Automatic Speaker Recognition for Series 60 Mobile Devices

Juhani Saastamoinen, Evgeny Karpov, Ville Hautamäki, Pasi Fränti
University of Joensuu, Dept. of Computer Science, P.O.Box 111, 80101 Joensuu, Finland
{juhani,ekarpov,villeh,franti}@cs.joensuu.fi

3rd August 2004

Abstract

Mobile phone environment and other devices that require low power consumption, restrict the computations of DSP processors to fixed point arithmetic only. This is a common practise based on cost and battery power savings. In this paper, we first discuss the specific software architecture requirements set up by the Symbian operating system and the Series 60 platform. We analyse mel frequency cepstral coefficient based classification, and show techniques to avoid information loss, when a floating-point algorithm is replaced by a corresponding algorithm that uses fixed-point arithmetic. We analyse the preservation of discrimination information, which is the key motivation in all classification applications. We also give insight to the relation between information preserving and operator presentation accuracy. The results are exemplified with tests made on algorithms that are identical except for the different arithmetics used.

1 Introduction

We are working in a speech technology project, where one of the main goals is to integrate speaker recognition technique in Series 60 mobile phones. The classification algorithm that we use in this paper is a common unsupervised learning vector quantizer.

Our research team has been developing speaker recognition methods based on signal processing models that are commonly used in speech recognition, and a generic automatic learning classification. Up to now, we have mainly been doing research on the closed set speaker identification problem. We have been looking into different aspects of the problem. A thourough investigation on the effect of the feature vector computation is presented in [8]. We reported some real-time speaker identification results in [10]. We have also investigated the concurrent use of several speaker cues [9].

In this paper, we consider application of closed set speaker identification. In general, this means that we are looking at a system where mathematical models of voices of N speakers are created and stored in a speaker database. During the recognition a speech sample is compared to the models in the database. The result is the best identified speaker, or a list of several best matched speakers along with matching scores. We have ported such application to a Symbian mobile phone with Series 60 user interface. In this report, we study the properties of this platform and our implemented system. In particular, we focus on the numerical analysis of the signal processing algorithms converted to fixed point arithmetic. We also discuss the effect of numerical round-off error in dicrimination properties of the classification.

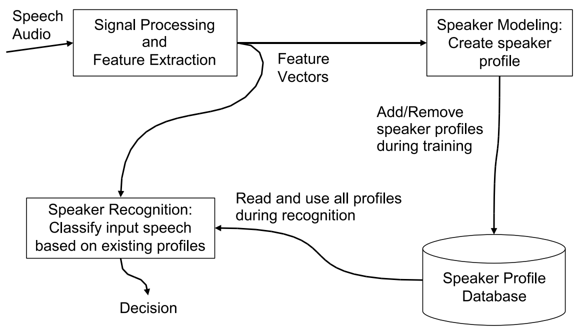
Speech Audio
Signal Processing and Feature Extraction
Feature Vectors
Speaker Modeling: Create speaker profile
Add/Remove speaker profiles during training
Speaker Recognition: Classify input speech based on existing profiles
Read and use all profiles during recognition
Speaker Profile Database
Decision

Figure 1: Closed set speaker identification system

This research was carried out in the project *New Methods and Applications of Speech Processing* [1] and was supported by the Finnish Technology Agency and Nokia Research Center.

2 Speaker identification system

We consider a speaker identification system with separate modules for speech signal processing, training and classification, and speaker database (Fig. 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 (Fig. 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 feature vector set separately for each speaker. 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 from each speaker. Each feature vector set is replaced by a *codebook*, a smaller set of

[1] *Puheenkäsittelyn uudet menetelmät ja sovellukset* (PUMS), `http://cs.joensuu.fi/pages/pums`

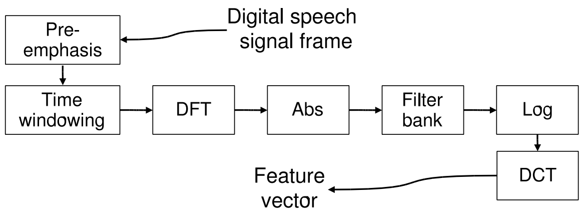

Figure 2: MFCC signal processing steps

code vectors with fixed size that is stored in the speaker database to represent the speaker. A common codebook design goal is minimizing quantization distortion of training data, i.e. we look for code vectors that minimize the quantization distortion, when training vectors are replaced by their nearest neighbours from the codebook. We use the *generalized Lloyd algorithm* (GLA) [12] to minimize the mean squared error (MSE) distortion.

The *recognition mode* input speech sample is processed with a signal processing model identical to the training. The resulting feature set is matched against the speaker database by computing the distortion of the input data against each stored codebook. This is done by quantizing the data using the codebook of each candidate and computing how much it is distorted. The speaker whose codebook gives the least distortion is identified. If N-best decision is requested, the system returns N smallest distortions and corresponding speakers.

The signal processing of the system uses the *mel frequency cepstral coefficient* (MFCC) computation, which is commonly used in speech recognition [13]. The speech is divided into overlapping frames, within a frame the pre-emphasized signal is multiplied with a windowing function, and a Fourier spectrum is computed. A mel filter bank is applied to the magnitude spectrum, and logarithm of the filter bank output is finally cosine transformed. The first coefficient of the cosine transform is omitted as an energy normalization step, and only part of cosine transform output coefficients are used as the feature vector.

The filter bank spectra of speech are non-negative. Typically their shape varies around a certain set of typical vectors corresponding to the phones appearing in speech. Another major source of variance is speaker dependent variability. However, it appears that the speech energy lost in truncation is always small as all the energy is concentrated in the beginning of the representation in the cosine basis. Figure 2 illustrates the signal processing steps.

3 Symbian OS

Nowadays market for mobile phones is changing very rapidly. Sales are highly dependent on the innovative features available in particular phone. These are, for example, handling of still and moving images, wireless communications, among many others. In these circumstances, software developing for mobile phones becomes a complex task. To reduce this cost the leading manufactures have co-operatively developed a new industry standard operating system for mobile phones, known as Symbian OS [7].

3.1 Mobile devices

The mobile devices are small in size, which leads to greater demand on manufacturers. To meet these requirements each new hardware design must be cheap to manufacture, fit to small space, and have low power consumption.

Most commonly used mobile phone processors are developed by the company Advanced RISC Machines (ARM). These are fully 32-bit RISC processors with 4 GB of address range. A three-stage pipeline is used there, which allows executing one instruction every cycle [6].

Modern ARM processors implement 32-bit wide instruction set as well as new *Thumb* instruction set, which is 16-bit wide. Therefore it needs less ROM to store program binaries and works fast with narrow memory [6]. Traditional 32-bit instruction set is compressed into 16-bit Thumb instruction set, which is then decompressed and executed at execution time. Thumb decode is very simple and it is possible to decompress it on the fly without additional cycles [6]. Usually there is no floating point arithmetic support in such processors because of its complexity and hard power consumption.

3.2 Series 60 Platform

Symbian was formed from Psion Software in 1998 by the leaders of wireless industry: Nokia, Ericsson, Panasonic, Motorola, Psion, Siemens, and Sony Ericsson. The goal of Symbian was to develop an operating system for advanced, data-enabled mobile phones [3]. Symbian OS evolved from EPOC operating system developed by Psion. It has modular microkernel-based architecture [3]. The core of Symbian OS consists from the *base* (microkernel and device drivers), *middleware* (system servers) and *communications* (telephony, messaging, etc.) [3].

Symbian OS is fully multitasking operating system. It has support for simultaneously running processes, threads, separate address space, and pre-emptive scheduling [7]. However, because it runs on limited performance mobile phones, it is recommended that most applications use built-in framework, called *Active Objects*, which implements non-pre-emptive multitasking [3]. Symbian OS also has a file system, which is stored in phone ROM or RAM memory, or on small removable flash-disks. It also supports dynamic link libraries [3].

Although Symbian OS is used in most mobile phones, they are further equipped with different user interface (UI) platforms. A 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 80, and Series 60 (developed by Nokia).

3.3 Programming for Symbian OS

There are two programming languages that can be used to write programs that run on Symbian OS phones: Java and C++. However, Java has very limited API and execution speed, therefore real programs are usually written in C++. Symbian also has limited support for ANSI C standard library [3, 7]. Symbian OS offers a wide range of different API's for C++ programmer.

Almost all programs developed for Symbian OS can first be tested and debugged on an emulator running in a PC. Manufacturers provide emulators for different user interface platforms. Emulators have few distinctions, compared to real phone. Therefore applications are usually first tested on emulator and only after that they are run on real hardware. Emulators can also simulate exceptional situations like absence of free memory or file system errors [3, 7].

The main difference of Symbian OS programming from conventional PC is that 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 a higher priority. Programs must also be as small and efficient as possible to not overwhelm limited hardware resources. Robustness is also important, because mobile phones are supposed to work without restart for months or even more [7].

Algorithms must be selected carefully so that there are only numerically stable ones with low time complexity. However, there is no floating-point arithmetic support by hardware, it is emulated by software components. But this option should be used very rarely because of its complexity and higher power consumption. Instead algorithms must be ported to use fixed-point arithmetic only.

There are several restrictions to C++ language made by Symbian OS. First of all, there is no standard C++ exception handling. Instead Symbian desingers have implemented their own mechanism [7]. This has been done mainly because GCC compiler, used in target builds, had no support for this in the time when Symbian was designed [7]. First consequence of this decision is that C++ class constructor cannot create any other objects because this can cause, for example, out-of-memory exception but there is no way in Symbian to handle exceptions from constructor. To overcome this problem Symbian uses *two-phase construction*, where object is first created and then initialized [7]. Second consequence is that memory stack is not unrolled after exception and programmer must use special framework called the *cleanup stack* that is used to manually maintain the stack, which will be unrolled by the framework after exception occurs [7]. This framework requires that all objects that can be allocated in the heap are derived from common base class (*CBase*), added to the stack immediately after allocation and removed only just before deletion [7]. Basically, absence of C++ exception handling adds more work that the programmer must do by hand and what is usually compiler task in conventional C++ programming.

Another important aspect of Symbian programming is dictated by efficiency requirements. Applications or DLL's can be executed in ROM without copying them first to RAM. This makes another programming limitation: An application stored in a DLL has no modifiable segment and can not use static data [7]. Basically, all applications interacting with the user are stored in a DLL and loaded by framework, when user selects to execute that particular program [7]. However, Symbian provides a *thread local storage* mechanism for static data [7].

We decided to implement most of the computational algorithms in ANSI C language and use POSIX standard where applicable. The reasons for such decision were good portability, an existing prototype written in C, and the (limited) ANSI/POSIX support in the Symbian OS. Symbian OS offers an implementation of the standard C library, so many programs can be easily ported to Symbian OS. Main limitation is that no static data, i.e. global variables can be used. Another restriction is file handling in multi-threaded programs: the function `fopen` and other file processing functions may not work as expected in multi-threaded programs. The Symbian OS developer documentation encourages to use the provided *file server* mechanisms instead.

4 Numerical analysis of MFCC and VQ in fixed point arithmetic

During the speaker recognition the speaker information carried by the signal propagates through the signal processing (Fig. 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.

The MFCC algorithm consists of a lot of evaluations of different vector mappings f between vector spaces, denote such evaluation $f(x)$. During the execution of our computer implementation a lot of values $\hat{f}(\hat{x})$ are evaluated, where $\hat{x}$ is an approximation of x represented in some finite accuracy number system, and $\hat{f}$ is our implementation that tries to capture the behaviour of f. From information theory point of view, it is important to minimize the *relative error* of implementation $\hat{f}$

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

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

Most MFCC processing steps are linear mappings and the non-linear ones behave well, there are two of them. The real valued magnitudes of complex valued Fourier spectrum 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 [13]. 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. Assume that the MFCC input is a stream of signed integers, sampled signal amplitudes represented with 16 bits. In many parts we use different integer value interpretation, a scaling integer $I > 1$ represents 1 in the normal algorithm. In some parts we also divide input, output, or intermediate results to ensure they fit in the used integer type, usually a 32-bit integer. Let us now analyse the system.

4.1 Pre-emphasis

Many speech processing systems apply a *pre-emphasis* filter to the signal before further processing. The difference formula $x_t = x_t - \alpha x_{t-1}$ is applied to the signal x_t, our choice is a common $\alpha = 0.97$. Such filter emphasizes higher frequencies and damps the lowest.

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 artefacts 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 smoothly forced to zero near the endpoints. The smooth transition requires that we use enough bits to represent the window function values.

4.3 Fourier spectrum

The frequency spectrum of a digital signal can be computed as the N-point *discrete Fourier transform* (DFT) $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$

$$\mathcal{F}(x) = \sum_{t=0}^{N-1} \mathrm{e}^{2\pi \mathrm{i}\omega t/N} x_t, \quad \omega = 0, \ldots, N-1. \qquad (1)$$

As a (unitary) linear map, $\mathcal{F}$ has a corresponding matrix $F \in \mathbb{C}^{N\times N}$, and $\mathcal{F}(x)$ can be computed as a matrix-vector product Fx using $\mathcal{O}(N^2)$ operations. The radix-2 fast Fourier transform (FFT) is a tool for computing DFT efficiently for $N = 2^m$, $m > 0$. It utilizes the structure of F and computes Fx in $\mathcal{O}(N \log N)$ operations.

Each element $\lambda_k = (Fx)_k = \sum_{t=1}^{N} F_{kt} x_t$ of the product is linear combination of N elements of x, whereas the FFT algorithm actually transforms the computations to $\log_2 N$ *layers* of $N/2$ *butterflys*

$$\begin{aligned} f_t^{l+1} &= f_t^l + W_t^l f_{t+T}^l, \\ f_{t+T}^{l+1} &= f_t^l - W_t^l f_{t+T}^l. \end{aligned} \qquad (2)$$

The time complexity of one butterfly is $\mathcal{O}(1)$. The superscripts denote the layer. The first layer input is the signal $f_k^0 = x_k$, $k = 0, \ldots, N-1$. The offset constant T depends on the layer, and whether we are using decimation in time FFT or decimation in frequency FFT [15]. Decimation means reorganizing either input or output to a *bit reverse order* [15] and takes $\mathcal{O}(N)$ operations. When both decimation and butterfly computations are done, the outputs $f_k^{\log_2 N}$ consist of the same values as λ_k above.

The FFT efficiency is based on the layer structure. But fixed point implementations, like code generated with *fftgen* [11], introduce significant error. The fixed point arithmetic round-off errors accumulate in the repeatedly applied butterfly layers. We gain back some the lost accuracy by improving the information preserving properties in our fixed point implementation.

4.3.1 FFT in fi xed-point arithmetic

The FFT code generated by *fftgen* [11] has the butterfly layers and the element reordering all merged in few subroutines, with all loops unrolled. It uses 16-bit integers for the input signal, intermediate results, and the automatically computed power spectrum output. Multiplication results in (2) are 32-bit integers, but stored in a 16-bit integers after shifting 16 bits to the right. This is necessary in order to keep the next layer input in proper range. Overflow in addition and subtraction is avoided by shifting inputs 1 bit to the right appropriately. Such truncations introduce round-off error and information loss.

We employed the generated FFT code in the fixed point MFCC implementation and compared to its floating point counterpart. The MFCC outputs computed from identical inputs with the two implementations did not correlate much. One reason for this may be accumulating errors. However, detailed analysis of results showed that the first source of large relative error is the FFT (DFT in Fig. 2). We also verified that the large error is not due to the final truncation of power spectrum elements to 16 bits, but the FFT algorithm itself.

We started to improve the accuracy with an existing radix-2 complex FFT implementation [14]. First we changed the data types, additions, and multiplications similar to the *fftgen*-generated code.

The generated code uses 16 bits for the layer input real and imaginary parts, as well as real valued trigonometric constants arising from (2), after the Euler formula $\mathrm{e}^{\mathrm{i}\varphi} = \cos\varphi + \mathrm{i}\sin\varphi$ is applied in (1). We changed the data type of intermediate results in (2) from 16-bit to 32-bit integers. But this alone does not really help to preserve more than 16 bits of intermediate results (signal information) if operator constants still use all 16 bits. The multiplication would overflow. A solution is to reduce the DFT operator representation accuracy in order to increase the 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 numeric accuracy. Now repeat the same analysis but consider a linear butterfly layer in FFT algorithm $g = Gy$ and its implementation $\hat{g} = \hat{G}\hat{y}$. The inputs $\hat{y}$ carry information about accurate values y, i.e. information about signal x. In (2), each multiplication of the layer input $f^l \in \mathbb{C}$ with the operator constant $W^l \in \mathbb{C}$ expands to two additions and four multiplications of real values. If we use more than 16 bits for f^l, we must use less bits for the operator implementation constants W^l, in order to fit the multiplication result in 32-bits. Thus we allow the relative layer operator error $||G - \hat{G}||/||G||$ to increase while 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. More information about y propagates to the next layer input $\hat{g}$ in all layers. Thus information loss decreases in the whole implementation. 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. Here the norm and difference of linear operators have their usual definition. This is the main idea and can also be applied to other algorithms implemented in fixed point arithmetic.

The N-point DFT uses the values $\pm \sin \pi m/N$ and $\pm \cos \pi m/N$, $m = 0, \ldots, N/2 - 1$. Before deciding how many bits to use for the signal and the operator, we look at the relative round-off error made in these values for different FFT sizes N and bit usages $B > 0$. For each B, we find the scaling integer minimizing the maximum relative round-off error

$$E(c, N) = \max_{m=0,\ldots,N/2-1} |s_m - \hat{s}_m|/|s_m|, \qquad (3)$$

where $s_m = c \sin \pi m/N$ and $\hat{s}_m$ denotes s_m rounded to the nearest integer. It is enough to consider plus signed sines, since cosine values are in the same set. We found out that for $N = 256$, 512, 1024, 2048, and 4096 there are several peaks downwards in the graph of $E(c, N)$ as function of c, that are good choices of c even if they do not minimize $E(c, N)$ for certain N. Table 1 shows pairs of such good values c and $E(c, N)$ for different N. The bit usage B is 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 constant $c = 980$ appears to be the best choice with $B = 10$. That leaves 22 bits for the signal information. We replaced the signal/operator bit usage $16/16$ with $22/10$. We did not check all possible combinations of B and c. There may be better choices in terms of the relative error, but our choice fits our purpose, as we mostly use $N = 256$.

In our floating point MFCC, we compute the FFT with the Fastest Fourier Transform in the West (FFTW) C library [5]. The FFTW relative error is very small and we compare fixed point algorithms to it. Figures 3–5 illustrate the different error of the *fftgen* FFT, and the proposed FFT with $c = 980$ and $B = 10$. The input speech in Fig. 3 is near the beginning of the first file of the first speaker in the TIMIT corpus, but sampled at 8 kHz.

Table 1: Selected $c/E(c, N)$ value pairs with small $E(c, N)$ and different FFT sizes N; the values $E(c, N)$ have been multiplied by 10^3.

$N = 256$	512	1024	2048	4096
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			

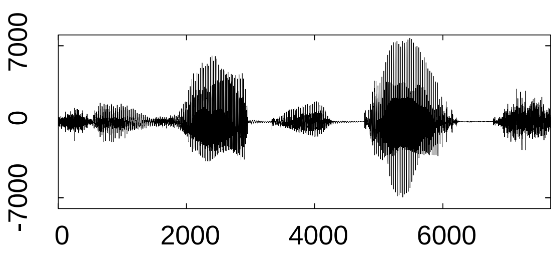

Figure 3: Speech near the beginning of speaker 1 file 1 in the TIMIT corpus.

Fig. 4 has two scatter plots of pairs of logarithms of absolute values of *fftgen* FFT and a floating point FFT (FPFFT). Fig. 5 has the same for proposed FFT and FPFFT value pairs. Without error all dots reside on a straight line. Comparison of the FFT's of the original signal x (left) and a scaled $4x$ (right) shows that in a fixed point arithmetic we may decrease the error when using the integer scale more efficiently, relative round-off error is smaller. The proposed FFT is accurate even without scaling, also note the increased range of accurate values.

4.4 Magnitude spectrum

The Fourier spectrum is $\{\lambda_k; k = 0, \ldots, N/2 - 1\}$, the power spectrum is $\{|\lambda_k|^2\}$, and the magnitude spectrum $\{|\lambda_k|\}$ consists of the absolute values. There is little difference in using the square or not in the filter bank input. However, the usage of the fixed point number range in the power spectrum values is non-uniform. The same is true for magnitude spectrum, but the usage of the value range is not as sparse as with power spectrum. By the uniformity of the number range usage we mean the density of

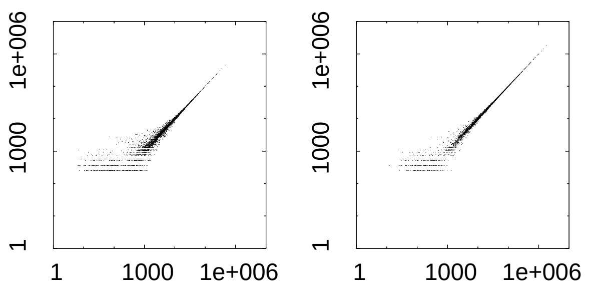

Figure 4: Scatter plots of *fftgen* FFT output (x-axis) against FFTW output (y-axis). Scales are logarithmic.

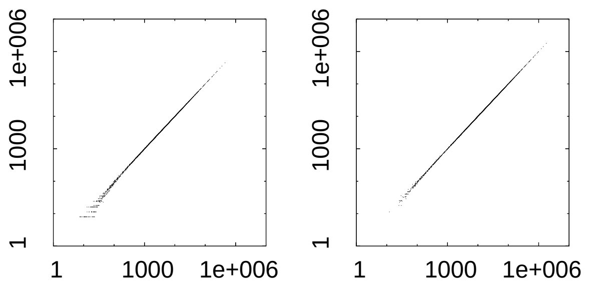

Figure 5: Scatter plots of proposed FFT output (x-axis) against FFTW output (y-axis). Scales are logarithmic.

the values $|\lambda_k| = |x_k^2 + y_k^2|$ or $|\lambda_k|^2$, when x_k and y_k take all possible 32-bit integer values. The FFT output is a vector of complex numbers. The absolute value of a complex value is a square root

$$|z| = |x + \mathrm{i}y| = \sqrt{x^2 + y^2} \tag{4}$$

In our experiments we tried the familiar and efficient *Newton iteration* [4] applied to the square root. We also developed an efficient non-iterative solution.

The square root argument in (4) allows an efficient polynomial approximation. Without loss of generality, assume that $|x| \geq |y|$. Then (4) can be written

$$|z| = \sqrt{x^2 + y^2} = |x|\sqrt{1 + \left(\frac{y}{x}\right)^2},$$

where $1 + \left(\frac{y}{x}\right)^2 \in [1, 2]$ always. By introducing a parameter $t = |y/x| \in [0, 1]$ we can approximate $|z|$ with

$$|z| = |x|\sqrt{1 + t^2} \approx |x|P_n(t),$$

where $P_n : [0, 1] \to [1, \sqrt{2}]$ is a polynomial of order $n \geq 1$ with $P_n(0) = 1$ and $P_n(1) = \sqrt{2}$. To ensure that the boundary conditions are met, 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 $S = \{t - t^2, t - t^3, t - t^4, t - t^5\}$ defined for $t \in [0, 1]$, i.e. fit a least squares polynomial. Our approximation is

$$\begin{aligned}\sqrt{1 + t^2} \quad \approx \quad & 1 + (\sqrt{2} - 1)t \\ & - 0.505404(t - t^2) + 0.017075(t - t^3) \\ & + 0.116815(t - t^4) - 0.043182(t - t^5).\end{aligned}$$

The maximum relative approximation error for $t \in [0, 1]$ is 1.30×10^{-5}.

The motivation for our boundary conditions is that a least squares polynomial often maximizes approximation error in the endpoints of closed interval. We are approximating a function, therefore trying to minimize the maximum error, instead of minimizing the integrated error. The idea of selecting basis to meet boundary conditions is borrowed from the *finite element method* for solving partial differential equations.

There are likely numerically better choices for basis besides S. However, it is straightforward to evaluate t^{k+1} from t^k and t in our scaled integer arithmetic. Also S is a basis and meets boundary conditions. Note also that $0 \leq t, t^k, t - t^k \leq 1$ for $t \in [0, 1]$.

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

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

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.

If the parameter is $s = (y/x)^2$ we get much smaller maximum relative approximation error 1.62×10^{-6}. In this case we approximate different function $\sqrt{1 + s} - (1 + (\sqrt{2} - 1)s)$ similarly as above. However, when $s = t^2$ is used instead of t, there are more multiplication-rescale combinations in fixed point implementation in powers of s. For example, computing s^4 means computing $(y/x)^8$. Such high powers introduce more error.

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. Applying a linear filter bank (FB) means applying several filters, and 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.

The choice of FB has a large effect on the speaker recognition accuracy [8]. Our choice is a common, but not optimal mel FB with triangular filter shape.

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

4.6 Logarithm

A lot of work has been done on the accurate evaluation of the logarithm function for integers, for example [2] introduces several methods for computing integer logartihms and [1] analyses the error in several methods. Our choice is efficient and has low error. We compute logarithms $\log_2 t$, $t > 1$ by using lookup tables and piecewise linear interpolation. If $t > 2$, the problem is first transformed to logarithm problem in the set $(1, 2]$ by using the recursion $\log_2(t) = 2\log_2(t/2)$ [2]. The maximum relative error in the logarithm evaluation depends on the amount of bits used for the lookup table indices. We use 8 bits and reach the maximum error 4.65×10^{-6} at $t = 272063$.

4.7 Discrete cosine transformation

The *discrete cosine transformation* (DCT) is a linear invertible mapping, that 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 DCT output, so we compute it with the direct formula without FFT. We utilize the most common DCT form [16, DCT-II]

$$\mu_j = \sum_{k=0}^{N_{\mathrm{FB}}-1} l_k \cos\left(\frac{\pi}{N_{\mathrm{FB}}}\left(k + \frac{1}{2}\right) j\right), \qquad (5)$$

where $j = 0, \ldots, N_{\mathrm{MFCC}} - 1$, input l_k consists of filter bank outputs or their logarithms, $k = 0, \ldots, N_{\mathrm{FB}} - 1$, and N_{MFCC} is the number of the MFCC coefficients needed. Usually μ_0 is ignored, as it only depends on the signal energy. This DCT form is orthogonal if μ_0 is multiplied by $1/\sqrt{2}$ and all coefficients are output [16]. The DCT is applied to filter bank outputs in speech applications for few reasons. It rescales and decorrelates the filter bank outputs. This is important for the GLA clustering and the VQ classification.

We did not analyse the DCT error in the fixed point imlementation. Instead, we simply assigned the scaling factor 32767 for cosine values and truncate 16 bits from the 32-bit input values. Similar analysis as with the FFT above would help to gain accuracy.

4.8 System training and recognition

The GLA algorithm constructs a codebook $\{c_k\}$ that minimizes the MSE distortion

$$\mathrm{MSE}(X, C) = \sum_{j=1}^{N} \min_{1 \le k \le K} ||x_j - c_k||_2^2, \qquad (6)$$

of 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. Also the round-off error in the MSE distance computations is different in fixed point arithmetic.

In speaker identification, the distortion (6) 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.

Table 2: Identification rates for two different implementations with 100 first speaker subset of the TIMIT corpus.

Implementation	Identification rate (%)
Fixed point arithmetic	82
Floating point arithmetic	100

Table 3: Identification rates in GSM/PC experiments.

Rec. device	13/16	14/16	15/16	16/16
Symbian audio	1	3	3	10
PC audio	0	0	0	17

5 Experiments

In our training-recognition experimients, we used 8 kHz signal sampling rate, $\alpha = 0.97$ for pre-emphasis, 30 ms frame length with 10 ms overlap Hamming window, FFT size 256, logarithm of mel FB with 27 outputs, and finally 12 coefficients from the DCT. The GLA speaker modeling used 5 different random initial solutions picked from the training data. Codebook size was always 64.

The results with subset of 100 first speakers of the TIMIT corpus are listed in table 2. We eliminated the effect of randomly sampled GLA initial solutions by picking the same initial solutions for both implementations.

We tested our implementation in a mobile phone for some time. The recognition accuracy was poor and we decided to invesitigate the signal. We created a 16-speaker GSM/PC dual recordings data for that. Speech was recorded simultaneously with a Symbian phone via the Symbian API, and a basic PC microphone attached to the phone with a rubber band. All speakers spoke the same text. Indeed, a quick signal analysis showed systematically different frequency content. Before recognition experiments all pairs of nearly 1 min long recodings were time aligned with a multi-resolution algorithm, and finally split to separate training and test segments.

We repeated training/recognition cycles for both GSM and PC data with the fixed point implementation. We eliminated the random effect of GLA initial solutions by using the same initial solutions for both data sets. The worst case identification rate for the phone data was 13/16. We did not carefully record all results but we recall that approximately the frequencies listed in table 3 did occur.

6 Conclusion

We had four main obstacles when porting the speaker recognition PC application to a Series 60 mobile phone: limited memory, numeric accuracy, and processing power, as well as the Symbian programming constraints. The numeric error in the fixed point implementation can be kept at a feasible level with a careful numerical analysis. The memory usage and computational complexity of the speaker identification algorithms are low enough for real-time operation already in today's mobile phones. The programming constraints take some learning effort for a programmer familiar with more common platforms.

Our experience is that the current smart phone hardware has enough resources for real-time speaker identification with small speaker databases. However, the quality of the signal obtained directly from the Symbian audio API, is not good enough as such for reliable recognition, not even with small speaker database.

The recognition accuracy in the fixed point implementation is not yet the same as with a normal floating point implementation, but we are working on it.

7 Future work

We plan to utilize a more efficiently speaker discriminating filter bank. This is needed especially because the tested mobile phone hardware seems to enhance speech frequences while loosing information in frequencies that are useful when discriminating speakers.

If signal pre-conditioning cannot improve the identification results enough, we plan to use a better transformation in place of the DCT. This means that a generic DCT, whose purpose is to normalize or balance filter bank outputs, is replaced by something that would do this optimally with respect to *speech* as opposed to any signal.

Reworking the scaling factors of signal windowing, filter bank, and DCT in the context of the whole MFCC process would give more accurate MFCC output.

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

We are going to extend the current implementation to also handle the open set identification task and also include the real-time methods we have developed [10]. Naturally, the open set problem requires a working background model computation framework. There are such models used in the speaker verification problem, so actually we are going to attack that problem also.

References

[1] Arnold, M., Bailey, T., and Cowles, J., "Error Analysis of the Kmetz/Maenner Algorithm", the Journal of VLSI Signal Processing, vol. 33, iss. 1–2, pp. 37–53(17), 2003.

[2] Dattalo, S., Logarithms, *http://www.dattalo.com/technical/theory/logs.html,* Dec 2003.

[3] DIGIA Inc., Programming for the Series 60 Platform and Symbian OS, John Wiley & Sons Ltd., England, 2003.

[4] Elden, L. and Wittmeyer-Koch, L., Numerical Analysis: An Introduction, Academic Press, Cambrigde, 1990.

[5] Frigo, M., and Johnson, S., "FFTW: An Adaptive Software Architecture for the FFT", ICASSP conference proceedings, Vol. 3, pp. 1381–1384, 1998.

[6] Gunasekara, O., "Developing a digital cellular phone using a 32-bit Microcontroller", Tech. Rep., Advanced RISC Machines Ltd., 1998.

[7] Harrison, R., Symbian OS C++ for mobile phones, John Wiley & Sons Ltd., England, 2003.

[8] Kinnunen, T., "Spectral Features for Automatic Text-Independent Speaker Recognition", Ph.Lic. Thesis, Univ. of Joensuu, Dept. of Computer Science, Feb 2004.

[9] Kinnunen, T., Hautamäki, V., and Fränti, P., "On the fusion of dissimilarity-based classifiers for speaker identification", European Conf. on Speech Communication and Technology, (Eurospeech'2003), Geneva, Switzerland, 2641–2644, Sep 2003.

[10] Kinnunen, T., Karpov, E., and Fränti, P., "A speaker pruning algorithm for real-time speaker identification", Lecture Notes in Computer Science vol. 2688, Int. Conf. on Audio- and Video-Based Biometric Person Authentication (AVBPA'03), Guildford, UK, 639–646, Jun 2003.

[11] Lebedinsky, E., C program for generating FFT code, *http://www.jjj.de/fft/fftgen.tgz*, Jun 2004.

[12] Linde, Y., Buzo, A., and Gray, R. M., An Algorithm for Vector Quantizer Design, Trans. IEEE Commun., Vol. COM-28, pp. 84-95, 1980.

[13] Rabiner, L. and Juang, B.-H., Fundamentals of Speech Recognition, Prentice Hall, 1993.

[14] Saastamoinen, J., "Explicit Feature Enhancement in Visual Quality Inspection", Ph.Lic. Thesis, Univ. of Joensuu, Dept. of Mathematics, 1997.

[15] Walker, J., Fast Fourier Transforms, CRC Press Inc., 1991.

[16] "Discrete cosine transform" in Wikipedia, the Free Encyclopedia, *http://en.wikipedia.org/wiki/Discrete_cosine_transform,* Jul 2004.