an HMM for each word, i.e., the matrices A and B. Classifying an unknown utterance is basically the calculation of the generation probability of the utterance's feature vector sequence for each word. The word corresponding to the HMM with the highest probability is then selected. Vector quantization using a global codebook is applied to limit the number of possible emission feature vectors. An HMM classifier is shown in Figure 4.

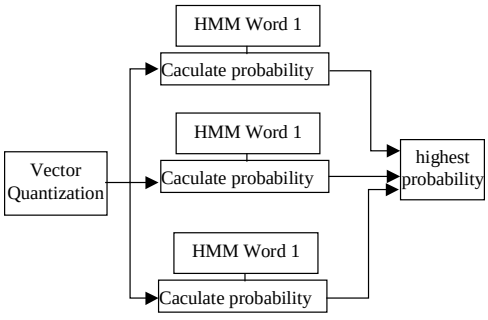

Figure 4: HMM classifier

3. AN ASR SYSTEM FOR EMBEDDED APPLICATIONS

In this section we describe our ASR system for embedded systems. First, we present some design considerations for the feature extractor and the classifier. Then, we describe a prototype implementation using a TMS320C40 DSP. Finally, we present an experimental evaluation of the recognition rates and the runtime performance.

3.1 Design Considerations

Feature Extractor

To keep the feature extractor as simple as possible, the feature vectors are based on the power spectrum of the speech signal using a Fast Fourier Transformation (FFT). However, the energy of the recorded speech changes over time, e.g., due to a different distance to the microphone or a different volume of the speaker.

To make the feature extractor more robust to the energy variation of the utterances, a *logarithmic* power spectrum is used for the feature vectors, i.e., the logarithm of the FFT coefficients [4,5]. The logarithmic power spectrum of the same utterances with distinct energy differs only by an offset. To get rid of this (variable) offset, the mean value of the logarithmic power spectrum is calculated and subtracted from the spectrum. This results in nearly constant feature vectors of the same utterances with different energies.

Classifier

Our ASR system is indented to be used for command and control in embedded applications. The vocabulary is quite small in such applications. However, the memory and computing restrictions might be very challenging, and the classifier has a great influence on required computing resources. The decision which classifier is actually applied in our ASR system is based on a comparison of the VQ and HMM classifier with regard to memory and runtime complexity.

The following comparison [3] presents first the theoretic memory and runtime complexities and determines then the bounds for a specific recognition rate. Following notation is used for this comparison: m represents the total number of words (utterances), c defines the codebook size and v determines the size of the feature

vectors. The number of states of the HMM is given by N.

- VQ memory complexity

  m codebooks are required to store all utterances. Each codebook consists of c codebook vectors each of size v. Hence, the memory complexity is given by $O(c*v*m)$.

- VQ runtime complexity

  The runtime performance for classifying is determined for one feature vector. The distance of two vectors is based on the Euclidean metric. Thus, this distance calculation requires v operations. Distances are required for all codebook vectors in all codebooks. Therefore, the total runtime is given by $O(c*v*m)$.

- HMM memory complexity

  The HMM classifier requires one global codebook and an HMM for each word. The memory complexity for the codebook is $O(c*v)$. One HMM model consists of the transition probability matrix A of size $N*N$ and the emission probability matrix B of size $N*c$. Thus, the total memory complexity is $O(c*v+m*(N^2+N*c))$.

- HMM runtime complexity

  The HMM classifier quantizes first each feature vector and then calculates the generation probability for each HMM. The runtime complexity of VQ for one feature vector is given by $O(c*v)$. The Viterbi algorithm has a runtime complexity of $O(N^2)$. Therefore, the overall complexity is $O(c*v+m*N^2)$.

To achieve a recognition rate of 97 % for m=20 words with a feature vector size of v=32, a codebook size of c=16 is required. The HMMs are modeled using N=4 states [3].

Classifier	*Memory*	*Runtime*
VQ	10240	10240
HMM	3944	1344

Table 1: Comparison of the VQ and HMM classifier

Table 1 presents a comparison of both classifiers using these parameters. The HMM classifier requires much less memory and runtime than the VQ classifier. Therefore, HMM classification is applied in our ASR system.

3.2 TMS320C40 based Implementation

Our ASR system is implemented on a PC with a data acquisition board from ADAC (DHR5403) and a TIM40 motherboard from Transtech (TDMB412). The TIM40 motherboard is equipped with one TIM40 module (TMS320C40). Software has been mostly developed in ANSI C using the CodeComposer from GoDSP [7].

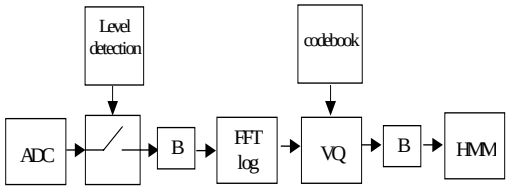

Figure 5: Block diagram of the prototype

A block diagram of the overall ASR system is depicted in Figure 5. The speech signal is sampled by the data acquisition board at a sampling rate of 10 kHz and a resolution of 16 bit. The samples are directly transferred to the TIM40 module via a commport of the TMS320C40. A level detector decides whether a word is spoken. If the short time energy of the samples exceeds a limit, the samples from the data acquisition board are written to a buffer for a specific period of time. The spectrum of the signal stored in the buffer is calculated by an assembler-optimized 64-point FFT algorithm. The sample frames are overlapped by 30 % for successive Fourier transformations. The feature vector is generated by the FFT coefficients (compare Section 3.1) and has a size of 32. Each feature vector is quantized using the global codebook. Indices of the corresponding codebook vectors are stored in a second buffer. The HMM classifier generates finally the probability of the sequence of indices by a Viterbi algorithm.

The training can be divided into two tasks, the optimization of the global codebook and the generation of the word models . The optimization of the codebook vectors is performed by the binary split algorithm [6]. After the codebook optimization, the HMM models for all words are trained by the Viterbi algorithm. In our prototype implementation training [1,3] is performed off-line, i.e., the generated feature vectors of the test data are transferred from the TMS320C40 to the PC. The actual training is then performed on the PC. Before our ASR system is able to recognize words, the trained HMM models and the global codebook have to be downloaded into the memory of the TMS320C40 by the CodeComposer.

In our prototype we exploit the display capabilities of the CodeComposer for the presentation of the classification results. The probabilities of all words of the vocabulary are plotted in a diagram. By this representation the confidence of a recognition can be easily seen. The logarithmic probabilities are actually displayed. An example of this representation is shown in Figure 6. The word with index 7 is actually selected in this example.

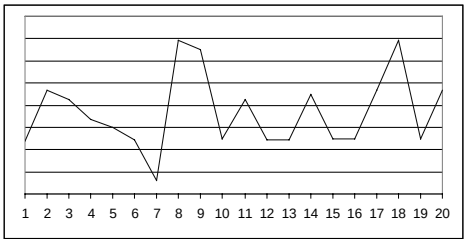

Figure 6: Presentation of the word probabilities

3.3 Experimental Results

The experimental evaluation of our prototype is divided into two parts. First, we evaluate the recognition rate and then we present the runtime performance.

Recognition evaluation

The test set for our recognition evaluation consists of a vocabulary of 20 words. Examples for these words are simple commands, like "stop" and "go", and the numbers "0" to "9". For each word 10 different utterances have been recorded during a period of 5 days. Hence, the overall training set consists of 200 utterances. We used only 5 different utterances per word for the training, and we used all utterances for the classification. Therefore, we were able to evaluate the recognition of trained as well as untrained data. The recognition rate was evaluated dependent on different model parameters, i.e., the size of the codebook c and the number of states N .

In Figure 7, the recognition error (1-*recognition rate*) of the HMM classifier is shown with codebook sizes of 16, 32, 64 and 128.

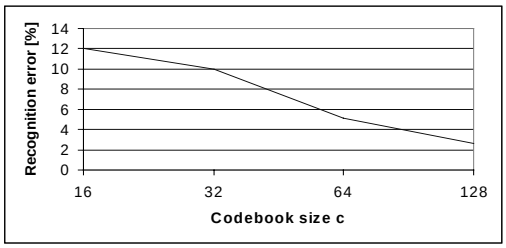

Figure 7: Recognition error vs. codebook size

Measurements were taken with different numbers of states ranging from 1 to 9. The presented recognition error in Figure 7 is the average of recognition errors measured with different numbers of stages. The recognition error decreases monotonically with increasing codebook size from 12 % to nearly 2 %. This monotonic behavior is not a surprising result because a higher number of codebook vectors reduces the quantization error and, hence, the distance of the feature vectors.

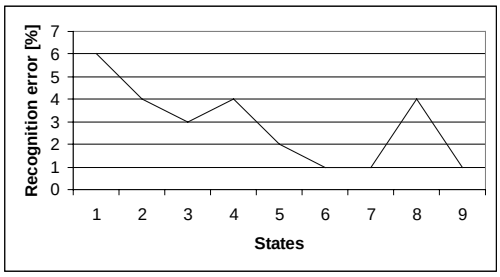

Figure 8: Recognition error vs. states

Figure 8 presents the recognition error dependent on the number of states N of the HMMs. The codebook has a fixed size of c=128, and the number of states N are varied from 1 to 9. In this case the recognition error does not show such a monotonic behavior as in Figure 7. This is

due to the fact that there is a tight relation between the number of states and the phonems in the words. Best recognition rates are achieved when the number of states corresponds to the number of phonems. Increasing the number of states makes the recognition worse. For our vocabulary, best results are achieved using HMMs with 6, 7 or 9 states.

Performance evaluation

Our performance evaluation is based on runtime measurements on our prototype. The runtimes were measured using the timing utilities of the CodeComposer. In our prototype the recognition is implemented in three consecutive steps: feature extraction, vector quantization and probability calculation (classification).

The recognition including these three steps was measured for a typical word of our test vocabulary, using a codebook size $c = 32$ and a number of states $N = 5$. 103 feature vectors were generated for this word that is equivalent to an utterance time of 0.6 s. The runtimes for this word are shown in Table 2. The total time required to recognize this word is 738 ms.

Since the runtime of all recognition steps is linear with the number of feature vectors of the utterance, the total recognition time can be determined by the length of the vocabulary's word.

Recognition step	*Runtime*
Feature extraction	183ms
Vector Quantization	264ms
Classifying	291ms
Total	738ms

Table 2: Runtime performance

4. CONCLUSION

We presented a prototype implementation of an ASR system for command and control applications. This prototype allows online recognition with limited memory and runtime. A recognition rate of 99 % was achieved by using a test vocabulary of 20 words.

Future work is directed towards the improvement of our ASR prototype, i.e., to enable speaker independent recognition and to advance the feature extractor and the classifier, as well as the application in an embedded system, like the speech control of an autonomous robot.

ACKNOWLEDGEMENTS

The authors are grateful to Alexander Haymaier and Martin Proßnigg who implemented the communication between the ADC and the TMS320C40 as well as the Fourier transformation.

REFERENCES

[1] Roland Auckenthaler. *Implementing a Hidden Markov Model for time-variant Feature Vector Processing*. Technical Report. Institute for Technical Informatics, TU Graz. 1998.

[2] Judith A. Markowitz. *Using speech recognition.* Prentice Hall. 1996.

[3] Bernhard Obermaier. *Entwurf und Implementierung eines Spracherkennungs - systems auf einem TMS320C40.* Master Thesis. TU Graz. 1998.

[4] Alan Oppenhaim and Roland Schafer. *Zeitdiskrete Signalverarbeitung* Oldenbourg Verlag München, Wien. ISBN 3-486-22948-6. 1995.

[5] Joseph Picone. Signal Modelling Techniques in Speech Recognition. *In Procedings of the IEEE.* Vol 81, No.9. September 1993.

[6] Lawrence Rabiner und Biing-Hwang Juang. *Fundamentals of Speech Recognition.* Prentice Hall PTR. ISBN 0-13-015157-2. 1993.

[7] Texas Instruments. *TMS320 Floating Point DSP Optimizing C Compiler.* 1995.

[8] Steve Young. A Review of Large-vocabulary Continous speech Recognition. *IEEE Signal Processing Magazine.* September 1996.