Rapid Porting of ASR-Systems to Mobile Devices

Thilo W. Köhler, Christian Fügen, Sebastian Stüker, and Alex Waibel

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe (TH), Karlsruhe, Germany
{tkoehler,fuegen,stueker,waibel}@ira.uka.de

10.21437/Interspeech.2005-116

Abstract

Portable devices for the consumer market are becoming available in large quantities. Because of their design and use, human speech often is the input modality of choice, for example for car navigation systems or portable speech-to-speech translation devices. In this paper we describe our work in porting our existing desktop PC based speech recognition system to an off-the-shelf PDA running WindowsCE3.0. We do this in a way that our already well performing language and acoustic models can be taken over without the need of retraining them for the PDA. In order to achieve an acceptable run-time behavior we apply several optimization techniques to the preprocessing and decoding process. Among other things we introduce the newly developed early feature vector reduction. In that way the execution time of our recognition system can be reduced from initially 28x real-time to 2.6x real-time with a tolerable increase in word error rate. The size of the acoustic models is reduced to 25% of its original size.

1. Introduction

Over the last couple of years mobile electronic devices have become available for the consumer market in large quantities. Nowadays portable GPS based navigation units, cell phones, personal digital assistants (PDAs), and electronic translation devices are common appliances in every day life. These devices are often being used in situations where their control by means of manual input, e.g. through a keyboard or pen, is difficult or impossible, for example when driving a car. In such situations human speech very often is a suitable input modality. Also, human speech often makes for a more natural input mode. A PDA running a translation program that takes human speech as input and gives its translation output also in the form of human speech will be much more convenient to the user than one that requires keyboard or pen input.

Though the computational power of small devices, such as PDAs, has increased significantly, it is still no match for the capabilities of today's desktop machines. State-of-the-art large vocabulary conversational or spontaneous speech recognition systems that offer good performance in acceptable run-time still require the computational power and memory resources of full grown desktop machines. Porting them to mobile devices therefor makes changes to their system design necessary.

In [1] Viikki gives an overview on the topic of automatic speech recognition (ASR) on mobile devices. Though it is often possible to transmit the audio signal from the PDA to a server for recognition, in the light of increased computational power of mobile devices and the gained independence from a server connection, it is desirable to have the recognition process performed by the mobile device directly. Since the decoding, especially the score computation of the acoustic model, is the most expensive part during the recognition process, many speed and memory optimizations target the computation of the Gaussian Mixtures, for example the Bucket Box Intersection Algorithm [2]. As Novak mentions in [3], the memory bandwidth is also a speed limiting factor. State-Clustered Tied-Mixture (SCTM) HMMs [4] can reduce the size of the acoustic model while accelerating its computation, but requires specifically trained models. For faster computation of Mahalanobis distances, Vasilache suggests to use look-up tables [5]. [6] uses integer computation for decoding and preprocessing instead.

In this work we present our approach in porting the Janus Recognition Toolkit (JRTk) [7] featuring the IBIS decoder [8] to an off-the-shelf PDA running WindowsCE 3.0. We do this in such a way that we do not need to train new models, but rather can take the models of an already existing and well performing system. One of the most prominent limitations of todays hand-held devices is the lack of a floating point unit (FPU), due to limitations in battery supply and price of the PDA. By default, floating point operations are being emulated by software. Since in an ASR system floating point operations occur very frequently during preprocessing and decoding, and because their default software emulation is very time intensive, one of the major tasks is to either reduce their frequency in the PDA system or to replace them by integer operations. Another limiting factor is the small bandwidth of the memory access. During decoding the CPU has to process data with bad locality characteristics. At the same time the data cache of mobile devices is very small, usually ranging from 8KB up to 32KB. One way to alleviate this problem is to compress the models and data of the recognition system by quantizing them.

Chapter 2 introduces our recognition system that we ported to the PDA and the performance of the naive port without any modifications to the recognition process. In chapter 3 we describe our experiments in improving the execution time of the preprocessing, while chapter 4 describes the modifications to the decoder. Finally in chapter 5 we demonstrate how the individual modifications can be combined in order to achieve a good trade-off between execution time and recognition accuracy.

2. Baseline System Description

2.1. Task and Test Data

In our work we ported a recognition system for spontaneous requests and commands to a navigation system for the city of Karlsruhe, Germany [9]. Beside the development data of about 20 minutes from 11 speakers, another 52 minutes of German speech were collected from 4 female and 9 male native speakers for evaluation. Recordings were made with a sampling rate of 16kHz and a 16bit resolution with the integrated microphone of the PDA using a push-to-talk mechanism. We used an iPAQ

H5550 Pocket PC with an Intel XScale 400MHz processor for our experiments.

2.2. Recognition System

Our originally desktop based system is a three-state subphonetically tied semi-continuous recognizer composed of 2000 distributions with as many codebooks with 16 Gaussians each that take 32 dimensional Mel Frequency Cepstral Coefficients (MFCC) after linear discriminant analysis (LDA) as input. The language model is a semantic, context free grammar with about 300 rules. The search vocabulary contains 2500 words, of which 2000 are names of streets, city areas, and places of interest. Utterance based mean subtraction is applied during decoding. When porting the system to a PDA without applying any optimization, it runs in 28 times real time and gives a word error rate (WER) of 19.7% on the evaluation data.

3. Optimization of Preprocessing

Figure 1 shows the preprocessing of the baseline system which runs in about 0.7 times real time. When aiming for a recognizer running in real time this is too slow. In Figure 1 every preprocessing step is labeled with the time it took in average for processing 1s of audio signal. The optimizations we describe below will focus on processing steps marked by boxes with a solid outline in the figure.

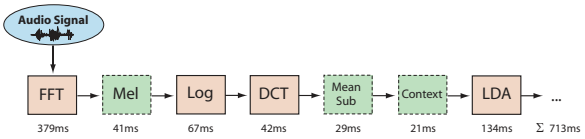

Figure 1: Preprocessing steps and run-time for 1s of audio. Optimization will focus on boxes with solid outlines.

3.1. Integer Matrix Multiplication

The matrix multiplication $\mathbf{A}^{(m_A \times n)} * \mathbf{B}^{(n \times m_B)} = \mathbf{C}^{(m_A \times m_B)}$ needs $O(n^3)$ multiplications and additions. Optimizations such as the method of Strassen that assume that additions are cheaper than multiplications but do not reduce model complexity, are of little use, as software emulated floating point additions are not significantly cheaper than multiplications. We perform matrix multiplications with the naive implementation, however using integer operations, by converting the matrices into integer values, then perform the matrix multiplication and convert the result back to floats. When using converted integer values, the accuracy of the computation is lower and heavily dependent on the value range of the matrices $\mathbf{A}$ and $\mathbf{B}$.

In order to make good use of the integer value range while avoiding integer overflows at the same time, the coefficients of both matrices must be scaled by s_A and s_B before they are converted to integers, scaling back the result accordingly. This scaling needs $O(n^2)$ floating point multiplications. Let $\lfloor ... \rfloor$ denote the error of typecasting from floating points to integer values. Then we approximate $\mathbf{C}$ as:

$$\mathbf{C} \approx \frac{1}{s_A s_B}(\lfloor s_A \mathbf{A} \rfloor * \lfloor s_B \mathbf{B} \rfloor) \tag{1}$$

A conservative estimation of s_A and s_B that avoids integer overflows is to find the maximum coefficients $\hat{a}$ and $\hat{b}$ and to choose the scaling factors in a way that $s_A \hat{a} \cdot s_B \hat{b} \leq \frac{INT_{max}}{n}$. However, considering that the maximum coefficients of the matrices do not necessary meet each other during multiplication, the scaling factors can be better approximated by:

$$s_A = \sqrt{\frac{INT_{max}}{\sum_{k=1}^{n} \max_i a_{ik} \max_j b_{kj}} \cdot \frac{\hat{b}}{\hat{a}}} \tag{2}$$

$$s_B = \sqrt{\frac{INT_{max}}{\sum_{k=1}^{n} \max_i a_{ik} \max_j b_{kj}} \cdot \frac{\hat{a}}{\hat{b}}} \tag{3}$$

In order to avoid the calculation of s_A and s_B for every new matrix multiplication, the scalers are estimated only once on similar data beforehand. Since working in two's complement space, the individual series of additions can overflow during computation as long as the final result still fits within the integer value range.

In our system the integer matrix multiplication is used for the DCT and the LDA steps of the preprocessing. Table 1 shows that the recognition performance of the recognizer does not degrade while the execution time of the matrix multiplication decreases significantly. The achieved acceleration depends on the size of the matrices, as the overhead for converting the matrices to integer values is $O(n^2)$ while the matrix multiplication takes $O(n^3)$.

Matrix Multiplication	DCT	LDA	WER
Baseline	42ms	134ms	19.7%
Integer Mat. Mul.	5ms	10ms	19.7%

Table 1: Speed and WER of the matrix multiplication.

3.2. Approximation of Logarithm

The evaluation of the $\log$ function usually requires many floating point operations. The emulation by software of these operations is especially expensive, since higher numeric accuracy than provided by integers is needed. To obtain a fast approximation of the $\log$ function consider the internal representation of a floating point number according to the IEEE-754 standard:

$$f = s \cdot m \cdot 2^x \tag{4}$$

where s is the sign, m the mantissa normalized to $[1, 2)$ and x the exponent. The exponent can be extracted by simple bit masking and shifting and already represents a coarse approximation of $\log_2 f$. It can be turned into $\log_b f$ to any base b by multiplication with $\log_b 2$ as shown in equation (5). The error is always lower than $\log_b 2$, but can be improved by additionally approximating the logarithm of the mantissa m. By using a polynomial of first (6) or second (7) degree the computation is reduced to a few multiplications and additions:

$$\log_b f \approx \log_b 2 \cdot x \tag{5}$$

$$\log_b f \approx \log_b 2 \cdot (m - 1 + x) \tag{6}$$

$$\log_b f \approx \log_b 2 \cdot \left(\frac{1}{3}(m-1)(5-m) + x\right) \tag{7}$$

Table 2 shows the word accuracy of our baseline system when replacing the logarithm with approximation (7). We decided to use this approximation for our experiments, because it has the best numerical accuracy, already has a negligible runtime and gave a slightly better recognition performance on the development data.

Log calculation	Speed	WER
Baseline	67ms	19.7%
2nd degree polynomial	9ms	19.8%

Table 2: Speed and WER of the logarithm approximation.

3.3. FFT Using Intel Performance Primitives

For speeding up the FFT computation in our system, we used the Intel Performance Primitives (IPP)[10], a function library optimized for different types of Intel processors. In the IPP the FFT is performed on 16bit integer values as input and 16bit integers as output. However, the accuracy of the implementation seems to be very low. The best accuracy was observed by normalizing the input signal to 2% of the 16bit value range. One explanation for this could be, that exceedingly high values cause integer overflows during the calculation. Because of the lower accuracy of the FFT the word error rate increases slightly as shown in Table 3; but at the same time the execution time is sped up significantly.

FFT	Speed	WER
Baseline	379ms	19.7%
IPP	25ms	21.6%

Table 3: Speed and WER of the FFT.

4. Optimization of Decoding

Decoding the feature vectors to a textual hypothesis is the most expensive part in ASR. Especially the evaluation of the Gaussian Mixtures uses a lot of floating point operations for computing the Mahalanobis distances. The use of look-up tables can avoid these computations [5] either in part or completely. But on a PDA the necessary additional memory access is more expensive than doing the computation by means of integer arithmetic. Figure 2 shows the execution time of the decoding for the direct system port without any PDA specific optimizations.

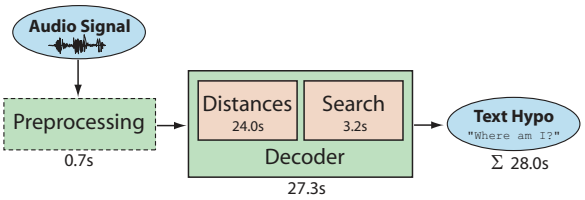

Figure 2: Preprocessing and decoding run-time for 1s of audio.

4.1. Mahalanobis Distances Using Integer Computation

Computing the Mahalanobis distances is one of the most expensive parts during decoding using many floating point operations. In order to apply integer operations instead the codebook is converted into an integer representation. Similar to the matrix multiplication, scalers for the means and the covariances are used to better cover the integer value range. Since the range of the values encountered during the calculation can be robustly estimated beforehand, the scalers can also be precomputed. A better accuracy for the majority of the values can be reached by using a higher scaling factor, that leads to clipping of only few numbers with high absolute values. We tuned the scalers on the development data. The best trade-off on 16bit values was reached, when less than 10% of the covariances and nearly no mean values were clipped. The 16bit values of means and covariances can be stored interleaved in order to be fetched by one single 32bit memory access. In combination with rolling out the innermost loop of the distance calculation, this leads to a gain in speed of 10-25% relative.

Quantisation	Distances	Search	Memory	WER
Baseline	24.0s	3.2s	19MB	19.7%
16bit	2.0s	3.2s	15MB	19.9%
8bit	1.8s	3.1s	13MB	19.5%

Table 4: Speed, memory and WER comparison of 8 and 16bit quantized codebooks using the integer score function.

It can be seen in Table 4, that no significant increase in WER is caused by using the integer score function and a 16bit or 8bit representation of the codebooks. But the speed can be improved by a factor of more than 10 times. If the codebook is not quantized on the fly, but precomputed and stored as integer values, the size of the acoustic model can be reduced down to 50% (16bit) or 25% (8bit). The overall memory footprint of the recognizer changes accordingly.

Quantizing the codebooks to 8bit instead of 16bit increases the recognition speed further. Now 2 means and 2 covariances can be fetched with one 32bit memory access, leading to a speed-up of ∼10% and a memory reduction of 50% compared to the 16bit quantization (see Table 4).

4.2. Early Feature Vector Reduction (EFVR)

The idea behind the EFVR is to reduce the number of feature vectors, before decoding. Similar approaches like the Variable Frame Rate Algorithm [11] or the Conditional Frame Skipping [12] are picking out frames that carry potentially new information while skipping similar frames. In contrast to that we do not skip frames, but merge them by taking their arithmetic mean. To compare neighbored feature vectors, we use a modified Euclidean distance in which lower coefficients are weighted higher. This is motivated by the fact that the coefficients in the feature vector where arranged by the LDA according to their discriminative capabilities.

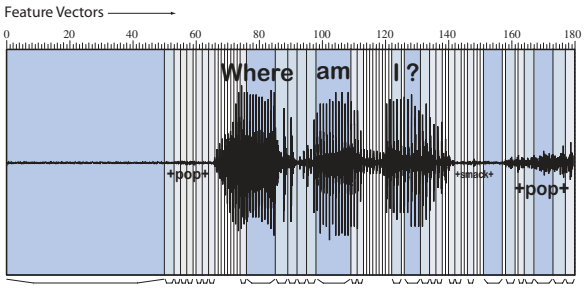

Figure 3: EFVR on the utterance "Where am I ?": It can be seen that the 180 feature vectors are reduced to ∼60 vectors, mainly in silence or stationary areas.

Typically, passages of silence are merged together, but also longer vowels or other static sounds, as it can be seen in Figure

3. Also, the spreading factor of the search network is locally lower depending on the number of vectors that are merged because the confidence about the merged vector is expected to be higher. Generally, the number of active states can be reduced down to 20-50%, but heavily depends on the characteristics of the input audio data. Tighter segmentation or more reverbration lower amount of merged vecotrs.

Combination	Distances	Search	WER
Baseline	24.0s	3.2s	19.7%
EFVR	21.2s	2.8s	20.2%

Table 5: Speed, memory and WER comparison of the EFVR.

Table 5 shows the effect the EFVR has on the execution time of our recognition system.

4.3. Bucket Box Intersection (BBI)

A further speed-up technique based on Gaussian selection used in our decoder is BBI [2]. Using a BBI-Tree with a depth of 6 a good improvement in speed can be achieved. However the WER increases by $\sim$7.6% relative, as shown in Table 6. It has also a significant memory overhead.

	Distances	Search	Memory	WER
Baseline	24.0s	3.2s	19MB	19.7%
BBI	8.1s	3.1s	21MB	21.2%

Table 6: Speed, memory and WER comparison when using a BBI-Tree.

5. Combination of the Optimizations

In the previous two chapters every optimization was tested independently from the others for a better comparison of the several techniques. In this chapter we show the improvements in decoding speed and memory usage when combining the various techniques. From observations of the performance on the development data, we chose different combinations of optimizations, trying to find a good trade-off between speed and WER.

Combination	Preproc.	Dist.	Search	RTF	WER
Baseline	0.7s	24.0s	3.2s	28.0	19.7%
Pre	0.1s	24.0s	3.2s	27.4	20.2%
8bit+Pre	0.1s	1.8s	3.2s	5.2	20.2%
EFVR+8bit+Pre	0.1s	1.7s	3.0s	4.9	21.8%
+BBI	0.1s	0.7s	2.5s	3.4	22.1%
+Beam	0.1s	1.3s	2.0s	3.4	21.1%
+BBI+Beam	0.1s	0.6s	1.8s	2.6	23.0%

Table 7: Results in combining different optimizations.

Table 7 shows, that the biggest part of the acceleration is achieved by quantizing the codebooks to 8bit values, with an increase in WER of 2.5% only, mainly due to the coarse calculations of the FFT during preprocessing. Further acceleration can be achieved while increasing the WER more significantly. The last two experiments use a smaller beam size for the search.

6. Conclusions

In this paper we describe the steps taken to port a spontaneous speech recognition system running on desktop computers to PDAs without the need of training new models. In order to achieve an acceptable run-time behavior and memory consumption we implemented various optimization techniques. Those techniques focus on accelerating the preprocessing, search, and acoustic model evaluation by replacing floating point calculations with either approximations or integer arithmetic, by quantizing the models on the fly, and by reducing the amount of data that needs to be decoded by introducing EFVR as a new feature vector reduction technique. On the evaluation data the EFVR did not work as well as expected, because the audio recordings contain reverberation and environmental noise, that makes it more difficult to find static areas in the recording to merge. This issue will be addressed in further experiments, by combining the EFVR with speech detection. By introducing these measures we managed to reduce the execution time of our recognition system on the PDA from 28x real-time to 5x real-time while the word error rate of the recognizer increased by 2.5% relative only. When reducing the execution time down to 2.6x real-time, we get a 17% relative increase of word error rate.

7. Acknowledgements

This work has been funded in part by the European Union under the integrated project TC-Star - Technology and Corpora for Speech to Speech Translation - (IST-2002-FP6-506738, http://www.tc-star.org).

8. References

[1] O. Viikki, "ASR in portable wireless devices," in *ASRU*, 2001.

[2] J. Fritsch and I. Rogina, "The bucket box intersection (BBI) algorithm for fast approximative evaluation of diagonal mixture gaussians," in *ICASSP*, 1996.

[3] M. Novak, "Towards large vocabulary ASR on embedded platforms," in *ICSLP*, 2004.

[4] J. Park and H. Ko, "Compact acoustic model for embedded implementation," in *ICSLP*, 2004.

[5] M. Vasilache, J. Iso-Sipilä, and O. Viikki, "On a practical design of a low complexity speech recognition engine," in *ICASSP*, 2004, pp. 113–116.

[6] B. Zhou, Y. Gao, J. Sorensen, D. Dchelotte, and M. Picheny, "A hand-held speech-to-speech translation system," in *ASRU*, vol. 1, 2003, pp. 664–669.

[7] M. Finke, P. Geutner, H. Hild, T. Kemp, K. Ries, and M. Westphal, "The Karlsruhe-Verbmobil speech recognition engine," in *ICASSP*, vol. 1, 1997, pp. 83–86.

[8] H. Soltau, F. Metze, C. Fügen, and A. Waibel, "A one pass-decoder based on polymorphic linguistic context assignment," in *ASRU*, 2001.

[9] C. Fügen, M. Westphal, M. Schneider, T. Schultz, and A. Waibel, "Lingwear: A mobile tourist information system," in *HLT*, 2001.

[10] "Intel Performance Primitives." [Online]. Available: http://developer.intel.com/software/products/perflib/

[11] Q. Zhu and A. Alwan, "On the use of variable frame rate analysis in speech recognition," in *ICASSP*, 2000.

[12] M. Woszczyna, "Fast speaker independent large vocabulary continuous speech recognition," Ph.D. dissertation, University of Karlsruhe, Germany, 1998.