EURASIP Journal on Applied Signal Processing 2005:17, 2816–2827
© 2005 Hindawi Publishing Corporation

Accuracy of MFCC-Based Speaker Recognition in Series 60 Device

Juhani Saastamoinen

Department of Computer Science, University of Joensuu, P.O. Box 111, 80101 Joensuu, Finland
Email: juhani@cs.joensuu.fi

Evgeny Karpov

Department of Computer Science, University of Joensuu, P.O. Box 111, 80101 Joensuu, Finland
Email: ekarpov@cs.joensuu.fi

Ville Hautamäki

Department of Computer Science, University of Joensuu, P.O. Box 111, 80101 Joensuu, Finland
Email: villeh@cs.joensuu.fi

Pasi Fränti

Department of Computer Science, University of Joensuu, P.O. Box 111, 80101 Joensuu, Finland
Email: franti@cs.joensuu.fi

Received 1 October 2004; Revised 14 June 2005; Recommended for Publication by Markus Rupp

A fixed point implementation of speaker recognition based on MFCC signal processing is considered. We analyze the numerical error of the MFCC and its effect on the recognition accuracy. Techniques to reduce the information loss in a converted fixed point implementation are introduced. We increase the signal processing accuracy by adjusting the ratio of presentation accuracy of the operators and the signal. The signal processing error is found out to be more important to the speaker recognition accuracy than the error in the classification algorithm. The results are verified by applying the alternative technique to speech data. We also discuss the specific programming requirements set up by the Symbian and Series 60.

Keywords and phrases: speaker identification, fixed point arithmetic, round-off error, MFCC, FFT, Symbian.

1. INTRODUCTION

The speech research and application development deal with three main problems: speech synthesis, speech recognition, and speaker recognition. We are working in a speech technology project, where one of the main goals is to integrate automatic speaker recognition technique into Series 60 mobile phones.

In speaker recognition, we have a recorded speech sample and we try to determine to whom the voice belongs. This study involves *closed-set speaker identification*, where an unknown sample is compared to previously trained voice models in a speaker database.

The speaker identification is a speech classification problem. Based on the training material, we create speaker-specific voice models, which divide the feature space into distinct classes. Unknown speech is transformed to a sequence of features, which are scored against voice models. That speaker is identified and his model has the best overall match with the input features. There are many ways to choose the used features and how they are used. Our research team has studied, for example, how the feature design [1], or the concurrent use of multiple features [2], affects the recognition accuracy.

Our speaker identification method is a generic automatic learning classification with *mel-frequency cepstral coefficient* (MFCC) features. The classification algorithm that we use in this study is a common unsupervised vector quantizer. We have ported the identification system to a Series 60 Symbian mobile phone. In this study, we introduce the Series 60 platform and the ported system. In particular, we focus on the numerical analysis of the signal processing algorithms which had to be converted to fixed point arithmetic.

When the system is run on a mobile phone, the two biggest problems are sound quality and the numerical error in FFT. Straightforward fixed point implementation reduces accuracy dramatically. We obtain good recognition accuracy by decreasing the numerical error in critical parts of

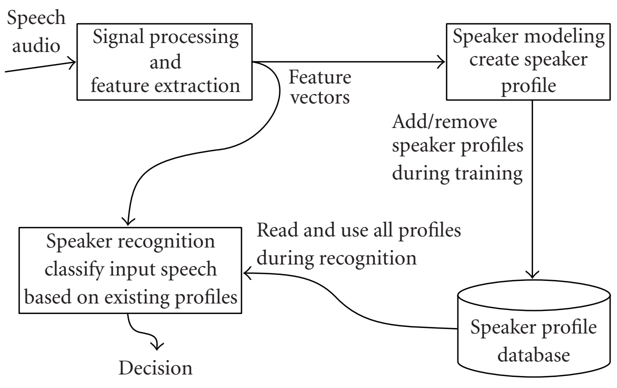

Speech audio
Signal processing and feature extraction
Feature vectors
Speaker modeling create speaker profile
Add/remove speaker profiles during training
Speaker recognition classify input speech based on existing profiles
Read and use all profiles during recognition
Speaker profile database
Decision

Figure 1: Closed-set speaker identification system.

our proposed system. For example, with 100 TIMIT speakers, the recognition rates for different implementations are 100% (floating point), 9.7% (straightforward fixed point), and 95.8% (proposed system).

2. SPEAKER IDENTIFICATION SYSTEM

We consider a speaker identification system with separate modules for speech signal processing, training and classification, and speaker database (Figure 1). The system operates in training mode or recognition mode. The two different chains of arrows starting from the signal processing module describe the data flow (Figure 1).

The system input in *training mode* is a collection of speech samples from N different speakers. A signal processing model is applied to produce a set of feature vectors for each speaker separately. Then a mathematical model is fitted to the feature vector set. We use the *vector quantization* (VQ) model to represent the statistical distribution of the features of each speaker. Each feature vector set is replaced by a *codebook*, which is a smaller set of *code vectors* with fixed size. Codebooks are stored in the speaker database to represent the speakers. A common goal of the codebook design is to minimize the quantization distortion of the training data, that is, we look for code vectors which minimize the distortion, when training vectors are replaced by their nearest neighbors in the codebook. We use the *generalized Lloyd algorithm* (GLA) [3] to generate the codebook.

In the *recognition mode*, the input speech sample is processed by the same signal processing methods as in the training. The features are quantized using each codebook in the database. The speaker whose codebook gives the least distortion is identified. If needed, the system lists the smallest distortions and corresponding speakers.

The signal processing module computes MFCC features (Figure 2). They are commonly used in speech recognition [4]. The speech is divided into overlapping frames. Within a frame, the signal is preemphasized and multiplied by a windowing function before computing the Fourier spectrum. A mel-filter bank is applied to the magnitude spectrum, and logarithm of the filter bank output is finally cosine

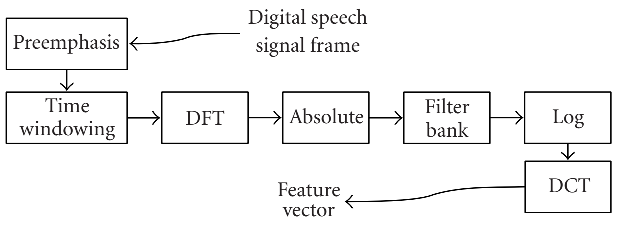


Figure 2: MFCC signal processing steps.

transformed. The first coefficient of the cosine transform is omitted as it depends on the signal energy. We want to discard absolute energy information which depends, for example, on the distance to the microphone, or on the voicing degree. If we kept the first coefficient, then the vectors with high overall intensity, for example vowels, would dominate the distance computations. Only part of the cosine-transform output coefficients are used as the feature vector.

3. SYMBIAN ENVIRONMENT

The small size of mobile phones is demanding for manufacturers. A hardware design must be cheap to manufacture, fit in small space, and have low power consumption.

The company Advanced RISC Machines (ARM) has developed the most commonly used mobile phone processors. They are fully 32-bit RISC processors with a 4 GB address range. A three-stage pipeline is used, which allows execution of one instruction per every cycle [5].

One drawback of the ARM processors is that they have no floating point support because of its complexity and hard power consumption.

3.1. Symbian OS and Series 60

In order to reduce phone development costs, the leading manufacturers started developing an industry standard operating system for advanced, data-enabled mobile phones [6]. The company Symbian was formed in 1998 by the leaders of the mobile industry: Nokia, Ericsson, Panasonic, Motorola, Psion, Siemens, and Sony Ericsson. They developed the *Symbian OS* operating system [7], which evolved from the EPOC operating system developed by Psion. It has a modular microkernel-based architecture [6], whose core consists of *base* (microkernel and device drivers), *middleware* (system servers), and *communications* (telephony, messaging, etc.) [6].

The Symbian OS is fully multitasking. It supports simultaneously running processes, threads, separate address space, and preemptive scheduling [7]. However, because of the limited hardware performance, it is recommended that most applications use the built-in *active objects* framework for non-preemptive multitasking [6]. Symbian OS also has a file system. Files are stored in the ROM or RAM of the phone, or on removable flash disks. Dynamically linked libraries are also supported [6].

The Symbian OS can be combined with different user interface (UI) platforms. A UI platform is a set of programmable UI controls, which all have similar style. There are three UI platforms known to the authors: UIQ (developed by Sony Ericsson), Series 60, and Series 80 (both developed by Nokia).

3.2. *Programming for Symbian OS*

Programs for Symbian OS can be written in Java and C++. The Java API and execution speed are limited, so C++ is used for computationally intensive programs. A lot of APIs are available for the C++ programmer, and there is also a limited ANSI C standard library [6, 7].

The main difference to conventional PC programming in Symbian OS is that the program must always be ready for exceptional situations. Device can easily use all available memory or program can be interrupted by incoming phone call, which has higher priority. Programs must also be as small and efficient as possible to not overwhelm the limited hardware resources. Robustness is also important, because mobile phones are supposed to work without restart for months or even more [7].

The used algorithms must be selected carefully, numerically stable low-time complexity methods are preferred. There is no hardware floating point support. There exists a software implementation of double-precision floating point arithmetic but it should be used rarely because of its complexity and higher power consumption. Also there is a 64-bit integer type available for the programmer, but it is a software implementation where the data is stored in a pair of 32-bit integers. The ported algorithms must be efficient, therefore we use fixed point arithmetic and only native data types, that is, integers whose basic operations are directly supported by the processor.

3.3. *C++ restrictions*

The Symbian OS restricts the use of C++ features. There is no standard exception handling. Symbian designers implemented their own mechanism for it, mainly because the GCC compiler used in target builds did not support it at the time [7]. Consequently, a C++ class constructor cannot create other objects. It might cause an exception, and Symbian has no way to handle exceptions thrown from a constructor. Therefore, a *two-phase construction* must be used, where object creation and initialization are separated [7]. As another consequence, the memory stack is not unrolled after an exception, so the programmer must use a *cleanup stack* framework, which unrolls the stack automatically after an exception [7]. That is why all objects allocated from the heap must be derived from a common base class (*CBase*), added to the stack immediately after allocation, and removed only just before deletion [7]. Here, conventional C++ compiler duties have become manual programming tasks.

Efficiency requirements dictate another important aspect of Symbian programming. Applications or DLLs can be executed from the ROM without copying them first to the RAM. It creates another programming limitation: an application stored in a DLL has no modifiable segment and cannot use static data [7]. However, Symbian provides a *thread-local storage* mechanism for static data [7]. Basically, any application interacting with the user is stored in a DLL and loaded by the framework, when a user selects to execute the particular program [7].

We implemented most of the computational algorithms in the ANSI C language and used the POSIX standard where applicable. The reasons were good portability, an existing prototype written in C, and the ANSI/POSIX support of the system. The Symbian OS has a standard C library, so programs are easy to port to it. The main limitation is that static data, that is, global variables cannot be used. Also file handling is restricted: fopen and other file-processing functions may not work as expected in multithreaded programs. The developers are encouraged to use the provided *file server* mechanisms instead.

4. NUMERICAL ANALYSIS OF MFCC AND VQ IN FIXED POINT ARITHMETIC

During the recognition, the speaker information carried by the signal propagates through the signal processing (Figure 2) and classification to a speaker identity decision. The mappings involved in the MFCC process are smooth and numerically stable. In fact, the MFCC steps are one-to-one mappings, except those where the mapping is to a lower-dimensional vector space, for example, computing magnitudes of the elements of the complex Fourier spectrum.

The MFCC algorithm consists of evaluations of different vector mappings f between vector spaces, denote such evaluation by $f(x)$. A computer implementation evaluates values $\hat{f}(\hat{x})$, where $\hat{x}$ is an approximation of x represented in a finite-accuracy number system, and the computer implementation $\hat{f}$ tries to capture the behavior of f. When implementing $\hat{f}$, we aim at minimizing the *relative error* of the values $\hat{f}(\hat{x})$,

$$\epsilon = \frac{\|f(\hat{x}) - \hat{f}(\hat{x})\|}{\|f(\hat{x})\|}, \tag{1}$$

instead of their *absolute error* $\|f(\hat{x}) - \hat{f}(\hat{x})\|$. The motivation for using relative error is that all elements of all vectors, during all MFCC stages, may carry information that is crucial to the final identification decision. The importance of each element to the final speaker discrimination is independent of the numerical scale of the data in the subspace corresponding to the element. The input $\hat{x}$ is usually the output of the previous step.

Most MFCC processing steps are linear mappings and the two nonlinear ones behave well. The real-valued magnitudes of complex Fourier spectrum elements are computed before applying the filter bank, and later filter bank output logarithms are used in order to bring the numerical scale of the outputs closer to linear relation with human perception scale [4]. However, in fixed point arithmetic, not even computing the value of a well-behaving mapping is always straightforward.

We consider a system capable of fixed point arithmetic with signed integers stored in at most 32 bits. The input consists of sampled signal amplitudes represented as signed 16-bit integers. In many parts, we use different integer value interpretation, a scaling integer $I > 1$ represents 1 in the normal algorithm. Often we must also divide input, output, or intermediate result to ensure that it fits in a 32-bit integer. We now analyze the system.

4.1. *Preemphasis*

Many speech processing systems apply a *preemphasis* filter to the signal before further processing. The difference formula $y_t = x_t - \alpha x_{t-1}$ is applied to the signal x_t, our choice is a common $\alpha = 0.97$. The filter produces output signal y_t where higher frequencies are emphasized and lowest frequencies are damped.

4.2. *Signal windowing*

Numerically speaking, there is nothing special in the signal windowing. A signal frame is pointwise multiplied with a *window function*. The motivation is to avoid artifacts in the Fourier spectrum that are likely to appear because of the signal periodicity assumption in the Fourier analysis theory. Therefore, the window function has usually a taper-like shape, such that the multiplied signal amplitude is near-original in the middle of the frame but gradually forced to zero near the endpoints. Getting the multiplied signal gradually to zero requires using enough bits to represent the window function values. For example, in the extreme case of using only one bit, the transition from original signal to a zeroed multiplied signal is sudden, not gradual. We use 15 bits in the experiments.

4.3. *Fourier spectrum*

The frequency spectrum is computed as the N-point *discrete Fourier transform* (DFT) $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$,

$$\mathcal{F}(x) = \sum_{k=0}^{N-1} e^{-2\pi i \omega k/N} x_k, \quad \omega = 0, \ldots, N-1. \tag{2}$$

As a linear map, $\mathcal{F}$ has a corresponding matrix $F \in \mathbb{C}^{N\times N}$, and $\mathcal{F}(x)$ can be computed as the matrix-vector product Fx using $\mathcal{O}(N^2)$ operations. The *radix-2 fast Fourier transform* (FFT) [8] utilizes the structure of F and computes Fx in $\mathcal{O}(N \log N)$ operations for $N = 2^m$, $m > 0$. The FFT executes the computations in $\log_2 N$ layers of $N/2$ *butterflies*,

$$\begin{aligned} f_k^{l+1} &= f_k^l + W_k^l f_{k+T}^l, \\ f_{k+T}^{l+1} &= f_k^l - W_k^l f_{k+T}^l. \end{aligned} \tag{3}$$

Superscripts denote the layer and the constants $W_k^l \in \mathbb{C}$ are called *twiddle factors*. The first layer input is the signal $f_k^0 = x_k$, $k = 0, \ldots, N-1$. The offset constant T varies between layers, the value depends on whether the FFT element reordering [8] is done for input or output.

4.3.1. *Existing fixed point implementations*

The FFT efficiency is based on the layer structure. However, fixed point implementations introduce significant error. The round-off errors accumulate in the repeatedly applied butterfly layers.

Our reference FFT is C code generated by the *fftgen* software [9]. he generated code computes the squared FFT magnitude spectrum (Section 4.4) of a signal in fixed point arithmetic. The butterfly layers and the element reordering are all merged in few subroutines, with all loops unrolled. It uses 16-bit integer representation for the input signal, intermediate results between layers, and the automatically computed power spectrum output. Multiplication results in (3) are 32-bit integers, but stored in 16-bit integers after shifting 16 bits to the right in order to keep the next layer input in proper range. Overflowing 16-bit result of addition and subtraction in (3) is avoided by shifting their inputs 1 bit to the right. The truncations increase error and introduce information loss.

We employed the generated FFT code in the fixed point MFCC implementation and compared it to the floating point counterpart. The MFCC outputs computed from identical inputs with the two implementations did not correlate much. It might originate from the accumulation of errors in the MFCC process. However, detailed analysis showed that the greatest error source is FFT (DFT in Figure 2). We also verified that the error does not originate from the final truncation of the power spectrum elements to 16 bits, but from the FFT algorithm itself. In order to verify it, we tuned the generated code to output the complex FFT spectrum instead of the power spectrum.

Many techniques have been developed for decreasing the error in fixed point implementations. A comprehensive analysis of various possibilities was presented by Tran-Thong and Liu in [10]. There are also many improvements tailored for specific microprocessors and applications. For example, Sayar and Kabal consider an implementation for a TMS320 digital signal processor [11].

4.3.2. *Proposed FFT*

Our approach is more general than the implementations listed above. We consider any processor capable of integer arithmetic with signed 32-bit integers. We use an existing radix-2 complex FFT implementation [12] as the starting point. First, we change the data types, additions, and multiplications similar to the *fftgen*-generated code.

The generated code uses 16 bits for the real and imaginary parts of the layer inputs, and for the real-valued trigonometric constants arising from (3), after the Euler formula $e^{i\varphi} = \cos\varphi + i\sin\varphi$ has been applied in (2). We changed the data type used for the intermediate results in (3) from 16-bit to 32-bit integers. But this alone does not really help to preserve more than 16 bits of the intermediate results if operator constants still use 16 bits. The multiplication result must fit in 32 bits. Our solution is to reduce the DFT operator representation accuracy in order to increase the amount of preserved signal information.

Consider the DFT in the operator form $f = Fx$, and our implementation $\hat{f} = \hat{F}\hat{x}$. The approximation error $f - \hat{f}$ consists of the input error $x - \hat{x}$ and the implementation error. Since F and $\hat{F}$ are linear, the implementation error is $F - \hat{F}$. This is not exactly true, as we have a limited-accuracy numeric implementation, which is only linear up to the numeric accuracy.

Now repeat the same analysis but consider a linear butterfly layer in the FFT algorithm $g = Gy$, and its implementation $\hat{g} = \hat{G}\hat{y}$. The inputs $\hat{y}$ carry information about accurate values y, that is, information about the signal x. In the butterfly (3), each multiplication of the layer input element $f_k^l \in \mathbb{C}$ with the operator constant $W_k^l \in \mathbb{C}$ expands to two additions and four multiplications of real values. If we use more than 16 bits for the real values that correspond to f_k^l, then we must use less bits for the real values that correspond to the operator constant W_k^l, in order to represent the real values that correspond to the multiplication result with 32 bits.

We allow increase in the relative error of the layer operator $\|G - \hat{G}\|/\|G\|$, meanwhile the relative input error $\|y - \hat{y}\|/\|y\|$ is decreased so that more information about y fits into $\hat{y}$, and more is preserved in the multiplication result. Consequently, more information about y propagates to the next layer input $\hat{g}$ in all layers, therefore less information is lost in the whole FFT. We increase the FFT operator error $\|F - \hat{F}\|/\|F\|$ little but preserve more information about x. Consequently, the relative error $\|F\hat{x} - \hat{F}\hat{x}\|/\|F\hat{x}\|$ decreases. This is the main idea and it can also be applied to other algorithms implemented in fixed point arithmetic. Here the norm of a linear operator A is defined as $\|A\| = \max_{\|x\|=1} \|Ax\|/\|x\|$, and the difference $A - B$ of the operators A and B is defined by $(A - B)x = Ax - Bx$, for all x.

4.3.3. Bit allocation

The twiddle factors of an N-point DFT are constructed from the values $\pm \sin \pi k/N$ and $\pm \cos \pi k/N$, $k = 0, \ldots, N/2-1$. Before deciding how the bits are allocated for the signal and the operator, we look at the relative trigonometric value round-off errors for different FFT sizes N and bit allocations $B > 0$. For each B, we look for a scaling integer c which gives small value of the maximum error

$$E(c, N) = \max_{k=0,\ldots,N/2-1} \left| \frac{s_k - \hat{s}_k}{s_k} \right|, \tag{4}$$

where $s_k = c \sin \pi k/N$ and $\hat{s}_k$ denotes s_k rounded to the nearest integer. It is enough to consider only the positive sines, since the cosine values are in the same set. For $N = 256$, 512, 1024, 2048, and 4096, there are several peaks downwards in the graph of $E(c, N)$ as a function of c. They are good choices of c, even if they do not minimize $E(c, N)$. Table 1 shows the pairs of good values of c and $E(c, N)$ for different N. The bit allocation B is defined as the number of bits needed to store c.

We decided to limit the FFT size to $N \leq 1024$ and not minimize $E(c, N)$ for each N separately. For all $N = 256$, 512, and 1024, the value $c = 980$ is the best choice with $B = 10$.

Table 1: Pairs of values c and $E(c, N)$ for different FFT sizes N, the pairs are selected where E is small; the values of the function E have been multiplied by 10^3.

N									
256		512		1024		2048		4096	
c	E	c	E	c	E	c	E	c	E
82	16.6	164	9.7	327	6.4	654	3.7	1306	2.4
164	9.5	327	6.4	328	6.3	1306	2.4	1307	2.5
246	7.1	328	6.3	653	4.1	1307	2.5	2610	1.6
327	5.9	490	4.8	654	3.7	1958	1.9	2611	1.5
409	5.3	491	4.4	979	3.1	1959	1.8	3915	1.1
491	4.2	653	4.0	980	2.9	2610	1.6	3916	1.2
572	4.0	654	3.6	—	—	2611	1.5	—	—
654	3.3	815	3.8	—	—	3262	1.3	—	—
735	3.5	816	3.6	—	—	3263	1.3	—	—
736	3.5	817	3.1	—	—	3915	1.1	—	—
817	3.1	979	3.1	—	—	3916	1.2	—	—
899	2.9	980	2.7	—	—	—	—	—	—
980	2.7	981	3.2	—	—	—	—	—	—

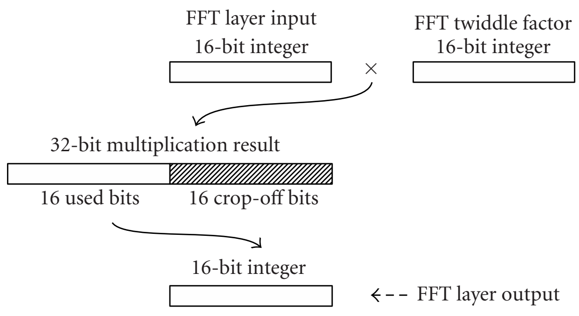


Figure 3: Multiplication of a 16-bit integer, followed by a bit shift in a layer of the *fftgen* FFT.

That leaves 22 bits for the signal information. Thus, we replace the signal/operator bit allocation 16/16 with 22/10. The choice with one c for all N and $B = 10$ is good enough for us, as we mostly use $N = 256$. The diagrams in Figures 3-4 illustrate the bit allocation in integer multiplications and truncations in a layer of the *fftgen* FFT and the proposed FFT.

4.3.4. Evaluation of the accuracy

We compare the proposed fixed point solution to the *fftgen* generated FFT code. In our floating point MFCC implementation, we compute FFT using the *Fastest Fourier Transform in the West* (FFTW) C library [13]. The FFTW relative error is very small. We refer to FFTW output as the accurate solution when comparing the fixed point algorithms. We use a TIMIT speech segment as the input signal, resampled at 8 kHz (Figure 5).

Figure 6 shows two scatter plots of pairs of logarithms of absolute values of the *fftgen* FFT and the floating point FFT. If there were no errors, all dots would reside on the diagonal. Figure 7 shows the same for the proposed FFT.

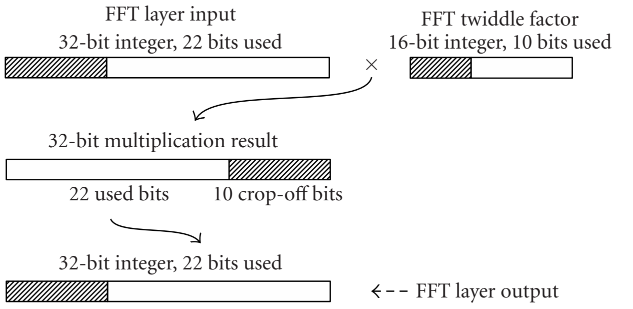

FFT layer input
32-bit integer, 22 bits used
FFT twiddle factor
16-bit integer, 10 bits used
32-bit multiplication result
22 used bits
10 crop-off bits
32-bit integer, 22 bits used
FFT layer output

Figure 4: Multiplication of a 22-bit integer, followed by a bit shift in a layer of the proposed FFT.

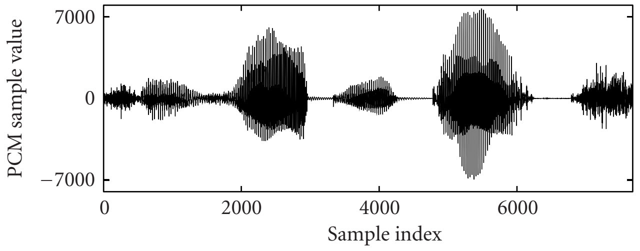


Figure 5: A speech sample from the TIMIT corpus.

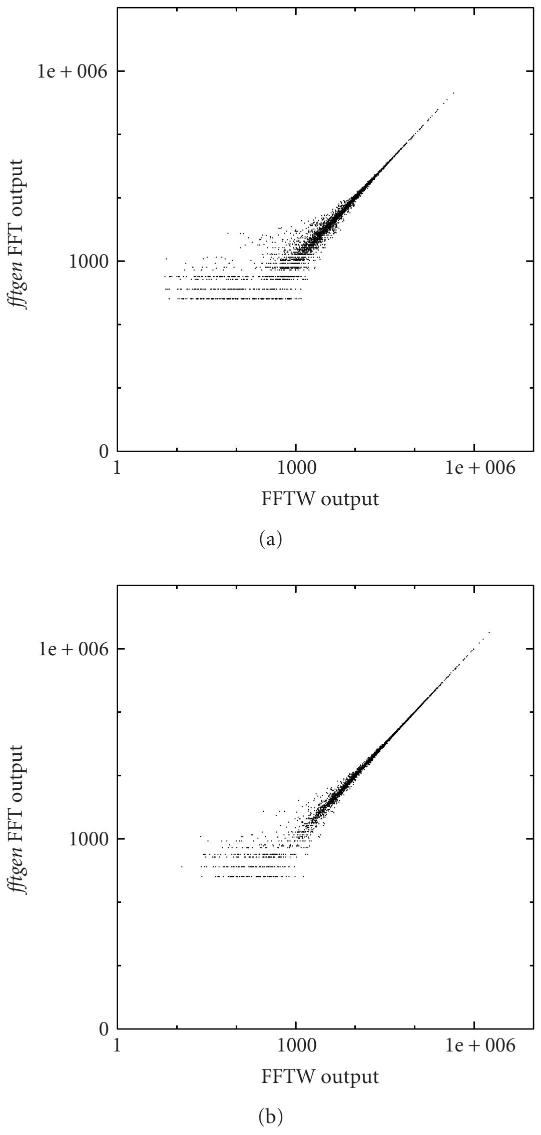


Figure 6: Scatter plot of *fftgen* FFT output against FFTW output for the TIMIT signal x (a) and $4x$ (b), scales are logarithmic.

Comparison of the FFT magnitude scatter plots in Figures 6-7 shows that in fixed point arithmetic, we may decrease the error by using the integer scale more efficiently. The proposed FFT is accurate without scaling also. Also note that the proposed FFT has an increased range of accurate values, that is, the distance along the diagonal from the rightmost observation to the place where the observations start to deviate from the diagonal is much longer for the proposed FFT than the *fftgen* FFT.

The statistical distribution of the relative error of the fixed point FFT elements is very skew, but the logarithmic error behaves nearly like a normal distribution. The histograms in Figures 8-9 illustrate the distribution of $\log_{10}\epsilon = \log_{10}(|f_k - \widehat{f}_k|/|f_k|)$, which is the same as the signal-to-noise ratio in decibels divided by -10. Here f_k and $\widehat{f}_k$ are elements of the correct FFT and the fixed point FFT, correspondingly. The *fftgen* FFT error histogram is shown in Figure 8, whereas Figure 9 shows the error of the proposed FFT. For statistical analysis, it makes sense to consider the logarithmic errors. Their interpretation is easier because of the original skew error distribution.

Table 2 summarizes the logarithmic error statistics. The numbers -0.775 and -2.118, for example, suggest that for the test signal, the proposed method has less than 1% error per element on average, whereas the same value is more than 10% for the *fftgen*. In terms of signal-to-noise ratio, the advantage of our method is 13.43 dB for the original signal, and also a significant 10.32 dB for the more optimally scaled signal. The statistics state clearly that the proposed FFT is a lot more accurate.

Until now, we have only described the advantages of the proposed FFT but it also has little drawbacks. The scaling of the numbers between the FFT layers requires more operations than the *fftgen* implementation.

The *fftgen* input signal is represented by 16-bit integers. In our case, we wanted to replace the *fftgen* program module with minimal effect to the other parts, and therefore, we need to scale the input and output. We input 16-bit integers also to the proposed algorithm. They are first scaled up to use 22 bits, so that minimal amount of signal information will be lost when the 32-bit multiplication results are truncated back to 22-bit representation for the next FFT layer. There are other multiplications and bit shifts involved besides the scaling related to the multiplications in (3). In contrast to floating point FFT algorithms, the twiddle factors are rep-

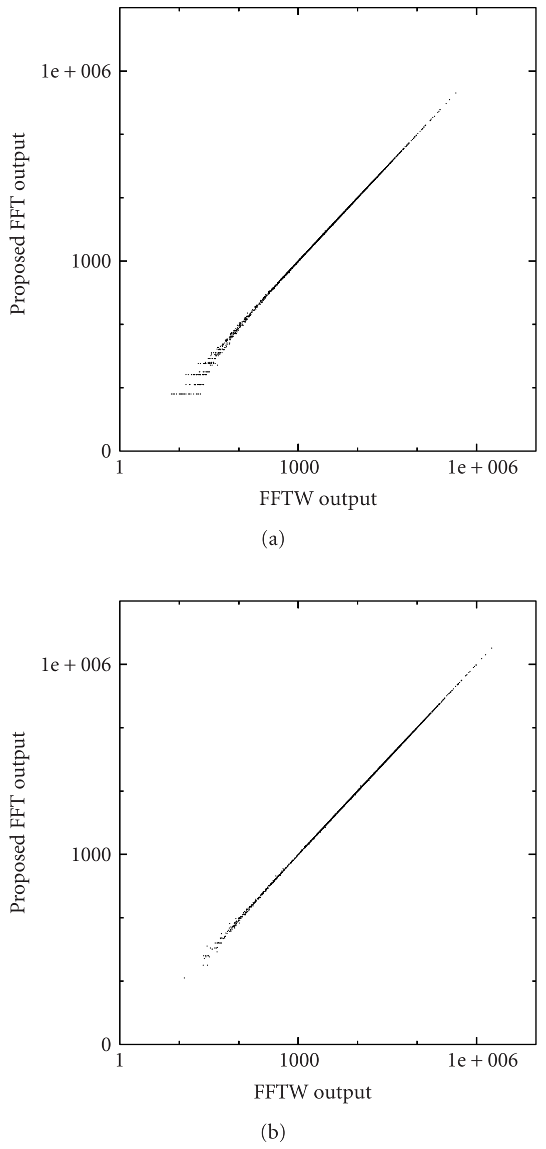


Figure 7: Scatter plot of proposed FFT output against FFTW output for the TIMIT signal x (a) and $4x$ (b), scales are logarithmic.

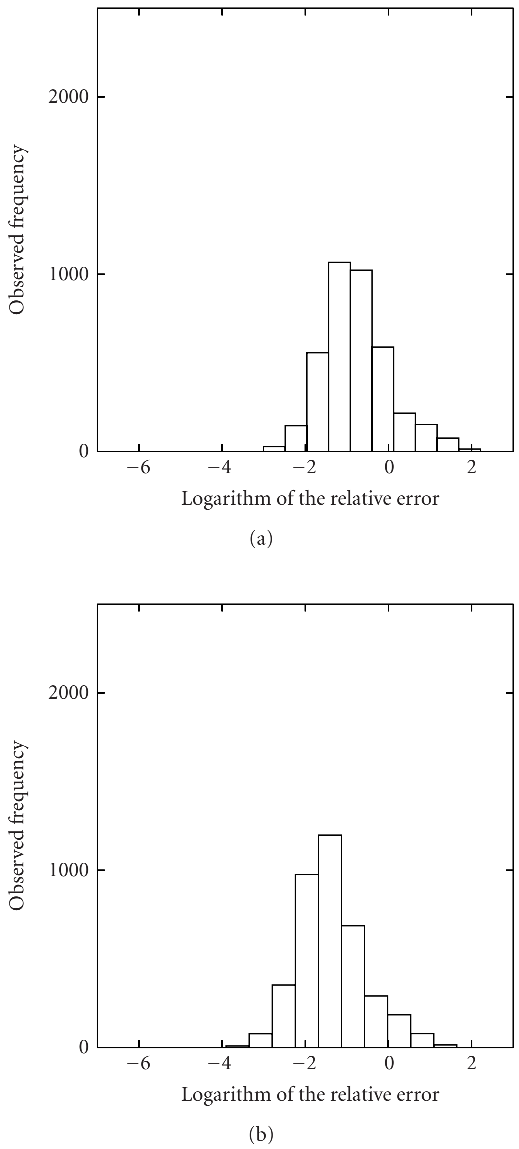


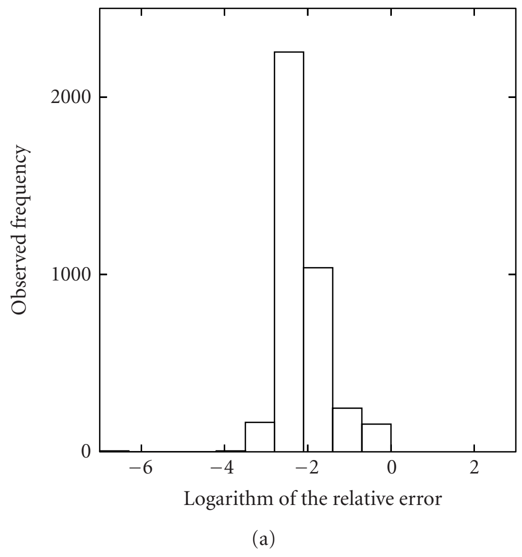
Figure 8: Histogram of logarithmic relative error values for the *fftgen* FFT with input signals x (a) and $4x$ (b), the error increases to the right.

resented using integers. Therefore, before the addition and subtraction in a butterfly (3), we must scale up f_k^l before adding it to the result of the complex multiplication $W_k^l f_k^l$.

In other parts of the MFCC algorithm, the more accurate 22-bit representation of the proposed FFT output could be utilized instead of scaling down to 16 bits. However, based on our error analysis and the statistic in Table 2, the 16-bit output of *fftgen* FFT is really not accurate up to 16 bits, and neither is the proposed FFT. On average, there are 3–5 most significant bits correct in the *fftgen* FFT output and 7-8 most significant bits correct in the proposed FFT. Thus, there is no need to use more than 16 bits for the real part and 16 bits for the imaginary part of the FFT output elements.

4.4. Magnitude spectrum

The Fourier spectrum is $\{f_k \in \mathbb{C};\ k = 0,\dots,N/2-1\}$, the power spectrum is $\{|f_k|^2 \in \mathbb{R}\}$, and the magnitude spectrum is $\{|f_k| \in \mathbb{R}\}$. The squaring has no significant effect in the recognition rate for the floating point implementation. In fixed point arithmetic, the usage of the number range is not uniform for the power spectrum. The distribution of values $|f_k|^2$ is dense for small $|f_k|$ and sparse for large $|f_k|$. The values $|f_k|$ are more uniformly distributed when the real and imaginary parts of f_k take all possible values within the integer range. We use the magnitude spectrum approximated as follows.



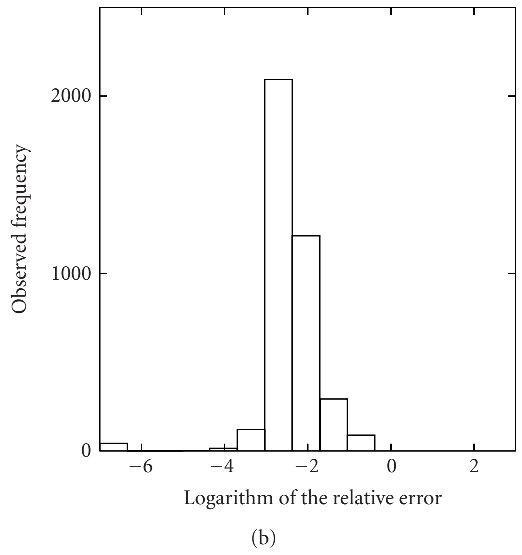


Figure 9: Histogram of logarithmic relative error values for the proposed FFT with input signals x (a) and $4x$ (b), the error increases to the right.

Without loss of generality, assume that $|a| \geq |b|$ and $|a| > 0$ for $f_k = a + \mathrm{i}\, b$. We may write

$$|f_k| = \sqrt{a^2 + b^2} = |a|\sqrt{1 + \left(\frac{b}{a}\right)^2}, \tag{5}$$

where $1 + (b/a)^2 \in [1, 2]$ always. By introducing a parameter $t = |b/a| \in [0, 1]$, we can approximate $|f_k|$ with

$$|f_k| = |a|\sqrt{1 + t^2} \approx |a| P_n(t), \tag{6}$$

Table 2: Average (AVG) and standard deviation (SD) of the base-10 logarithm of the relative error, and signal-to-noise ratio (SNR) in decibels for two FFT implementations, applied to the same signal on two different scales.

Used FFT	Input	AVG	SD	SNR (dB)
fftgen	x	−0.775	0.797	7.75
fftgen	$4x$	−1.374	0.797	13.74
Proposed	x	−2.118	0.590	21.18
Proposed	$4x$	−2.406	0.687	24.06

where $P_n : [0, 1] \to [1, \sqrt{2}]$ is a polynomial of order $n \geq 1$ with the boundary conditions

$$P_n(0) = 1, \qquad P_n(1) = \sqrt{2}. \tag{7}$$

In order to satisfy boundary conditions, we actually find the orthogonal projection of $\sqrt{1 + t^2} - (1 + (\sqrt{2} - 1)t)$ into the function space spanned by the set of functions $S = \{t - t^2, t - t^3, t - t^4, t - t^5\}$, that is, fit a least-squares polynomial. Our approximation is

$$\begin{aligned}\sqrt{1 + t^2} \approx 1 &+ \left(\sqrt{2} - 1\right)t \\ &- 0.505404(t - t^2) + 0.017075(t - t^3) \\ &+ 0.116815(t - t^4) - 0.043182(t - t^5),\end{aligned} \tag{8}$$

with the maximum relative error 1.30×10^{-5}.

The motivation for our boundary conditions (7) is that a least-squares polynomial often has a relatively large maximal error in the endpoints of the approximation interval. Here the polynomial is used for evaluation of MFCCs, and accurate approximation is needed regardless of t, the ratio of real and imaginary parts of f_k.

4.4.1. Complex magnitude with fixed point numbers

There probably are numerically better choices for the basis besides S. However, it is straightforward to evaluate t^{p+1} from t^p and t in our scaled integer arithmetic. Moreover, the basis S meets the boundary conditions. Note also that $0 \leq t, t^p, t - t^p \leq 1$ for $t \in [0, 1]$ so that all intermediate results in the polynomial evaluation are always within our number range.

In the fixed point implementation, we choose an integer scaling factor $d \in [1, 2^{15})$ to represent 1, because the multiplication results must always fit in 32 bits. The value t and coefficients of $1, t, \ldots, t - t^5$, are evaluated to rescaled integers before the polynomial evaluation. We chose $d = 20263$ because it minimizes the average relative round-off error in the scaled polynomial coefficients. The fixed point arithmetic square root approximation is

$$\begin{aligned}20263\sqrt{1 + t^2} \approx 20263 &+ 8393t \\ &- 10241(t - t^2) + 346(t - t^3) \\ &+ 2367(t - t^4) - 875(t - t^5),\end{aligned} \tag{9}$$

where the original $t \in [0,1]$ is multiplied with d and truncated to integer before the evaluation. During the evaluation, all multiplication inputs are within $[0,d]$ and multiplication results are always divided with d. The maximum relative error is 1.855×10^{-5} for $t = 0.9427$.

4.5. Filter bank

Applying a linear filter in the frequency domain is technically similar to the signal windowing in the time domain, a spectrum is pointwise multiplied with a frequency response. Each filter output is a weighted sum of the magnitude spectrum or power spectrum values. Applying a *linear filter bank* (FB) means applying several filters, and it is the same as computing a matrix-vector product where matrix rows consist of the filter frequency responses.

Numerically, the fixed point implementation is not complicated, we just need enough bits to represent the frequency response values. By our standard, we are using enough bits if a graphical visualization of the filter bank filters realizes our visual idea of the desired filter shape. We use 7 bits in the experiments, Technically, the purpose of filter bank is to measure energies in subbands of the frequency domain of the signal, with possible overlap between adjacent subbands. It is commonplace to define the filter bank so that

(i) for all input spectrum elements, the sum of weights over all filters is the same;
(ii) the width of the filters is defined by a monotonic *frequency-warping function* [4], such that
   (a) in the warped frequency domain, all filters have equal spacing, width, and overlap;
   (b) in the warped frequency domain, all filters have the same shape, for example, triangular or bell.

The shape of filters is not important for speaker recognition but the choice of the frequency warping function has significant effect on the recognition accuracy [1]. Our choice is the commonly used, although not optimal mel-frequency warped FB with triangular filter shape.

One could argue that the FB smoothing effect compensates the numeric error of the FFT and magnitude computations. However, discrimination information will be lost both in the numeric round-off and in the smoothing.

4.6. Logarithm

The nonnegative FB outputs are transformed into logarithmic scale during the MFCC processing. Several methods for evaluation of $\log_2$ have been introduced in [14] and there is a thorough error analysis in [15].

We use a modification of the method in [14], which uses a lookup table and linear interpolation. Consider an integer $n > 0$ whose bit representation is

$$n = 0000\underbrace{1\,b_m \dots b_1}_{m+1 \text{ bits}}. \tag{10}$$

The integer part of $\log_2 n$ is m. The decimal part is encoded in the bits $b_m, \dots, b_1$. We use the 8 most significant bits $b_m, \dots, b_{m-7}$ as an index to a lookup table consisting of the values $\log_2(1+j/256)$, $j = 0, \dots, 256$. The next 7 bits form the interpolation coefficients between two consecutive lookup table values. The maximum relative error 4.65×10^{-6} occurs for $n = 272063$, where the correct value is $\log_2 272063 = 18.053581$ and our approximation is 18.053497.

4.7. Discrete cosine transformation

Discrete cosine transformation (DCT) is a linear invertible mapping, which is most efficiently computed using the FFT and some additional processing. In our application, we transform 25–50-dimensional vectors to 10–15-dimensional vectors and use only part of the DCT output, so we compute it with the direct formula without FFT. We utilize the most common DCT form called DCT-II [16],

$$\mu_j = \sum_{k=0}^{N_{\text{FB}}-1} l_k \cos\left(\frac{\pi}{N_{\text{FB}}}\left(k + \frac{1}{2}\right)j\right), \tag{11}$$

where $j = 0, \dots, N_{\text{MFCC}} - 1$, and N_{MFCC} is the number of the MFCC coefficients needed. The input l_k consists of the FB outputs or their logarithms, $k = 0, \dots, N_{\text{FB}} - 1$. Usually, μ_0 is ignored as it only depends on the signal energy. The DCT-II form is orthogonal if μ_0 is multiplied by $1/\sqrt{2}$ and all coefficients are output [16]. DCT is applied to FB outputs in speech applications for many reasons. Here the rescaling and decorrelating of the FB outputs improves the clustering and the VQ classification.

We did not carefully analyze the DCT error in the fixed point implementation. The reason is that we found out that the FFT and the logarithm were the MFCC accuracy bottlenecks. We simply assign the scaling factor 32767 for cosine values and truncate 16 bits from the 32-bit input values. We might gain accuracy by similar analysis that we did with the FFT but not much. In contrast to the FFT, the direct DCT computation has only one layer.

4.8. Model creation and recognition

The GLA algorithm [3] constructs a codebook $\{c_k\}$ that aims at minimizing the MSE distortion

$$\text{MSE}(X, C) = \sum_{j=1}^{N} \min_{1 \le k \le K} \|x_j - c_k\|^2 \tag{12}$$

of the training data $\{x_j\}$. This is our speaker modeling. The algorithm is simple and does not really involve parts that require floating point arithmetic. The differences between floating point and fixed point implementations are due to limited accuracy in the relative MSE change near the convergence, and most importantly, the accumulating round-off error during the iteration. The round-off error in the MSE distance computations is also different in fixed point arithmetic.

Table 3: Recognition rate average and standard deviation for five different implementations of the MFCC-based speaker recognition system, varying number of speakers taken from the TIMIT corpus and number of repeated cycles of training, and recognition.

	Number of speakers	16	25	100	16	25	100
	Number of repeats	25	10	6	25	10	6
Feature extraction	Classification		AVG (%)			SD (%)	
Float	Float	100	100	100	N/A	N/A	N/A
Float	Fixed	100	100	100	N/A	N/A	N/A
Fixed (proposed FFT)	Float	100	99.2	98	N/A	1.69	0.63
Fixed (*fftgen* FFT)	Fixed	30.8	25.6	9.7	6.94	7.59	1.63
Fixed (proposed FFT)	Fixed	100	99.6	95.8	N/A	1.27	1.17

In speaker identification, the distortion (12) of input speech is computed for codebooks of all speakers stored in the speaker database. The result is a list of speakers and matching scores, sorted according to the score.

5. SPEAKER RECOGNITION EXPERIMENTS

In our training-recognition experiments, we use 8 kHz signal sampling rate, $\alpha = 0.97$ for the preemphasis, 30-millisecond frame length, 10-millisecond frame overlap, Hamming window, FFT size 256, 30 filters in mel FB, and 12 coefficients from the DCT. The GLA speaker modeling uses 5 different random initial solutions picked from the training data. The codebook size is 64. We use 1-norm in (12) instead of the usual 2-norm. Everything else is kept as defined above. The motivation for using the 1-norm is the decreased computational complexity. Before the experiments, we compared two systems where the only difference was the norm in (12) and there was no difference in recognition rates between 1-norm and 2-norm.

5.1. Simulations with PC

The TIMIT corpus has 630 speakers, 10 speech files per speaker. We divided them into independent training and test data consisting of 7 and 3 files, correspondingly. The results of the TIMIT experiments are listed in Table 3.

There are three columns of average recognition rates and three corresponding columns of standard deviations in Table 3. The statistics are computed for recognition rates in repeated cycles of training and recognition for subsets of 16, 25, and 100 speakers from the TIMIT corpus. The effect of the random initial solutions for the GLA, that are sampled from the training data, is taken into account in two ways. First, for each of the three TIMIT subsets, we use the same randomly picked GLA initial solutions in all experiments with the different computational techniques. On the other hand, repeating the same run with same technique but different GLA initial solutions informs us about the effect of randomness in the recognition accuracy. The standard deviation of the recognition rate measures it. If the recognition rate was the same in all repeats, we inserted "not available" (N/A) for the standard deviation. The used number of repeated training and recognition cycles was 25 repeats for the 16-speaker subset, 10 repeats for the 25-speaker subset, and 6 repeats for the 100-speaker subset.

For all used database sizes, the accurate floating point implementation of the MFCC-based speaker identification performs perfectly. The same is true even if we use the accurate features with a less accurate fixed point classification. If we use the fixed point features (proposed FFT) in combination with the floating point classification, the recognition rate decreases slightly. Based on this, we conclude that the numerical accuracy of the signal processing is more important to the recognition accuracy than the numerical accuracy of the classification.

When we use the straightforward fixed point implementation, less than 10 out of 100 speakers are identified correctly. The reason is the FFT inaccuracy. When the *fftgen* FFT is replaced by the proposed FFT, the recognition rate increases near the 100% level again.

5.2. Mobile phone

We tested our implementation in a Nokia 3660 mobile phone for some time outside the laboratory conditions. The recognition accuracy was poor and we decided to investigate the effect of different signals. We created a 16-speaker GSM/PC corpus of dual recordings, which was later extended to consist of 25 speakers. The speech was recorded to two files simultaneously with a Symbian phone via the Symbian API, and with a laptop that was equipped with a basic PC microphone. The PC microphone was attached to the side of the phone with a rubber band. Each recorded file consists of nearly 1 minute of speech. All speakers spoke the same text.

For each speaker, the recording program was started manually in both devices, so the signal contained in the pairs of recorded sound files are little misaligned. The first 16 files were clear speech. The extended data set has many files with a mixture of speech and a lot of impulsive noise caused by scratching the microphones. However, we used all available data in the experiments.

A visual spectrum analysis showed systematically different frequency content in all pairs of recorded files. The highest and lowest frequencies were attenuated in Symbian recordings. We wanted to measure the exact effect of it in the recognition rate. Therefore, before the experiment, the speech contained in all pairs of sound files was aligned in

Table 4: Recognition rate average and standard deviation for GSM/PC experiments with 25 speakers, 5 repeated cycles of training, and recognition.

Audio	Software	AVG	SD
PC	Float	100.0	N/A
PC	Fixed	100.0	N/A
Symbian	Float	83.2	4.38
Symbian	Fixed	76.0	2.83

time by using a multiresolution algorithm, so that we have file pairs where the only difference is the used microphone. There were 3 pairs in the extended data set where our automatic time-alignment method could not perfectly align the pair of signals. Those files were used as such regardless of a possible misalignment. After the MFCC computation, features resulting from all files were similarly split into separate training and test segments.

We repeated the training and recognition cycle 5 times for all combinations of GSM and PC data, and two implementations (the floating point implementation and the proposed algorithm). We eliminated the effect of the random GLA initial solutions by using the same initial solutions for both data sets and for the different implementations. Table 4 lists the results. If the recognition rate was the same in all repeats, we inserted "not available" (N/A) for standard deviation.

Based on the statistics in Table 4, we conclude that the Symbian sound recordings have a negative effect on the speaker recognition accuracy, when compared to PC microphone recordings of the same speech. Also we notice that the recognition rate depends on whether we use floating point arithmetic or fixed point arithmetic. However, the audio source is the most significant factor.

6. CONCLUSION

We ported an MFCC-based speaker identification method to Series 60 mobile phone. We encountered four problems: limited memory, numeric accuracy, processing power, and Symbian programming constraints. A careful numerical analysis helped us to achieve good recognition accuracy in the fixed point implementation. The memory usage and computational complexity of the speaker identification algorithms are low enough for interactive operation in today's mobile phones. The Symbian programming constraints require some learning effort from a programmer familiar with more common platforms.

The numerical accuracy of the MFCC signal processing is important to the speaker recognition, especially the FFT accuracy. Recognition is accurate with floating point signal processing, even if fixed point arithmetic is used for the classifier. If we combine fixed point signal processing (proposed FFT) and the accurate classification, the recognition rate slightly decreases. The signal processing accuracy is more important for correct recognition than the classifier accuracy.

The recognition results are poor when only fixed point arithmetic is used by the system and we are using the *fftgen* FFT. When the FFT is replaced by the proposed FFT, the results are good again. The FFT seems to be the most critical part in the fixed point implementation.

Further improvement could be obtained by utilizing a better filter bank [1], and replacing DCT with a transformation which is optimized for discrimination of speakers.

The FFT we implemented has a double loop. The innermost loop table indexes are computed from the outermost loop index. A better solution would integrate the proposed accuracy improvements in the *fftgen* method.

We also plan to include in our Symbian port the speed improvements that were introduced in [17].

The sound quality is currently the biggest problem. The audio system of the phone attenuates frequencies below 400 Hz and above 3400 Hz, because these are not needed in telephone networks. This has a negative effect on the recognition rate.

7. ACKNOWLEDGMENTS

The research was carried out in the Project *New Methods and Applications of Speech Processing*, http://www.cs.joensuu.fi/pages/pums, and was supported by the Finnish Technology Agency and Nokia Research Center.

REFERENCES

[1] T. Kinnunen, *Spectral features for automatic text-independent speaker recognition*, Licentiate thesis, Department of Computer Science, University of Joensuu, Joensuu, Finland, February 2004.

[2] T. Kinnunen, V. Hautamäki, and P. Fränti, "On the fusion of dissimilarity-based classifiers for speaker identification," in *Proc. 8th European Conference on Speech Communiation and Technology (EUROSPEECH '03)*, pp. 2641–2644, Geneva, Switzerland, September 2003.

[3] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, no. 1, pp. 84–95, 1980.

[4] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1993.

[5] O. Gunasekara, "Developing a digital cellular phone using a 32-bit microcontroller," Tech. Rep., Advanced RISC Machines, Cambridge, UK, 1998.

[6] Digia Incorporation, *Programming for the Series 60 Platform and Symbian OS*, John Wiley & Sons, Chichester, UK, 2003.

[7] R. Harrison, *Symbian OS C++ for Mobile Phones*, John Wiley & Sons, Chichester, UK, 2003.

[8] J. Walker, *Fast Fourier Transforms*, CRC Press, Boca Raton, Fla, USA, 1992.

[9] E. Lebedinsky, "C program for generating FFT code," June 2004, http://www.jjj.de/fft/fftgen.tgz.

[10] T. Thong and B. Liu, "Fixed-point fast Fourier transform error analysis," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 24, no. 6, pp. 563–573, 1976.

[11] P. Kabal and B. Sayar, "Performance of fixed-point FFT's: rounding and scaling considerations," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '86)*, vol. 11, pp. 221–224, Tokyo, Japan, April 1986.

[12] J. Saastamoinen, *Explicit feature enhancement in visual quality inspection*, Licentiate thesis, Department of Mathematics, University of Joensuu, Joensuu, Finland, 1997.

[13] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, vol. 3, pp. 1381–1384, Seattle, Wash, USA, May 1998.

[14] S. Dattalo, "Logarithms," December 2003, http://www.dattalo.com/technical/theory/logs.html.

[15] M. Arnold, T. Bailey, and J. Cowles, "Error analysis of the Kmetz/Maenner algorithm," *Journal of VLSI Signal Processing*, vol. 33, no. 1-2, pp. 37–53, 2003.

[16] "Discrete cosine transform," in Wikipedia, the free encyclopedia, July 2004, http://en.wikipedia.org/wiki/Discrete_cosine_transform.

[17] T. Kinnunen, E. Karpov, and P. Fränti, "Real-time speaker identification and verification," to appear in *IEEE Trans. Speech Audio Processing*.

Juhani Saastamoinen received his M.S. (1995) and Ph.Lic. (1997) degrees in applied mathematics from University of Joensuu, Finland, and the ECMI Industrial Mathematics Postgraduate degree in 1998. Currently, he is doing automatic speech analysis research in the Department of Computer Science in the University of Joensuu.

Evgeny Karpov received his M.S. degree in applied mathematics from Saint-Petersburg state University, Russia, in 2001, and the M.S. degree in computer science from the University of Joensuu, Finland, in 2003. Currently, he works at the Nokia Research Center in Tampere, Finland, and is a doctoral student in computer science in the University of Joensuu. His research topics include automatic speaker recognition and signal processing algorithms for mobile devices.

Ville Hautamäki received his M.S. degree in computer science in 2005 from the University of Joensuu where he is currently a doctoral student. His main research topic is clustering algorithms.

Pasi Fränti received his M.S. and Ph.D. degrees in computer science in 1991 and 1994, respectively, from the University of Turku, Finland. From 1996 to 1999, he was a postdoctoral researcher of the Academy of Finland. Since 2000, he has been a Professor in the University of Joensuu, Finland. His primary research interests are in image compression, pattern recognition, and clustering algorithms.