UNIVERSITY OF OSLO
Department of informatics

The Use of Java in Different Signal Processing Applications Including a Text Independent Speaker Recognizer Based on Cepstral Coefficients

Mathematical modeling

Kjetil Pedersen

November 2000

Abstract

The results in these thesis show that text independent speaker recognition in a speech utterance composed of several speakers based on a coarse average of the cepstrum coefficients will not give a satisfactory result. Though I find that it could be used with success in simpler speaker verification applications.

Preface

This thesis is written as a required part of the Candidatus Scientiarum (Master of Science) degree in informatics at the Department of Informatics, University of Oslo, Norway. The work was started in January 1999 and ended in November 2000.

The work is a practical and theoretical approach to study the use of the programming language Java in medical ultrasound, in streaming applications for the Internet, and in speech processing. Due to some problems that will be described more thoroughly in the Introduction chapter, the speech processing part is the main part in the thesis. Some background theory is included to motivate the reader and to introduce the notation required in the proposed methods. This will hopefully give a wider understanding of the methods and the conditions they apply to.

Finally, I would like to thank my supervisor, Professor Sverre Holm, for his encouragement and assistance through this work. I would also like to thank all my fellow students, and Helge Fjellestad in particular for helping me with technical as well as theoretical problems during this work, and my family, neighbors and friends for helping me reading the proofs and for participation while carrying through the process of collecting the speech data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The objective of this thesis

The objective of this thesis is to explore the possibility of using the programming language Java in a number of different signal processing applications. This is done on three slightly different tasks - an interaction between Java and C++ to process ultrasound data, a streaming application of video over the Internet, and third, and this is the main part, a text independent speaker recognition application written in Java, that utilizes the encoded data from the MPEG4-CELP coder, used for speech coding at low bit rates, to distinguish different speakers in an input stream. I ought to mention that there has been a couple of difficulties during the work, which is the reason why these thesis consists of three parts. I will deal with these problems below as I give a brief description of the three parts.

The first task is to develop an application where the Java program interacts with a C++ program that defines an ultrasound image format. This is a co-operation with G E Vingmed Ultrasound, and the question is to find out whether or not Java is suitable for this purpose; can it process and update the images quickly enough? As the ultrasound image format is defined in C++, I have to utilise the Java Native Interface (JNI) API (Application Programming Interface) to integrate Java with C++. In this part I show how one with success can use Java for the user interface part of a computational expensive task, and make a connection to the C++ program that takes care of the heavy computations. The interconnection between Java and C++ (or Visual C++ as I used) worked fine, but I could not get the image right. It did not look like the ultrasound image it was supposed to, and I was not able to solve the error.

Further, in the second part, which is a co-operation with the Internet com-

pany Fast Search & Transfer ASA, I use a recently developed Java API, the Java Media Framework (JMF), to exploit the possibility of streaming video over the Internet. This package is described more thoroughly in Chapter 3.3.3. The purpose of this part of the work was first to make an application that was able to stream well known media formats like MPEG, WAV, MP3 and AVI files, and then to implement a codec (encoder/decoder) for FAST's own video format, the FVT (Fast Video Transfer) format. The first part works just fine, the JMF package is well suited for use in streaming applications, but in the second phase, a problem that treated company confidential parts came into existence. To implement a codec for the FVT format, it is, of course, necessary to get access to the source code. But, as this code is company confidential, FAST would not let this source code be stored on one of the university's hard drives. The solution would be for me to do this part of the work in one of FAST's offices, but as they were already in lack of free office space, this was not possible. So, instead, it was decided to take another approach and look at the way that sound is compressed in the FVT format. What they use for this part of the media stream is the newly developed MPEG4 audio format, which is an open source format. This leads to the third part of this thesis.

In the last part the question is how well suited the encoded data from the MP4 encoder are in a speaker recognition application, and especially a text independent speaker recognition application. There would be a great computational benefit if it is possible to avoid decoding the data before we analyze them. In this part I show that the data, when transformed to so-called *cepstral coefficients* can be used in certain speaker verification application, but more sophisticated methods should be used in the text independent recognition tasks, and to improve the results when dealing with speaker verification. With this simple method the error rate is too high to be used for practical purposes, but if the method is exended it should be possible to obtain fairly good results. These extensions are suggested in Chapter 6, *Results and Further work*. I start with a speaker verification test, one where all the training utterances are the same, and one where they are all different, before I move on to the text independet speaker recognition test. This is a more challenging task, as I have to segment the speech sample into speakers. The difficulties this involves is described more thoroughly in the Method section.

Even though the first two projects were not completed the way they were supposed to be, I did use quite an amount of time reading background material and programming. Therefore I decided to include some of this theory in these thesis.

1.2 Java

Java was introduced in the late 1995, and it took the Internet with storm. One of the reasons why it became so popular was the ability to add fancy graphics and music to your web-pages. The language, developed by Sun Microsystem, is platform-independent. This means that you write your code once and can then run the program on any machine-architecture. This is another - and the main - reason for the popularity.

The platform-independence is at the same time one of the major drawbacks. When you compile the Java code the program is compiled into architecture neutrale byte-code format, and not architecture dependent machine-code. To run this byte-code your machine must implement the Java virtual machine (JVM). The JVM is the interpreter and run-time system. Even though the speed of the Java interpreter has increased tremendously over the past years, it will still not run as fast as e.g. C-code or C++-code. Perhaps this is just a matter of time as the compilers go through the process of development. One has to remember that especially C is an old language, and the compiler has been optimized during several years.

Chapter 2

Medical Ultrasound Imaging

Sound is one of the most important carriers of information. Sound is pressure waves that propagate through a medium. When, for instance, a tuning-fork is struck against something, it starts to vibrate and influences the molecules in the surrounding air. These molecules will then start to vibrate with the same frequency as the tuning-fork, and the variations in the air pressure will propagate through the air which make the membrane vibrate [3].

Sound with higher frequency than humans are able to hear, is called *ultrasound*. The frequency is typically in the range 2 - 10 MHz for use in medical imaging. Ultrasound has been examined for decades, but it was not before the 1930s and 1940s that the two brothers Friedrich and Theodore Dussik[1] discovered its potential in medical diagnostic. What really inspired the early ultrasound investigators was the SONAR (SOund Navigation And Ranging) - the technique of sending sound waves through water and observing the returning echoes to characterize subemerged objects. Even though it was discovered in the 1930s, the major breakthrough came in the 1970s with the B-mode presentation of two dimensional gray-scale imaging. I will discuss B-mode imaging in a subsection below.

2.1 Main Principles

The ultrasound is generated by a *probe* (also called *transducer*). One typical example of a probe is illustrated in Figure 2.1. It was the work done by Pierre and Jacques Curie[2] that in 1880 led to the modern-day ultrasound

[1]Karl Theodore Dussik, born in Vienna, Austria, on Jan. 9, 1908, was a psychiatrist and neurologist at the University of Vienna

[2]Pierre Curie, born in Paris on May 15, 1859. Married with the well known Marie Curie.

probes. They discovered piezoelectricity (from the Greek word meaning "to press") whereby physical pressure applied to a crystal resulted in the creation of an electric potential. The electric charge was directly proportional to the force applied to it. They also found the reverse piezoelectric effect that occurred when a rapidly changing electric potential was applied to the crystal and caused it to vibrate. The essence of todays ultrasound transducers is that they contain piezoelectric crystals that expand and contract to interconvert electric and mechanical energy.

Figure 2.1: Picture of a LOGIQ 700 MR Transducer

The information about distant events is carried to the sensors in the probe by the reflection of the waves that the probe transmitted. The physics of the wave propagation is described bye the *wave-equation* defined by

$$\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 s}{\partial t^2}, \tag{2.1}$$

where $s = s(\vec{x}, t)$ is a scalar field, c can be interpreted as the speed of propagation, and $\vec{x} = (x, y, z)$. This equation can be determined from Maxwell's equations, which describes electromagnetic waves [4]. It can also be solved for the problem of a vibrating string [5].

2.2 "A", "B" and "M" modes of ultrasound

One example of an early ultrasound technique is the *transmission method*, where a receiver was placed on the opposite side of the specimen being imaged. In this technique one measured the amount of sound that was *not* absorbed [6]. Another method was the *pulsed reflection method*, and now the transmitter and receiver was placed on the same side of the specimen.

2.2.1 A-mode

A technique that produced a one-dimensional image was the *Amplitude* or *A-mode* ultrasound. What is being displayed is the amplitude along the

vertical axis and the time along the horizontal axis. The greater the signal that returned to the transducer, the higher the "spike".

In A-mode imaging we also talk about *range resolution*, determined by the length of the transmitted pulse T_p, which is inversely proportional to the transducer bandwidth B_w. This resolution then becomes [7]

$$\Delta_r = cT_p/2 = c/2B_w, \tag{2.2}$$

where c is the sound velocity. The sound velocity depends on the material the waves propagate through, an if we let ρ be the mass density and κ tyhe volume compressibility we get

$$c = 1/\sqrt{\rho\kappa} \tag{2.3}$$

2.2.2 B-mode

The most commonly used technique is the *Brightness* or *B-mode*. This technique produces a two-dimensional characterization of the tissue whereby each pixel on a screen represents an individual amplitude spike. These images are gray-scale images where amplitude of varying intensity are assigned shades from black to white. Figure (2.2) shows an example of a B-mode image.

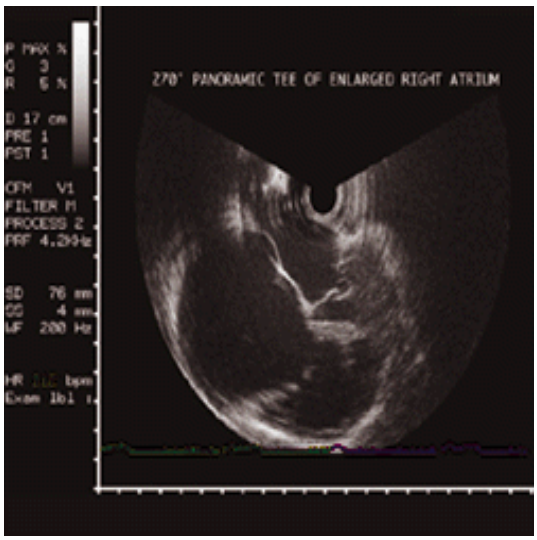

Figure 2.2: Panoramic TEE (transesophageal echocardiography, an imaging system) of enlarged right atrium.

2.2.3 M-mode

Another mode is the M-mode or *Motion-mode* imaging, which relates the amplitude of the ultrasound wave to the imaging of moving structures, such as cardiac muscle. The amplitude of the echo is displayed along a line in the object as a function of time. This is just like echo sounder.

One may also take into account the Doppler effect to image for instance the blood velocity, and also to produce color-coded images of the blood velocity[8]. When the scattered signal is moving, the frequency of the reflected signals will be altered from the transmitted frequency. It is this change in frequency that is called the *Doppler effect*. The Doppler shift, f_0, given by

$$f_d = 2f_0 \frac{v \cos\theta}{c} \tag{2.4}$$

- f_0: Frequency of transmitted ultrasound.
- c: Ultrasound wave velocity.
- v: Velocity of the scatterer (e.g. red cells).
- θ: The angle between the velocity direction and the ultrasound beam.

An important characterisic about the medium is how fast these pressure waves travel in the medium. Table 2.1 illustrates some examples. Another important problem in ultrasound imaging is the tradeoff between frequency and penetration. As we increase the frequency, we decrease the penetration depth. This also means that the image resolution is poor for organs hidden deep in the body because we have to increase the wave length to reach them.

2.2.4 Near-field and far-field

Transducers consists of array elements, and the arrays can be either linear, quadratic or circular.

When working with ultasound images, we operate in the *near-field*. This means that the wavefront of the propagating wave is perceptively curved with respect to the dimension of the array. The waves will hit the array as illustrated to the left in Figure 2.3.

In the far-field case, on the other hand, we consider the incoming waves as *plane* waves as illustrated to the right in Figure 2.3.

Biological material	Sound velocity (m/s)
Fat	1440
Kidney	1557
Muscles	1542 - 1626
Bone	2700 - 4100
Non biological material	**Sound velocity (m/s)**
Air	330
Salt water	1531
Quartz	5750
Gold	3240

Table 2.1: Sound velocity in different biological and non-biological material, found in [8].

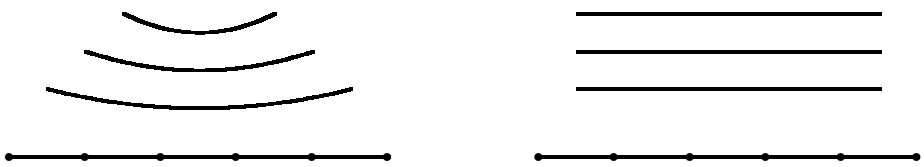

Figure 2.3: An illustration of incoming curved waves (near-field) to the left, and plane waves (far-field) to the right.

Chapter 3

Networking

In this first section I will give a brief introduction to some of the basic principals in network theory. This includes a description of how the Internet is build up, and I introduce the most important protocols which will be referred to later in this thesis.

3.1 Motivation

What distinguishes a computer network from others (like telephone- or cable network) is the *generality*. Computer networks are not optimized for a particular application, like making phonecalls or deliver television signals. They support many applications, and carry different types of data.

A network must provide a connectivity among a set of computers. At the lowest level, a network can consist of two or more computers connected by some physical medium, such as a coaxial cabel or an optical fiber. These physical media is referred to as a *link*, an the computers is referred to as *nodes*.

There are many different kinds of network technologies. Examples are Ethernet, FDDI (Fiber-Distributed Data Interface, a standard for data transmission on fiber optic lines) and ATM (Asynchronous Transfer Mode, a dedicated-connection switching technology), and the Internet is an interconnection of all the different technologies. As new networks are added to the Internet all the time, the system must scale well. This means that it is designed to support growth to an arbitrarily size. What supports this interconnection of multiple networking technologies into a single, logical internetwork is the Internet Protocol (IP), which is a protocol in the Internet Architecture. This architecture is a way of splitting the network services into layers of abstraction, and it was evolved out of experience with an earlier

packet-switched [1] network called ARPANET[1]. The Internet Architecture is illustrated in Figure 3.1.

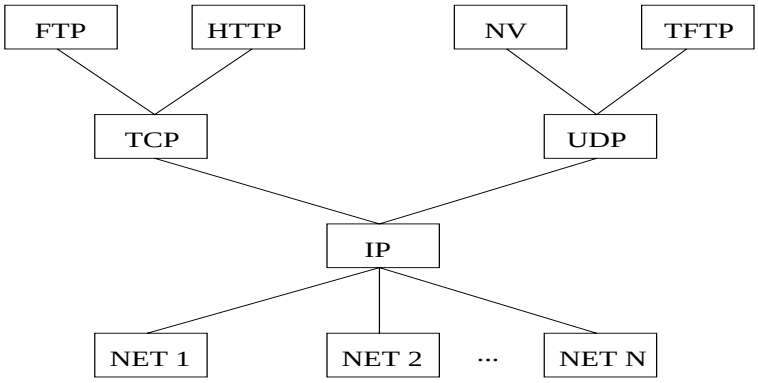

Figure 3.1: The Internet architecture.

It decomposes the problem of building a network into more manageable components and provides a more modular design. It is these abstract objects that make up the layers of a network system that are called protocols. The Internet Architecture is also referred to as TCP/IP, named after its two main protocols. IP defines the infrastructure that allows nodes and networks to function as a single logical network. The IP service model is connectionless, which means that we do not set up a connection; we just make sure that every packet contains enough information to get it to its destination. IP is a best effort model, which means that it does not guarantee that the packets will be delivered, but it will do its best. What it does NOT mean is that packets can get lost. Packets can be delivered out of order, or the same packet can be delivered twice. But if something goes wrong, the network does nothing.

3.2 UDP, RTP and RTCP

When media content is streamed to a client in real-time, the client can begin to play the stream without having to wait for the complete stream to download. Sometimes it is even impossible to download the entire stream before playing it, because the stream may not have a predefined length. In addition, when we are transmitting media across the net in real-time, we must require high network throughput. It is easier to compensate for lost

[1] A packet switched network: each node in the path to the other machine receives a complete packet, stores this packet in its internal memory, and then forwards the complete packet to the next node.

data than to compensate for large delays in receiving the data. That is, it is not that important that all the data arrive uncorrupted (or arrive at all). This is very different from accessing static data such as a file, where the most important thing is that all of the data arrive at its destination. This implies that perhaps the protocols used to transfer static data do not work well for streaming media.

3.2.1 Protocols used for streaming Media

Both the HTTP and FTP protocols are based on the TCP, a protocol designed for reliable data communications on low-bandwidth, high-error-rate networks. But TCP is above IP, and I just stated that IP is unreliable. So how can TCP guarantee that the packets will be delivered to the client? This is solved by retransmission. If a packet is missing, TCP makes sure that the packet is retransmitted. This overhead of guaranteeing reliable data transfer slows the overall transmission rate. This is why other protocols like UDP (User Datagram Protocol) are typically used for streaming media. UDP is an unreliable protocol; it does not guarantee that each packet will reach its destination. Further, there is no guarantee that the packets will arrive in the order they were sent. The receiver has to be able to compensate for lost data, duplicate packets, and packets that arrive out of order.

UDP is, just like TCP, a lower-level networking protocol, and more application-specific protocols are built on top of it. One example of such a protocol is the Real-Time Transport Protocol (RTP). RTP is the Internet standard for transporting real-time data such as audio and video, and Figure 3.2 shows the RTP architecture.

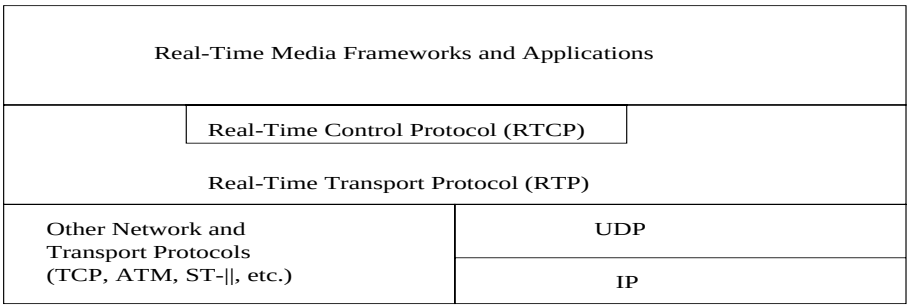

Figure 3.2: The RTP architecture.

As the figure illustrates, RTP is often used over UDP, but it is actually network and transport-protocol independent, and it provides end-to-end network delivery services for the transmission of real-time data. RTP can be

used over both unicast and multicast network sevices. In the *unicast* case, we send separate copies of the data from the source to each destination, whereas over a *multicast* network service, the data is sent from the source only once and the nework is responsible for transmitting the data to multiple locations. This last case is more efficient for many multimedia applications, such as video conferences. IP supports multicasting.

3.2.2 RTP Services

RTP enables one to identify the type of data being transmitted, determine what order the packets of data should be presented in, and synchronize media streams from different sources. As I mentioned above, RTP data packets are not guaranteed to arrive in the order they were sent - they are in fact not guaranteed to arrive at all. It is up to the receiver to reconstruct the sender's packet sequence and detect lost packets using the information provided in the packet header.

While RTP does not provide any mechanism to ensure timely delivery or provide other quality of service (QoS) guarantees, it is augmented by a control protocol (RTCP) that enables you to monitor the quality of the data distribution. RTCP also provides control and identification mechanisms for RTP transmissions.

Now, what if the quality of service is essential for a particular application. The solution is simply to use RTP over a resource reservation protocol that provides connection-oriented services.

An RTP *session* is an association among a set of applications communicating with RTP. A session is identified by a network address and a pair of ports. One port is used for the media data and the others is used for control (RTCP) data.

A *participant* is a single machine, host, or user participating in the session. Participation in a session can consist of passive reception of data (receiver), active transmission of data (sender), or both.

Each media type is transmitted in a different session. For example, if both audio and video are used in a conference, one session is used to transmit the audio data and a separate session is used to transmit the video data. This enables participants to choose which media types they want to receive. For example, someone who has a low-bandwidth network connection might only want to receive the audio portion of a conference.

3.3 Compression

As described in the previous section, data is transferred over the network as bits. Sometimes, however, it might be the case that the application program needs to send more data in a timely fashion than the bandwidth of the network supports. This problem leads to *compression*. Say, for example, that a video application have a 10-Mbps video stream that it wants to transmit, but it has only a 1-Mbps network available to it. We can then *compress* the data at the sender, then transmit them over the network, and finally *decompress* the data at the receiver.

This field has a rich history, dating back to Shannon's pioneer work on information theory in the 1940s, and in a nutshell data compression is concerned with removing redundancy from the encoding. I will in this section present briefly the most common compression techniques and standards for encoding images and video, and introduce some of the most widely used terms. When it comes to JPEG-coding, and MPEG-coding, I will look a little bit closer on some details because this is some of the most popular standards.

Generally speaking, there are two classes of compression algorithms: *lossless compression* and *lossy compression*. *Lossless* compression ensures that the data recovered from the compression/decompression process is exactly the same as the original data. *Lossy compression* on the other hand, does not promise that the data received is exactly the same as the data sent. This is because the algorithm removes information that cannot later be restored. So why would we like an algorithm that removes information from the chunk of data we would like to transmit over the network? This is because the lossless compression algorithms does not reduce the size as much as needed when we for instance are transmitting a large picture or video file. The challenge is therefore to design algorithms that take away information that will not be missed by the receiver.

3.3.1 Lossless compression algorithms

I begin by briefly introducing the three probably most known lossless compression algorithms: the *Run Length Encoding (RLE)*, the *Differential Pulse Code Modulation (DPCM)* and the *dictionary based methods*. The reason for introducing these methods is because they are often used in the lossy compression algorithms.

Run Length Encoding (RLE)

The idea behind this method is to replace consecutive occurrences of a given symbol with only one copy of the symbol, plus a count of how many times the symbol occurs; hence the name "run length". For example, the string EERHHJJJJK would be encoded as 2E1R2H4J1K.

RLE can be used to compress digital imagery by comparing adjacent pixel values and then encoding only the changes. This is quite effective for images that have large homogeneous regions. RLE is the key compression algorithm used to transmit faxes [2].

Differential Pulse Code Modulation (DPCM)

For this compression type, the basic idea is first to output a reference symbol, and then for each symbol in the data, to output the difference between that symbol and the reference symbol. For an illustration: the string AAABBCDDDD would be encoded as A0001123333 since A is the same as the reference symbol, B has a difference on 1 for the reference symbol, and so on. The benefit of this method is that if the difference between the symbols is small, one can encode them with fewer bits than the symbol itself. When the difference between the symbol and its reference symbol becomes to large, a new reference symbol is selected. By using DPCM, it has been measured compression ratios of 1.5-to-1 on digital images [2].

Dictionary based methods

The idea behind this method is to build a dictionary (table) with references to words and phrases you think might be common in a text. One can for example say that a particular word maps to a unique number, where the number will be encoded in far less bits than the word.

The question is now where this dictionary comes from. It could either be static, or dynamic which means that it is transferred in the beginning of the file.

3.3.2 Lossy compression algorithms

The compression ratio of lossless methods is not high enough for images and video compression, especially when the distribution of the pixel values is relatively flat. The next method, a lossy encoding called JPEG, uses something called *transform coding*, and it is largely based on the following observations:

- A large majority of useful image contents change relatively slowly across images, i.e., it is unusual for intensity values to alter up and down several times in a small area, for example, within an 8 x 8 block. If we translate this into the spatial frequency domain, it says that, generally, lower spatial frequency components contain more information than the high frequency components, which often correspond to less useful detail and noises.

- Psychophysical experiments suggest that humans are more receptive to the loss of higher spatial frequency components than the loss of lower frequency components.

Image Compression (JPEG)

JPEG (Joint Photographic Experts Group) compression takes place in three different stages, as shown in Figure 3.3.

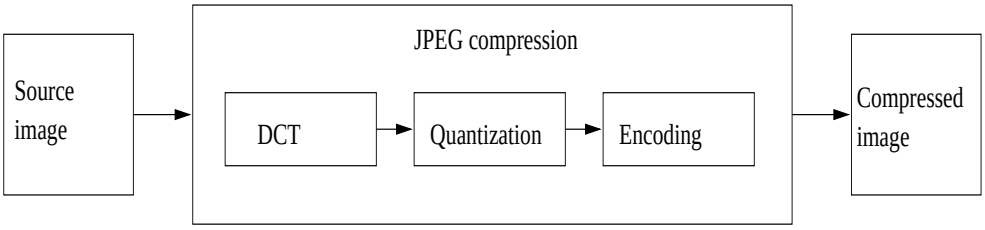

Figure 3.3: The three main steps in the JPEG encoding.

When the image is to be compressed, it is fed through these three phases one 8 x 8 block a time.

The first phase applies a Discrete Cosine Transform (DCT) to the block. This transforms the image from the spatial domain into the *spatial frequency domain*. The DCT is chosen instead of the Fast Fourier Transform (FFT) because it can approximate linear signals well with fewer coefficients. We apply DCT to separate the gross features from the fine details.

DCT along with its inverse which is performed during decompression, is defined by the following formulas:

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) cos\left[\frac{(2x+1)i\pi}{2N}\right] cos\left[\frac{(2y+1)i\pi}{2N}\right]$$

$$pixel(i,j) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)DCT(i,j)cos\left[\frac{(2x+1)i\pi}{2N}\right] cos\left[\frac{(2y+1)i\pi}{2N}\right]$$

$$C(x) = \frac{1}{\sqrt{2}} \quad \text{if} \quad x = 0, \quad \text{else} \quad 1 \quad \text{if} \quad x > 0$$

There are, of course, some loss of precision during the DCT due to the use of fixed-point arithmetic, but it is in the *quantization phase* the compression really becomes lossy. In this phase one simply drop the most insignificant bits of the frequency coefficients.

In the final phase, the *encoding phase*, we start at position $(0, 0)$ in the output matrix from the DCT phase and process the coefficients in a zigzag sequence as shown in Figure 3.4.

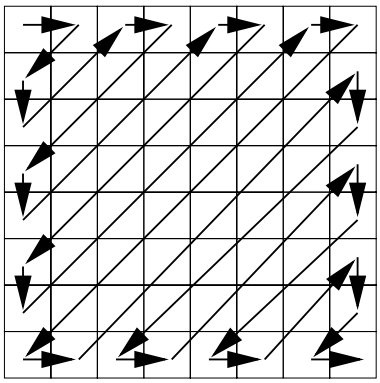

Figure 3.4: Zigzag encoding used in JPEG.

Along the zigzag, a form of RLE (Run Length Encoding) is used. Finally, the individual coefficients are encoded using a Huffman code (a minimal variable-length encoding based on the frequency of each character). Because of the different characteristics of different pictures it is impossible to say exactly what the compression ratio is, but a widely accepted generalization says that JPEG is able to compress a 24-bit color image by a ration of roughly 30-to-1.

Video Compression (MPEG)

If we do an approximation, we can say that a moving picture (i.e. video) is just a succession of still images (frames) displayed at some video rate. Each

of the pictures (frames) could be compressed using the same DCT-based technique used in JPEG. However, with this strategy we do not exploit the redundancy present between consecutive frames. MPEG (Moving Pictures Expert Group) takes this interframe redundancy into consideration.

MPEG takes as input a stream of video frames, and compress them into three different frame types: *I frames* (intrapicture), *P frames* (predicted picture) and *B frames* (bidirectional predicted picture).

I frames are also called *reference frames*. They are the JPEG compressed version of the corresponding frame in the video source. The I frames are self-contained, as opposed to the P and B frames. To be more precise; the P frames specify the difference from the previous I frame, while a B frame gives an interpolation between the previous and subsequent I or P frames. As an illustration, I show in Figure 3.5 an example found in [2] that illustrates how I, P and B frames are generated by MPEG. The seven frames in the input stream result in the specified output stream, when compressed by MPEG.

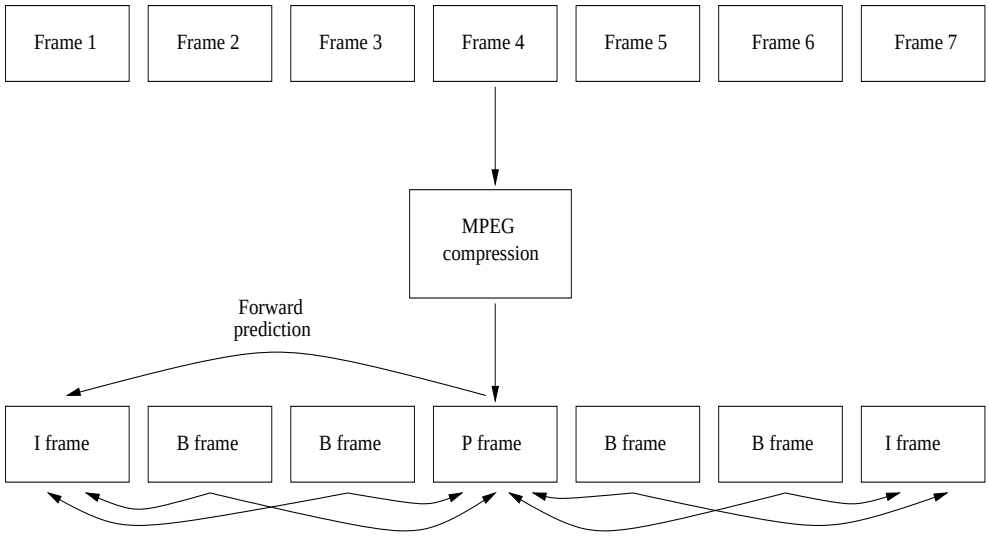

Figure 3.5: Sequence of I, P, and B frames generated by MPEG.

As mentioned above, the two I frames are self contained, and can be decompressed at the receiver end independently of the other frames. The P frames, on the contrary, depends on the preceding I frame, and can not be decoded at the receiver end if the preceding I frame is lost.

In the case of the B frame, the situation is more complicated. These frames depend on both the preceding I or P frame *and* the subsequent I or P frame. This means that both of these reference frames must arrive at the receiver before MPEG can decode the B frame to reproduce the original

video frame. These dependencies, and the need for some kind of frames to be able to decode others, implies that the compressed frames can not be transmitted in sequential order. The sequence of compressed frames in Figure 3.5, IBBPBBI, will therefore be transmitted as IPBBIBB.

It is important to note that there is no defined ratio of I frames to P and B frames, as one might expect when looking at the figure. This ratio do vary, depending on the picture quality and the compression.

MPEG coding is a very expensive task, it is rather time consuming. This is why the encoding is normally done off-line, and not in real-time. The data is therefore encoded and stored on disk ahead of time, and can then be transmitted. This may change in the future as the processor speed increases and the algorithms are getting faster. I will now take a closer look at the three different frame types.

As already mentioned, the I frames are approximately equal to the JPEG compressed version of the source frame. However, there is one main difference. MPEG works in units of 16×16 *macro blocks*. Let us say that a color video is represented by the YUV representation, where the first component, Y, represents the luminance, and the last two components, U and V, represents chrominance. Then the U and V components in each macro block are down sampled into an 8×8 block. The reason why this can be done is that the U and V components can be transmitted with less accuracy because of the fact that humans are less sensitive to color than they are to brightness. Figure 3.6 shows the relationship between the frame and the corresponding macro blocks.

From the figure, the down sampling of the U and V components into 8×8 blocks is visualized. Each of the 2×2 sub blocks in the macro block is given by one U value and one V value - the average of the four pixels. We have still got four Y values in the sub block.

The macro block processing is not typical for the I frames only, this principle is used for both the P and B frames as well. Loosely speaking, one can describe the P and B frames as carriers of information about the motion in the video. They carry for each macro block information about in what direction and how far the macro block moved relative to the reference frame(s).

Taking a closer look at the encoding of the B frame, it is important to note that the macro blocks in the B frame do not necessarily depend on both an earlier and later frame as suggested above. It may be specified to just one or another. It is in fact possible for a given macro block in a B frame to use the same intracoding as is used in an I frame. This flexibility exists because if there is a rapid change in the motion picture, it sometimes makes sense to give the intrapicture encoding rather than a forward- or backward-

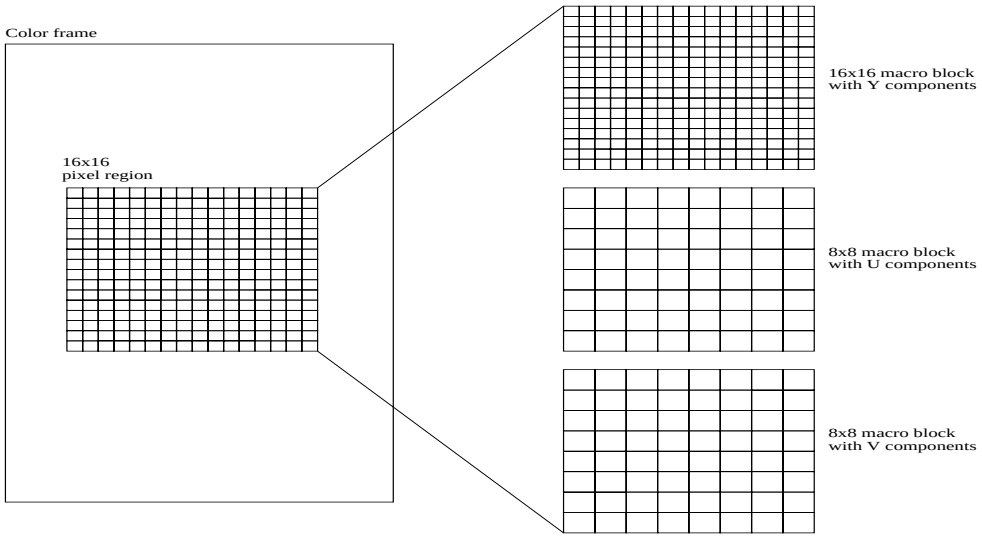

Figure 3.6: Each frame as a collection of macro blocks.

predicted encoding.

From now on I assume that the macro block uses bidirectional predictive encoding, that is, there is a dependency to both an earlier and a later frame, the macro block is represented in the following way.

- a coordinate for the macro block in the frame.
- a motion vector relative to the previous reference frame.
- a motion vector relative to the subsequent reference frame.
- a δ for each pixel in the macro block, indicating how much each pixel has changed relative to the two reference pixels.

I let F_p and F_f denote the past and future reference frames, respectively, and (x_p, y_p) and (x_f, y_f) the past and future motion vectors. To find the pixels in the macro block, we need to find the corresponding reference pixels in the past and future reference frames for every single one of them. This is done by using the two motion vectors associated with the macro block. After the average of the two reference pixels is computed, the δ for the pixel is added. If (x, y) is the coordinates of a particular pixel in the current frame, F_c, this can be stated mathematically as

$$F_c(x, y) = \frac{(F_p(x + x_p, y + y_p) + F_f(x + x_f, y + y_f))}{2} + \delta(x, y) \qquad (3.4)$$

The δ's are encoded by DCT and then quantized. The P frames are handled in the same manner, but now the frames depend on only reference frames.

3.3.3 The Java Media Framework

The Java Media Framework (JMF) is a new API, released about two years ago, which was developed to extend the Java language in the sense of meeting the increased demands for multimedia. The early features of Java only hinted at the possibilities: the audio capabilities of applets were limited to a single format, the AU-format[2] , there was no support for video, and animation was limited to series of GIF (Graphics Interchange Format) images. It was not until the introduction of the Java Media Abstract Programming Interfaces (Java Media APIs[3]), of which the Java Media Framework is one, that Java began to meet the increasing demands for multimedia. The Java Media APIs support the integration of a wide range of audio and video formats. Advanced imaging, animation, two- and three- dimensional graphics and modeling, as well as speech and telephony support, are features supported into Java applications and applets. The JMF part, that I have been concerned with in these thesis, consists of a suite of three APIs. These APIs are designed for the capture and playback of audio and video. The first of these three, which is the one that I have been working on, is the JMF Player API, describing audio and video playback. The other two APIs describe video and audio capture capabilities and video conferencing capabilities.

As with pure Java, an application written to the JMF Player API is capable of operating on any Java platform that supports a conforming implementation of the Player API. The API is still not integrated in todays web browsers, so to be able to view an applet written to the JMF Player API on the Internet, the user have to install the package on his/her computer.

Another feature, and this is the main reason why FAST wanted a multimedia streaming application in Java, is that no software installation is needed. The user do not have to install different software players to playback different file formats, it is all taken care of by the applet - downloaded in conjunction with the web page. The JMF Player API is independent of any particular data type. The Player implementation is written to support a particular type of media data, and is therefore easily integrated with the installed JMF, or, soon, JMF integrated in the web browsers. Some of the supported audio formats and video codecs supported by the current implementation of the JMF is: AIFF, AVI, MIDI, MPEG-1 video, MPEG Layer II Audio, MPEG

[2]These files uses μ-law encoding, a logarithmically 8 bits coding scheme

[3]The API is just a collection of classes that provides capabilities in areas such as audio/video playback in Java applications and applets

Layer III Audio (this is MP3), Quick Time movie and Wave. It has also got support for common protocols, which means that a JMF player can obtain media data using both the FILE, FTP (File Transfer Protocol), HTTP (HyperText Transfer Protocol) and RTP (Real-time Transport Protocol).

It is also possible to extend the player to support other media formats, which was the objective of the cooperation with FAST, and the API describes a simple mechanism to synchronize the playback of multiple media sources.

3.3.4 The Java Native Interface

Java Native Interface (JNI) Application Programming Interface (API) was introduced in release 1.1 of the Java Development Kit (JDK) and is a tool for integrating Java with native languages as C and C++, that is, make them "talk" to each other. There are several reasons why we want to do this. One reason may be that we need to perform some heavy computations that are better taken care of by C or C++, or, as in my case, we have an existing C++ class library we want to use without having to rewrite the whole library in Java.

The drawback is that we have to leave the confines of the Java Virtual Machine (JVM). The JVM executes the Java bytecode and performs system-specific operations on behalf of a Java application, and it is the virtual machine that makes it possible to "write once and run anywhere".

A Java application that uses native methods depends on the execution of code running directly on the host hardware. Java code is compiled to Java bytecode and is executed by the JVM. The JVM acts as an intermediary between the bytecode and the host hardware. Native code, on the other hand, can be thought of as running outside the JVM. It is platform-specific object code.

So, where does this native code exist? Most typically, in a dynamically loaded library (DLL) that was built using the host machine's native compiler and linker. The contents of this library are platform dependent because the object code generated by the native compiler is processor-specific. Beyond that, there may be all sorts of code in the library that references platform-specific devises or an application-specific database interface. In any case, the dynamically loadable library contains binary objects.

Once the library containing binary objects is built for a specific platform Y adhere to the JNI API, that library can be used, without rebuilding, with any JVM that supports JNI on platform Y. This means that if a vendor A and a vendor B both provide a JVM with JNI support on machine Y, then any library using only JNI calls to interface with the JVM will work with either A or B's JVM.

Chapter 4

Low-rate Speech Coding

Speech coding or *speech compression* are two names of the field concerned with obtaining compact digital representations of voice signals for the purpose of efficient transmission or storage. One has during the last decade witnessed substantial progress towards applications of low-rate speech coders both to civilian and military communications, not to mention the computer-related voice applications.

What has made this progress possible, is the development of new speech coders capable of producing high-quality speech at low data rates. As the optical fiber bandwidth in *wired* communication has become inexpensive, one could ask why we are so concerned about the preservation of low data rates. The answer is that there is a growing need for bandwidth conservation in *wireless* cellular and satellite communications. There are also voice-related applications designed for computers (like voice mail), and most of these applications require speech signal in digital format, so that it can be processed, stored, or transmitted under software control.

The digitalization gives us advantages in the form of flexibility and the opportunities for encryption, but at the same time, when uncompressed, it is associated with a high data rate and therefore high requirements of transmission bandwidth and storage. Speech-coding includes both sampling and amplitude quantization, and the objective is to represent speech with a minimum number of bits while maintaining its perceptual quality.

I will in this chapter describe some of the earlier coding techniques, and the development which leads to the technique I will be concerned with in these thesis, namely the MPEG-4 CELP algorithm, a highly effective and flexible coding technique used for low-bitrate speech coding. I will also give a brief introduction to the basic properties of speech and some historical perspectives.

4.1 Notation

Since I will be talking about discrete time signals, I will adopt the notation used in most textbooks concerned with discrete time signal processing. I will relate discrete time speech, $s(n)$, to analog speech, $s_a(t)$, by the relation

$$s(n) = s_a(nT) = s_a(t) \mid_{t=nT} \tag{4.1}$$

where T is the sampling period. Another common notation is to let lower-case symbols denote time-domain signals, and to let upper case symbols denote transform-domain signals, unless otherwise stated. When I refer to matrices and vectors, I will denote them by bold letters.

4.2 Basic Speech Properties and Historical Overview

Already more than fifty years ago speech coding research started, with the motivation of transmitting speech over low-bandwidth telegraph cables. In the earliest voice coders, or *vocoders* as I will refer to them throughout these thesis, the idea was to analyze speech in terms of pitch and spectrum and synthesize it by exciting a bank of ten analog band-pass filter, which represented the vocal tract, with periodic or random excitation. These periodic and random excitations represented voiced and unvoiced sounds respectively.

As illustrated, vocoders exploit the properties of speech. One of the problems with speech signals is that they are non-stationary. With a stationary process I mean a random process $x(n)$ where the following conditions are satisfied [7]:

1. The mean of the process is a constant, $m_x(n) = m_x$
2. The autocorrelation $r_x(k, l)$ depends only on the difference, $k - l$
3. The variance of the process is finite, $c_x(0) < \infty$

This implies that we can not analyze the signal with the conventional Fourier transform. What we *can* do, is to consider the signals as quasi-stationary over short time-segments, typically 15-20 ms.

Speech is typically classified as voiced (e.g. /a/, /i/, etc.), unvoiced (e.g. /sh/), or a combination of the two. Both has certain characteristics: voiced speech is quasi-periodic in the time domain and harmonically structured in the frequency domain, while on the other hand unvoiced speech is

random-like and broadband. In general, the voiced segments has higher energy than the unvoiced segments. Some examples are shown in Figure 4.1 and 4.2

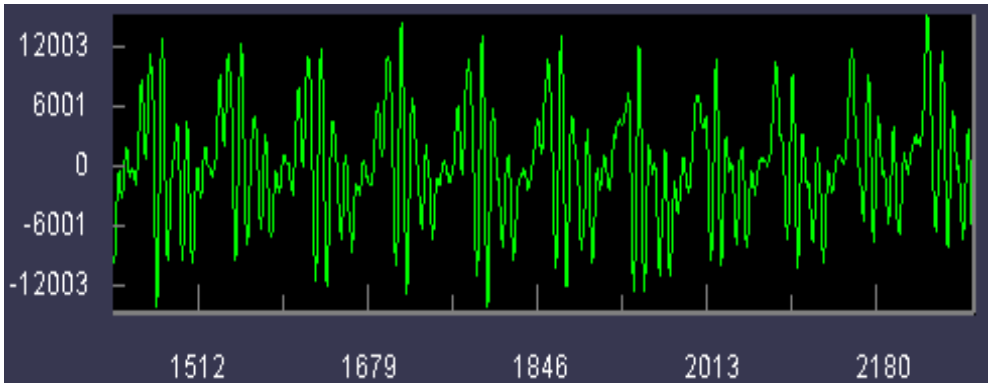

Figure 4.1: The periodicity obtained when pronouncing the voiced sound /a/.

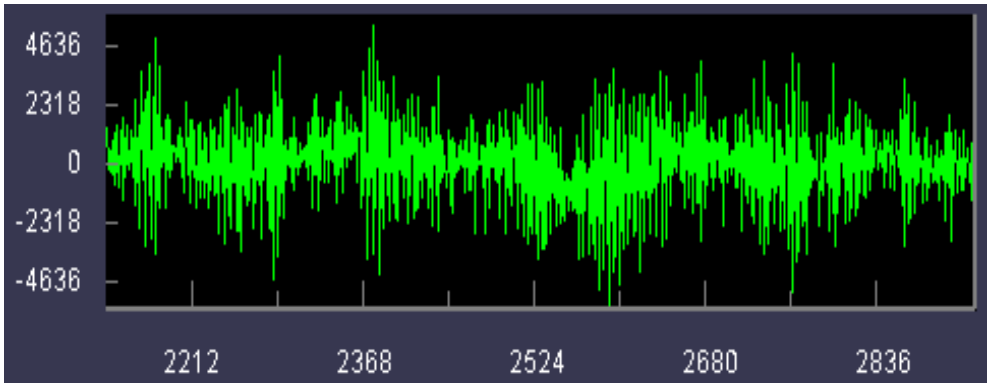

Figure 4.2: The random like time domain plot of the pronunciation of the unvoiced sound /sh/.

From these plots, we clearly see the quasi periodicity in Figure 4.1 where I have plotted a pronunciation of the vowel *a*. In Figure 4.2 on the contrary, we see the random like time domain plot of a pronunciation of the fricative (or unvoiced sound) *sh*.

The corresponding frequency domain plots of the /a/ and /sh/ sounds can be seen in Figure 4.3 and 4.4, respectively,and they both correspond to what I mentioned above.

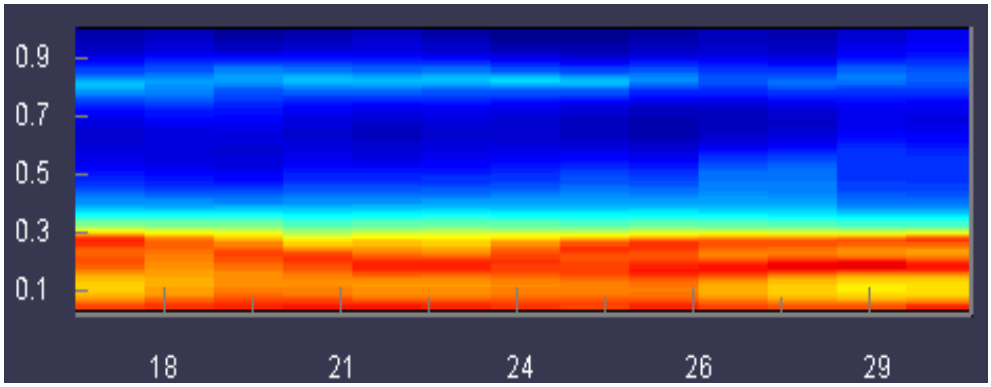

Figure 4.3: The harmonic frequency domain plot of the voiced sound /a/.

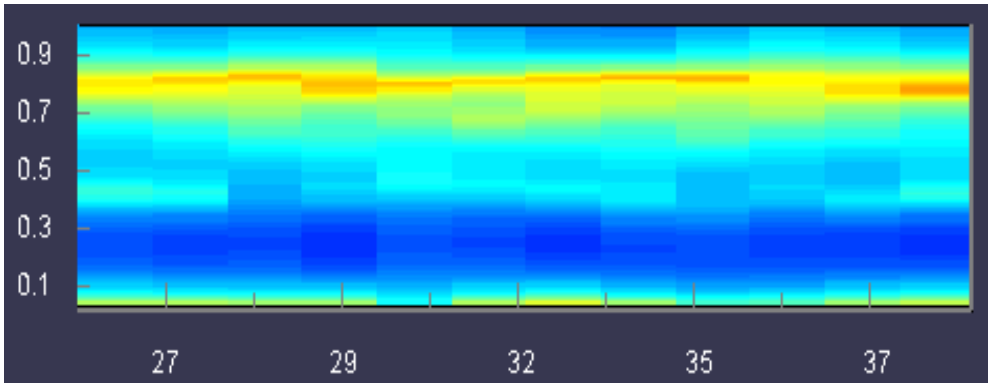

Figure 4.4: The broad band spectrum plot of the pronunciation of the unvoiced sound /sh/.

One characterize the short-time power spectrum (I will only refer to this as *spectrum* unless otherwise stated) by its *fine* and *formant* structure. The fine harmonic structure is represented by the peaks in the spectrum, and it is a consequence of the quasi-periodicity of the speech. It may be attributed to the vibrating vocal cords. We see from Figure 4.3that there are several peaks in the spectral envelope. These peaks are called *formants*, and for the average vocal tract there are between three and five formants below 5 kHz. The location and height of the formants gives us important information both in speech synthesis and perception, and wideband and unvoiced speech representations. We know that peaks in the spectrum indicates periodicity in the time domain representation of the signal. In the converse occasion; if the time domain representation consists of a single peak, we will get a flat

spectrum. This is what we see in the spectrum plot in Figure 4.4. In practical examples, we will have a combination of the two. The spectral envelope - the formant structure - is due to the interaction of the source and the vocal tract. When I refer to the vocal tract, what I mean is the pharynx and the mouth cavity.

Before I start describing the different coding techniques I will relate the properties of speech to the physical speech production system. We can classify speech sounds into three distinct classes [9] according to their mode of excitation. Starting with the *voiced* speech; the vocal cords are vibrating, generating quasi-periodic glottal air pulses that excite the vocal tract which in turn produce the voiced speech. *Unvoiced* speech or *fricative*, on the other hand, is generated by forcing air through a constriction in the vocal tract. *Plosive* sounds, like /k/, are due to abruptly releasing air pressure which was built up behind a closure in the tract. A time domain plot and a frequency domain plot of the plosive sound /p/ is shown in Figure 4.5 and 4.6

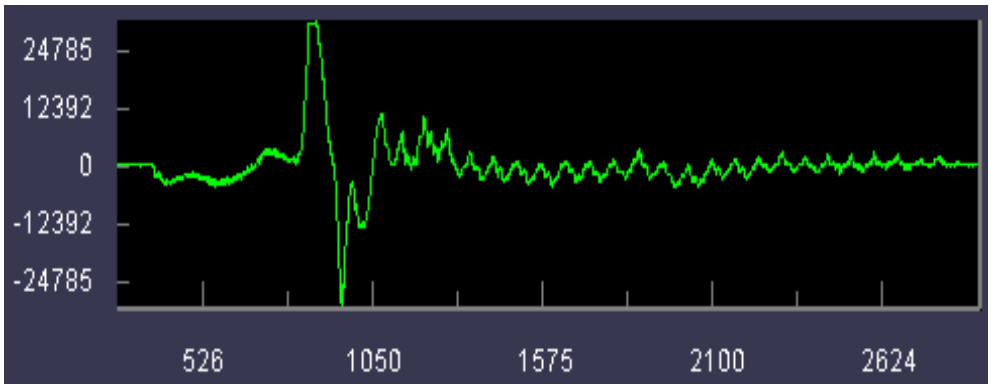

Figure 4.5: The time domain plot of the plosive sound /p/.

The sub-glottal system is composed of the lungs, bronchi and trachea. It is the source of the energy for the production of speech, which is simply the acoustic wave that is radiated from this system when air is expelled from the lungs and the resulting flow of air is perturbed by a constriction somewhere in the vocal tract.

4.3 Different Types of Vocoders

There are both parametric and non-parametric coding techniques. The simplest non-parametric coding technique is Pulse Code Modulation (PCM).

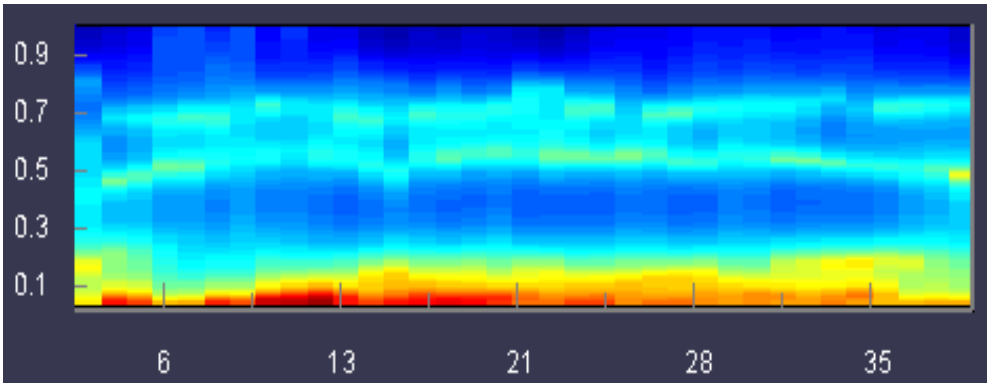

Figure 4.6: The frequency domain plot of the plosive sound /p/.

This is simply a quantization of the sampled amplitudes. It is clear that this technique alone will not give us the low data rates we are seeking.

To obtain speech coding at the medium-rates and below, what we need is to use an *analysis-synthesis* technique. This is a parametric approach. Briefly described; the speech is, in the analysis stage, represented by a compact set of parameters which are encoded efficiently. Then, in the synthesis stage, these parameters are decoded and used in conjunction with a reconstruction mechanism to form speech. This is the idea that forms the basis in the CELP algorithm, but here they use *analysis-by-synthesis* in the analysis stage. That is, the parameters are extracted and encoded by minimizing explicitly a measure of the difference between the original and the reconstructed speech. The measure is usually a mean square measure.

I mentioned the PCM method as a straight forward method for discrete time, discrete amplitude approximation of analog waveforms - but it does not take into account the redundancy in the signal. As digital computers became more powerful, and with the flexibility they offered, one could experiment with more sophisticated digital representations of speech.

The development has gone through several stages, from the simple speech source-system production model depicted in Figure 4.7, where a slowly time-varying system is excited by periodic impulse train for the voiced speech, and by random excitation for unvoiced speech, to the more complex linear prediction algorithm with stochastic vector excitation called "Code Excited Linear Prediction". The vocal tract filter in figure (4.7) is an all-pole filter, an AR-model. The parameters are obtained by Linear Prediction. This is a process where the present speech sample is predicted by the linear combination of previous samples. I will return to AR-models and Linear Prediction in a later chapter.

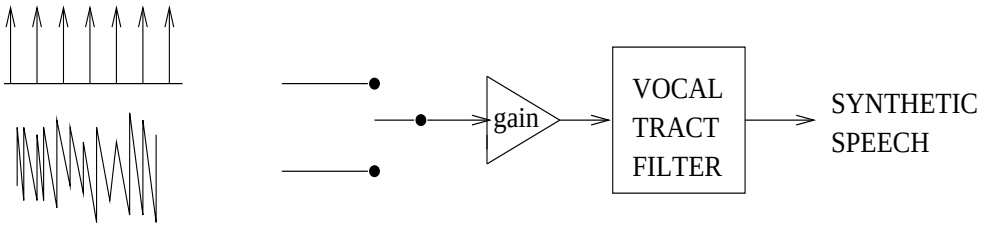

Figure 4.7: The engineering model for speech synthesis.

During the last decades, techniques like Short Time Fourier Transform (STFT), transform coding, and sub-band coding has also been exploited in the analysis - synthesis process. CELP is also based on LPC, but it also use an efficient technique to encode the LPC parameters, namely *vector quantization*.

4.4 How to Measure the Performance

Before one can rank the different algorithms and techniques, one needs to know how the performance is measured. The typical way to evaluate the different speech coding algorithms is based on the bitrate, the quality of reconstructed speech, the complexity of the algorithm, the delay introduced, and the robustness of the algorithm to channel errors and acoustic interference.

These are the criteria one have in mind, and the next important task is how to gauge the speech quality. This is not a simple task. One of the most common objective measures, not only in speech processing algorithms, but in all signal processing algorithms, is the signal-to-noise (SNR) ratio, defined by the following equation [10]

$$\mathrm{SNR} = 10 \log_{10} \left\{ \frac{\sum_{n=0}^{M} s^2(n)}{\sum_{n=0}^{M} (s(n) - \hat{s}(n))^2} \right\}, \tag{4.2}$$

where M is the number of samples, $s(n)$ is the original speech data, while $\hat{s}(n)$ is the coded speech data.

This tends to be a satisfying measure, but there is a minor problem: this is a long term measure, so it will "hide" temporal reconstruction noise. These temporal variations can be better measured using a short-time signal-to-noise ratio - that is, we compute the SNR for each N-point segment of speech. The segmental SNR (SEGSNR) is given by

$$\text{SEGSNR} = \frac{10}{L}\sum_{i=0}^{L-1}\log_{10}\left\{\frac{\sum_{n=0}^{N-1}s^2(iN+n)}{\sum_{n=0}^{N-1}(s(iN+n)-\hat{s}(iN+n))^2}\right\} \tag{4.3}$$

L is the number of N-point sequences we have split the signal into. What we should note here, is that the averaging occur *after* the logarithm. This implies that coders with variant performance will be more penalized.

The SNR is the most common measure, but not the only one. Other methods mentioned in the literature are the articulation index, the log spectral distance (a number that allows two measurement vectors to be compared), and the Euclidean distance (the square root of the sum of the squares of the entries of a vector).

4.5 Physical Aspects of Speech Modeling

The simplest physical configuration that has a useful interpretation in terms of the speech production is depicted in Figure 4.8 and was found in [9]

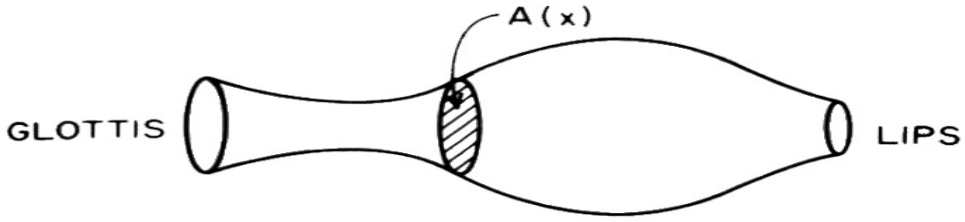

Figure 4.8: A simplified vocal tract.

Here, the vocal tract is modeled as a tube of non-uniform, time-varying, cross-section. As already mentioned, sound is almost synonymous with vibration. The sound - waves are created by vibration and they propagate through air. So, what is the fundamental in describing this generation and propagation of sound in the vocal system is the laws of physics, and in particular the laws of conservation of mass, conservation of momentum and conservation of energy, just to mention a few.

Describing the motion of air in the vocal system leads to a set of partial differential equations which formulation and solution is extremely difficult unless we do some very simple assumptions about vocal tract shape and energy losses in the vocal system. This is what we do in figure (4.8). What we do is to assume that the waves are *plane waves* (see Section 2.2.4) propagating

along the axis of the tube. This is a reasonable assumption if we assume the wavelengths are less than about 4000Hz, that is - the wavelengths are long compared to the dimension of the vocal tract. If we in addition assume that there are no losses due to viscosity or thermal conduction, and further assume the laws of conservation of mass, momentum and energy - it has been shown that the sound waves in the tube satisfy the following equations:

$$-\frac{\partial p}{\partial x} = \rho\frac{\partial(u/A)}{\partial t} \tag{4.4}$$

$$-\frac{\partial u}{\partial x} = \frac{1}{\rho c^2}\frac{\partial(pA)}{\partial t} + \frac{\partial A}{\partial t} \tag{4.5}$$

4.6 Auto Regressive Moving Average Models

When I later describe the theory behind linear predictive coding (LPC) and code excited linear prediction (CELP) [1], both central in the discussion of the MPEG4 coder, one will descover that these techniques are based on signal modeling using so-called auto regressive models (AR-models). AR-models are a special case of the more general auto regressive moving average models (ARMA-models).

4.6.1 ARMA-models

The idea is to design a filter, that, excited by a special sequence $v(n)$, will generate an approximation, $\hat{x}(n)$, of our original signal $x(n)$. The filters we use for this purpose are causal, linear, shift-invariant filters having a rational system function with p poles and q zeros. These are called ARMA-models, and are defined by the system function

$$H(z) = \frac{B_q(z)}{A_p(z)} = \frac{\sum_{k=0}^{q} b_q(k)z^{-k}}{1 + \sum_{k=1}^{p} a_p(k)z^{-k}} \tag{4.6}$$

Let $v(n)$ be white noise with power spectrum $P_v(z) = \sigma_v^2$. If we assume that $H(z)$ in equation (4.6) is stable, and filter the sequence $v(n)$ with this filter, the *power spectrum* of the output process $x(n)$ will be

$$P_x(z) = \sigma_v^2\frac{B_q(z)B_q^*(1/z^*)}{A_p(z)A_p^*(1/z^*)}, \tag{4.7}$$

[1] To be precise, CELP is an extension of LPC.

where the superscript * denotes the complex conjugate. An ARMA-*process* is a process having a power spectrum of the form of equation (4.7). In speech processing applications we excite $H(z)$ by white noise to produce unvoiced speech, and by an impulse train to produce voiced speech.

$x(n)$ and $v(n)$ are related by the following linear constant coefficient difference equation

$$x(n) + \sum_{l}^{p} a_p(l)x(n-l) = \sum_{l}^{q} b_q(l)v(n-l) \tag{4.8}$$

The autocorrelation of $x(n)$ and the cross correlation between $x(n)$ and $v(n)$ satisfy the same difference equation, so we get

$$r_x(k) + \sum_{l}^{p} a_p(l)r_x(k-l) = \sum_{l}^{q} b_q(l)r_{vx}(k-l) \tag{4.9}$$

4.6.2 Autoregressive Processes

As mentioned, an AR-model is a special case of an ARMA(p,q)-model, and it results when $q = 0$ in equation (4.6), that is, when $B_q(z) = b(0)$. Now $x(n)$ is generated by filtering a process (e.g. an impulse train or white noise) with an all-pole filter of the form

$$H(z) = \frac{b(0)}{1 + \sum_{k=1}^{p} a_p(k)z^{-k}} \tag{4.10}$$

I will refer to an autoregressive process of order p as an AR(p) process.

In general; in practical use we almost always use all-pole models. This is because these models have been found to provide sufficient accuracy representation for many types of signal in many different applications. In speech processing, the acoustic tube model for speech production leads to an all-pole model for speech [9].

4.7 The WAV file format

The input files to the speech analyzer application are in the WAV format, that is, the files have got a .wav postfix. In a WAV file, the samples are coded using pulse code modulation, see section 4.3. A WAV file is made up of three blocks, a RIFF block, a FMT block, and a DATA block, as illustrated in Table 4.1. For an 8 bit mono sample, each byte represents one sample. For a stereo sample, one byte will be the left channel, and the next the right

channel. For a 16 bit sample it is the same, but now each sample will be two bytes long, rather than just one byte. The data are in little-endian format, which means that the *least significant byte* is stored in the lowest memory address, which is the address of the data. The data in my experiments are in the 16 bit mono format.

In addition to this, there are a few things one should be aware of. A sample stored using one byte per sample, and a sample stored using two bytes per sample are stored in different variables. In the 8 bit case, the samples are unsigned, but in the 16 bit case they are signed.

RIFF block	12 bytes long
bytes	
0-3	"RIFF"
4-7	length of the block
8-12	"WAVE"
FMT block	24 bytes long
bytes	
0-3	"FMT"
4-7	length of the formant block (0x10)
8-9	0 = mono, 1 = stereo
10-11	channel numbers
12-15	sample rate (in Hz)
16-19	bytes per second
20-21	bytes per sample: 1 = 8 bit mono, 2 = 8 bit stereo or 16 bit mono, 4 = 16 bit stereo.
22-23	bits per sample
DATA block	Data: variable length
bytes	
0-3	"data"
4-7	length of chunk
8-end	sound data

Table 4.1: The WAV file format.

4.8 Linear Predictive Coding and Transmission of LP-parameters

When we sample a speech signal and want to transmit it to a receiver, what seems to be the most obvious thing to do is to quantize the samples, put them on the link, and transmit them to the receiver. As we have seen, this is very inefficient - we get a lot of redundancy. As mentioned, in the CELP coder the samples are coded using linear predictive coding (LPC), and here the a-parameters of the filter, or LPC-coefficients, are what is being transmitted over the link. These are the coefficients of the filter that will be excited and then produce the speech that is determined by linear prediction.

4.8.1 Linear Predictive Coding (LPC)

Linear predictive coding is used in estimating basic speech parameters, and in representing speech at low bit-rates. The idea behind the method is that the present speech sample could be approximated by a linear combination of earlier samples. Speech can be modeled by a linear, time-varying model, which is either driven by quasi periodic impulse trains or white noise. LP is a robust method for estimating the filter parameters of this linear time-varying system. When the prediction coefficients have been found, the system is uniquely determined in such a way that it can be modeled as a linear all-pole system.

For a p-th order forward linear predictor the present sample of the speech sample is predicted by the p past samples,

$$\hat{s}^f(n) = \sum_{i=1}^{p} a_i^f s(n-i), \tag{4.11}$$

where the subscript ^{f} indicates that it is a forward prediction. The system function is given by

$$P(z) = \sum_{k=1}^{p} a_k z^{-k} \tag{4.12}$$

The prediction parameter, a_i^f, are found by minimizing the least square error of the forward prediction, ϵ^f

$$\frac{\partial \epsilon^f}{\partial a_i^f} = 0, \quad i = 1, 2, \ldots, p, \tag{4.13}$$

where $\epsilon^f = E[(e^f(n))^2] = E[(s(n) - \hat{s}^f(n))^2]$. There are different numerical methods for solving least sqares problems, for example by using singular value decomposition [11].

The minimization gives us a set of Toeplitz equations,

$$r_{ss}(m) - \sum_{i=1}^{p} a_i^f r_{ss}(m-i) = 0, \tag{4.14}$$

where

$$r_{ss}(m) = E[s(n+m)s(n)], \tag{4.15}$$

and m indicates the lag. This gives us the following modeling error:

$$\epsilon_p = r_{ss}(0) + \sum_{i=1}^{p} a_i r_{ss}^*(i) \tag{4.16}$$

The autocorrelation can either be estimated by the following *biased* estimator

$$\hat{r}_{ss}(m) = \frac{1}{N-|m|} \sum_{i=0}^{N-|m|-1} s(n+|m|)s(n) \tag{4.17}$$

or by the following *unbiased* estimator

$$\hat{r}_{ss}(m) = \frac{1}{N} \sum_{i=0}^{N-|m|-1} s(n+|m|)s(n) \tag{4.18}$$

The system in equation (4.14) can be efficiently solved by using the Durbin recursion, which gives a recursive updating of the coefficients. The algorithm is illustrated in Table (4.2). In fact, the speech coding algorithm used in the GSM system is an extended version of LPC, the so-called Regular Pulse Excited-Linear Predictive Coder (RPE-LPC) [12] with a Long Term Predictor loop. Long term prediction will be further discussed in Section 4.9.1.

Once we have found the LP parameters, the major issue in LPC is the quantization of these coefficients. Direct quantization of the a-coefficients introduce quantization errors that may lead to instability of the synthesis filter. This approach is usually avoided.

There are other possible solutions to the problem. For example we could exploit the zeros of $(1-A(z))$, where $A(z) = \sum_{i=1}^{p} a_i z^{-i}$, so that the stability of the synthesis filter is ensured. The drawback is that these zeros are difficult

The Durbin Recursion

1. Initialize the recursion
   a) $\epsilon^f(0) = r_{ss}(0)$
2. **for** $m = 1, 2, \ldots, p$
   a) $\gamma_m = r_{ss}(m) - \sum_{i=1}^{m-1} r_{ss}(m-i)$
   b) $\Gamma_m = \gamma_m / \epsilon^f(m-1)$
   c) $a_m(m) = \Gamma_m$
   d) **for** $i = 1, 2, \ldots, m-1$
      $a_i(m) = a_i(m-1) - \Gamma a_{m-i}(m-1)$
   e) $\epsilon^f(m) = (1 - \Gamma_m^2)\epsilon^f(m-1)$

Table 4.2: The table illustrates the main steps in the Durbin recursion.

to compute and they do not form ordered sets of parameters. This, again, makes it difficult to develop statistical patterns for efficient quantization.

Another approach is to use the reflection coefficients which *are* an ordered set of parameters. The stability is ensured by coding the coefficients at discrete levels within the range of -1 and 1. These coefficients, or more precisely, *the transformation of* the reflection coefficients, are what is being used in the log area ratios (LARs).

4.8.2 Log Area Ratios (LARs)

As mentioned above, the log area ratio is a transformation/conversion of the reflection coefficients. The LARs are given by

$$\text{LAR}(m) = log\left\{\frac{1+k_m}{1-k_m}\right\}, \tag{4.19}$$

where k_m indicates the m-th reflection coefficient, and $|k_m| < 1, \quad m = 1, 2, \ldots, p$.

So, what is the benefit of this transformation? First, as already mentioned, the reflection coefficients are a ordered set of parameters, and the transformation will not alter this. This ordering can be further exploited, since we can encode the first few reflection coefficients with higher precision. At last, and perhaps the most important one, is that the transformation leads to a set of parameters that are less sensitive to quantization.

4.8.3 Line Spectral Pairs (LSPs)

Another widely used method for representing the LP parameters is the Line Spectrum Pairs (LSPs). LSP coefficients were introduced in the 1980's as an alternative to the filter coefficients $a_p(k)$ and the reflection coefficients Γ_j in the representation of an all-pole model for a signal $x(n)$.

To explain this method, we look at a 10-th order polynomial,

$$A_{10}(z) = 1 + a_1 z^{-1} + \cdots + a_{10} z^{-10} \tag{4.20}$$

We now look at the two polynomials

$$A_{10_1}(z) = A_{10}(z) + z^{-11} A_{10}(z^{-1}) \tag{4.21}$$

and

$$A_{10_2}(z) = A_{10}(z) - z^{-11} A_{10}(z^{-1}) \tag{4.22}$$

$A_{10_1}(z)$ and $A_{10_2}(z)$ each have a set of five complex conjugate pairs that lie on the unit circle. As a result of this, the angles of these zeros uniquely define the line sprectrum pairs, and, thus, the prediction error filter $A_p(z)$.

4.8.4 Cepstral coefficients

One useful representation of the the filterparameters a_p of the filter $H(z)$ is by cepstral coefficients. The cepstral coefficients can be derived directly form the filter coefficients by the following relation:

$$c_k = a_k + \frac{1}{k} \sum_{i=1}^{k-1} i c_i a_{k-i}, \quad 1 \leq k \leq p, \tag{4.23}$$

where a_j indicates the LPC coefficients. The cepstral coefficients, that can be interpreted as the coefficients of the Fourier transform representation of the log magnitude spectrum, has shown to be a robust and reliable feature set for speech recognition. One feature of the cepstrum coefficients is that they can be used to distinguish between voiced and unvoiced speech. If we plot the cepstrum coefficients for voiced speech, we will typically see peaks in the cepstrum at regular positions. These distances between the peaks represents the pitch period. In figure (4.9) we see the cepstral plot of a male speaker pronouncing the vowel /a/. One clearly sees the pitches at about every 10th millisecond.

On the contrary, for unvoiced speech we will not see this peak periodicity as a result of the random nature of the input signal. An illustration is shown

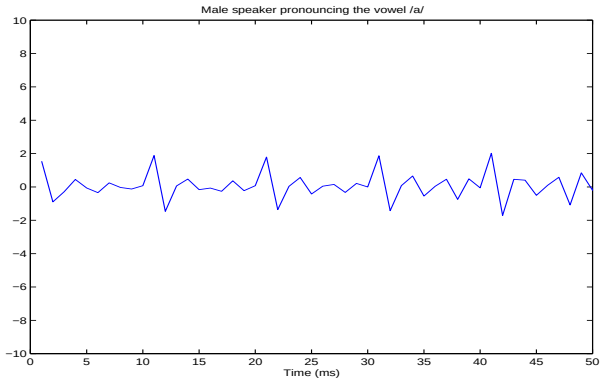

Figure 4.9: Cepstral plot of a male speaker pronouncing the vowel /a/.

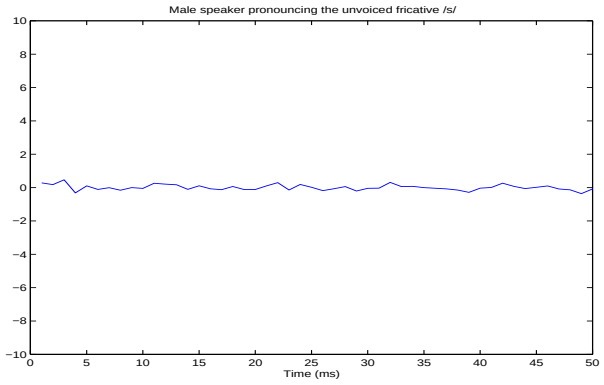

Figure 4.10: Cepstral plot of a male speaker pronouncing the unvoiced fricative /s/.

in figure (4.10). As we can see in the figure, no strong peaks appear in the spectrum.

Estimation of the pitch period is one application where cepstrum analysis has been applied with success. Other applications, that I will return to in later sections, are speaker verification and speaker identification.

4.9 CELP coders

The CELP coder is an extension of the LPC coder just described. The LPC codecs give a fairly good result at low bit rates (2.4 - 4.8 kbits/s), but to extend the result we have to consider some more complex techniques.

4.9.1 The conventional CELP coder

In its simplest form, the synthesis section of a linear predictive vocoder can be illustrated as Figure (4.11) below, where the filter parameters are determined by linear prediction.

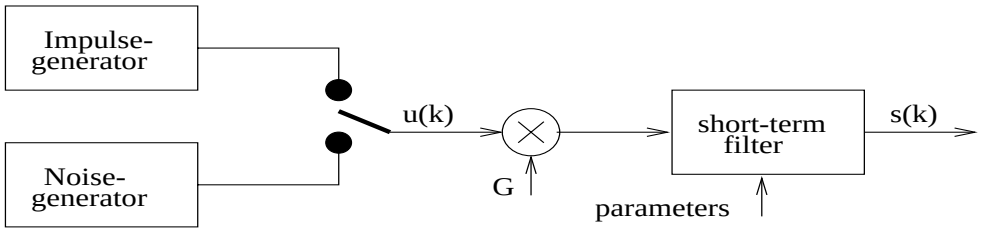

Figure 4.11: Synthesis section of a simple vocoder.

The synthesis filter may also include a pitch filter to model the long-term periodicities present in voiced speech. This pitch filter is not so important in LPC as it is in CELP coding.

As we can see in the figure, the filter is either excited by an impulse train, or by white noise. To achieve low-rate high-quality speech coding, what is required is a more efficient representation of the excitation sequence. This is the main difference between LPC and CELP. A CELP coder is shown in Figure (4.12).

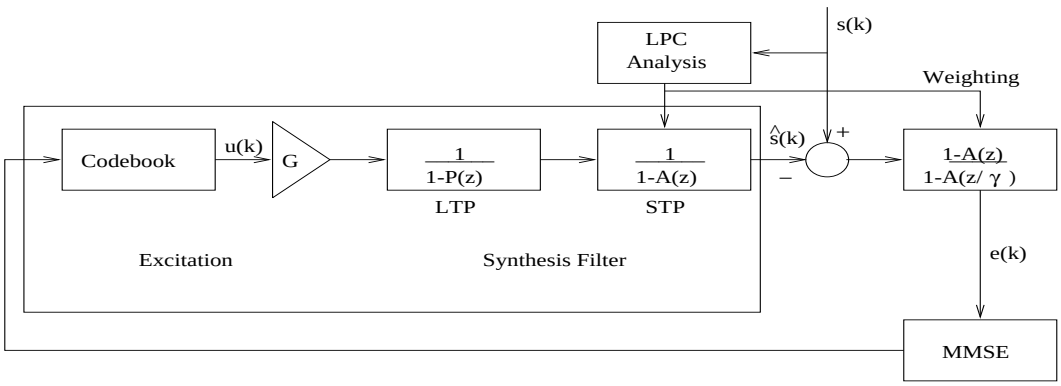

Figure 4.12: CELP encoder. The capsulated region equals the CELP decoder.

The coder works as follows: the speech signal is split into segments (also called frames) of typically 20 ms. These samples are then used in a linear

prediction analysis to determine the coefficients, $a_p(i)$, of the short term prediction filter. The long-term prediction coefficients , $p_j(i)$, are determined from the short-term prediction residual using an open loop configuration. The codebook contains in most cases 1024 vectors of Gaussian sequences. These vectors, after being scaled by a gain factor G, are used to drive the filters. The synthetic speech segment that results is compared to the original speech segment, the error is weighted and then evaluated in a minimum mean square error (MMSE) sense, and we try all the different codebook-vectors until we find the one that gives the minimum error.

I will now take a closer look at the coder, and explain some of the ideas behind the three filters that are used.

The reason for the weighting filter $W(z)$, defined by

$$W(z) = \frac{1 - A(z)}{1 - A(z/\gamma)} = \frac{1 - \sum_{i=1}^{p} a_p(i) z^{-i}}{1 - \sum_{i=1}^{p} \gamma^i a_p(i) z^{-i}}, \tag{4.24}$$

is to shape the spectrum of the reconstruction error so that the quantization noise is masked by the high energy formants. This exploits the properties of the human ear. The role of the γ, which is 0.8 in the MPEG4-CELP coder, is to de-emphasize the error energy in the formant regions.

The long-term prediction filter, which is a AR-filter (see Section 4.6.2) is of the form

$$H_{LTP} = \frac{1}{1 - A_L(z)} = \frac{1}{1 - \sum_{i=-j}^{j} a_j(i) z^{-i-\tau}}, \tag{4.25}$$

where j is a small integer, and the lag (also called pitch period), τ can be determined by finding the maximum of the autocorrelation sequence ($\hat{r}_{ee}(m)$) of the prediction residual. This filter serves to represent the pitch (fine) structure of speech.

The short-time prediction filter, on the other hand, is given by

$$H_{STP} = \frac{1}{1 - A(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_p(i) z^{-i}}, \tag{4.26}$$

and represents the formant structure of speech. The parameter updating rate varies for the two filters. The long-term predictor is updated at high rates, typically between 60 to 200 times per second, while the short-time predictor is updated at much lower rates - about 30 to 100 times per second. The use of adaptive long-term prediction in addition to short-time prediction provides additional coding gain, but at the expense of higher complexity.

The question left is how the codebook vectors are formed. It is here, in the design of the codebooks, and the search through them, that most researching

has been concentrated, and a lot of improvements have been done. There are many different CELP algorithms, but the basic principles as I just described are the same. Though there are different ways to implement the coder.

Obviously, the codebook must reside in the memory of both the transmitter and the receiver. When we have found the excitation vector from the codebook that minimize the error between the original speech frame and the synthesized speech frame, we transmit the *index* of the vector. This index is used to look up the right excitation vector at the receiver end. Let us say that the codebook consists of L code vectors, $\{\hat{\mathbf{s}}_n = [\hat{s}_n(0)\hat{s}_n(0)\cdots\hat{s}_n(N-1)]^T\}$, that is, L $N \times 1$ real valued vectors. These are designed by dividing the vector space into L non-overlapping cells, C_n. To illustrate this point, I have used a figure found in [13]. The dot in the center of each frame C_n (see Figure (4.13)) represents a template vector $\hat{\mathbf{s}}_n$. As the figure tries to illustrate, each cell contains more than just the vector $\hat{\mathbf{s}}_n$, but $\hat{\mathbf{s}}_n$ represents the *centroid* of the cell. For coders like Vector PCM (VPCM) we have an incoming signal $\mathbf{s}_n$, and if $\mathbf{s}_n$ belongs to C_n it will be represented by $\hat{\mathbf{s}}_n$ and the symbol u_n (the *channel symbol*) will be transmitted to the receiver. In CELP coding we try all the vectors in the codebook to see which of the excitation signals that will help us to produce the best approximation to the original speech frame. In CELP, these excitation vectors are Gaussian sequences.

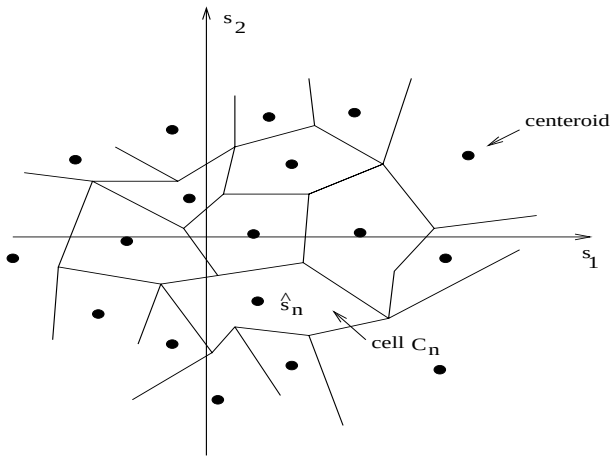

Figure 4.13: Cells for two-dimensional Vector Quantization (VQ).

The process of designing the codebook can be either fixed or adaptive. In the case of adaptive codebooks, they are designed a priori, and roughly the design procedure involves an initial guess for the codebook and then iterative improvement by using a large number of training vectors. In the

fixed codebook case, the vectors are sequences of white Gaussian noise with unit variance.

As one can imagine, one of the main disadvantages of the original CELP algorithm was the large computational effort for the codebook search. This is a problem that motivated a great deal of work, mainly focused upon developing structured codebooks and fast search procedures. One method, proposed by Murray and Fagan [14], is to use a sparse overlapping codebook structure that will take advantage of the architecture of a DSP coder implementation to reduce the complexity of the cross-correlation term and the energy term in the codebook search.

In the following table I will summarize the performance of some compression techniques [13], [15], [16].

Algorithm	**Bit Rate (kbits/s)**	**MOS**	**MIPS**
PCM	64	4.3	0.01
ADPCM	32	4.1	2
LD-CELP	16	4.0	19
VSELP	8	3.45	13.5
CELP	4.8	3.2	16
IMBE	4.15	3.4	3

Table 4.3: A summary of performance for some coders.

ADPCM means Adaptive Differential Pulse Code Modellation, and was adopted for the G.721 CCITT standard. It uses a pole-zero adaptive predictor. LD-CELP is a short for Low Delay-CELP and is associated with the ITU G.728 standard. This coder uses a backward-adaptive predictor and short excitation vectors to achieve low one-way delay. VSELP, or Vector Sum Excited Linear Prediction is an 8 kbits/s algorithm and was adopted for the North American Digital Cellular System. It is derived by combining excitation vectors form three codebooks: one pitch-adaptive and two highly structured stochastic codebooks. Improved Multi-Band Exciter (IMBE) treats the short-time speech spectrum as the product of an excitation spectrum and a vocal tract envelope. The excitation spectrum is modeled by a combination of harmonic and random-like contributions, and exploits the fact that the spectra of mixed sounds or noisy speech contain both voiced (harmonic) and unvoiced (random-like) regions.

The MOS (Mean Opinion Score) is a subjective measure of quality. The listeners are "calibrated" in the sense that they are familiarized with the listening conditions and the range of speech quality they will encounter.

Ratings are obtained by averaging numerical scores over several hundreds of speech records. Table 4.4 lists the MOS scale. It usually involves 12 to 24 listeners.

MOS Scale	Speech Quality
1	Bad
2	Poor
3	Fair
4	Good
5	Excellent

Table 4.4: The MOS scale.

4.9.2 Main Differences Between the Original CELP and the MPEG4-CELP

For both coders the basic technology is the same. What makes them different, and gives the MPEG4-CELP an advantage, is the *flexibility* if offers. While in conventional CELP schemes one is only offered compression at a single bit rate, targeted at a specific application. Within MPEG4, on the other hand, this high quality compression is just one of many features. It is possible to address many various applications from one basic coding scheme, generation of arbitrary bit rates are supported, so is bit rate scalability and decoder complexity scalability.

In the following sections I will give an introduction to the most important functionalities offered, and mention the less important ones.

Multiple Bit Rates

In the MPEG4 standard, two sampling rates are supported, namely 8 and 16 kHz. For both sampling rates there are associated certain bandwidths; 200-3400 Hz for the 8 kHz sampling rate, and 50-7000 Hz for the 16 kHz sampling rate. In addition, both fixed and variable bit rates are supported. The possible bit rates fall into two categories depending on the type of applied LPC quantizer. Starting with the 8 kHz sampling rate, Table 4.5 shows the fixed bit rates that are supported [17].

In the case of 16 kHz sampling rate, less fixed bit rates are supported as illustrated in Tabel 4.6 [17].

It is then possible to encode at any of the bit rates listed in Table 4.5 and 4.6 if a variable bit rate operation is requested.

Bit rates for Scalar Quantizer (bit/s)	Bit rates for Vector Quantizer (bit/s)
4325, 4725, 5125, 5534, 5834, 6134, 6650, 6950, 7250, 7550, 7850, 8050, 8250, 8650, 9250, 9650, 10050, 10450, 10850, 11250, 11450, 11650, 12900, 13300, 13700, 13900, 14100	3850, 4250, 4650, 4900, 5200, 5500, 5700, 6000, 6300, 6600, 6900, 7100, 7300, 7700, 8300, 8700, 9100, 9500, 9900, 10300, 10500, 10700, 11000, 11400, 11800, 12000, 12200

Table 4.5: Fixed bit rates supported for speech sampled at 8 kHz.

Bit rates for Scalar Quantizer (bit/s)	Bit rates for Vector Quantizer (bit/s)
13667, 15867, 18200, 20133, 24000	13267, 15067, 17000, 19333, 23200

Table 4.6: Fixed bit rates supported for speech sampled at 16 kHz.

Bit Rate Scalability

Bit rate scalability means that we are given the opportunity to transmit a subset of the bit stream and still decode the bit stream with the same decoder. This is provided, in the case of 8 kHz sampling rate, by adding *enhancement layers*. In this case there is the possibility of adding three such layers - each with a step of 2000 bit/s. The layers must be combined with a bit rate chosen from Table 4.5. It is important to note that the three enhancement layers only yield the 8 kHz sampling rate case.

Bandwidth Scalability

The definition of bandwidth scalability is the possibility of changing the bandwidth of the signal during transmission. To enable this bandwidth scalability to cover both the sampling rates, a bandwidth extension tool has been incorporated to the CELP coder. This tool is optional, and only supported in the 8 kHz sampling rate mode. If 16 kHz sampling rate is required we may add this tool. The bandwidth extension tool adds a layer of scalability. One thing that is important to remember, is not to confuse this tool with the default 16 kHz sampling rate mode. If no bandwidth scalability is required, it is recommended to use the 16 kHz sampling rate mode, but both configurations, the 8 kHz sampling rate mode with bandwidth scalability and the

16 kHz sampling rate coder, increase intelligibility and naturalness of the decoded speech.

Complexity Scalability

If we use the 16 kHz sampling rate mode we have the opportunity to run the decoder at various *complexity levels*, which means that we may decode the bit stream with a lower complexity decoder. This is not offered at the 8 kHz sampling rate mode. At the time this is written, there are defined three levels of complexity. Two of the levels involve different methods of interpolation, and the last gives us the possibility of reading less LPC coefficients from the bit stream. In the highest complexity level, level C3, the interpolation is performed on the LSPs, see Section 4.8.3. In lower complexity levels, the decoding is performed on the Log Area Ratios, see section 4.8.2. Table 4.7 summarize the decoder complexity levels.

Complexity level	Description
C3	Improved LPC interpolation (LSP): Full decoding
C2	Simplified LPC interpolation (LAR)
C1	Reduced order LPC synthesis filter

Table 4.7: A summary of the decoder complexity levels.

Chapter 5

Method and implementation

The program was written and tested on a Sun Ultra 10 work station. The utility programs for encoding, decoding and resampling of the input files are written in ANSI C, while the main application is written in Java 1.2. The Java application does not interact with the C program. The input files are resampled and encoded in order to get the parameters of interest, and then put in the same directory as the Java application. Since Java is a platform independent language, this means that the main program can be run on any platform, as long as the preprocessed input files are available in the same directory.

5.1 The software used in encoding the files

The objective of this part of the thesis was to utilize the encoded data from the MPEG4 encoder in a speaker recognition application. To get hand of these parameters I had to make some minor changes to the Mp4 encoder utility program I downloaded from the location http://www.tnt.uni-hannover.de/project/mpeg/audio/software/ where I chose the file containing the tool for encoding and decoding of natural audio elementary streams. In addition, this program needs some files from yet another program, a program that can be downloaded from ftp://ftp.tnt.uni-hannover.de/pub/audio/AFsp/ (AFsp-V3R2.tar.gz). The README files describes in detail the steps that have to be taken to get the program run properly. A documentation of the CELP coder can be found at
http://www.tnt.uni-hannover.de/project/mpeg/audio/documents/.

5.2 Experiment setup

I have a set of 7 speakers, 3 females, and 4 males, who each pronounced a set of 18 utterances of varying duration. Every speaker was recorded in a single session to make all the recordings for one person as equal as possible. The acquisition has taken place in a normal office room environment, and the recordings were done with an ordernary consumer microphone, without any other special acoustic care. First, each person pronounced the same sentence five times. Then each person pronounced five different sentences, but every person read the *same* five different sentences. These data were used for the speaker verification test. At last, every person pronounced 8 different sentences from a newspaper (Aftenposten), where all the sentences for all the speakers were different. This was done to simulate speech independence. I also recorded several speakers in the same speech sample, that is, person A started to read a sentence, when A was finished, person B read another sentence, and so on, to check how well the program was able to segment the different speakers in the speech sample, and then recognize them.

Below I have listed the equal sentences, written and pronounced in Norwegian. It is the first sentence that is pronounced five times.

- I tillegg sørget det administrasjonsstyrte Norsk Hydro for at den største eieren kom under 50 prosent, og nå er det politisk nær umulig for staten å kjøpe seg opp igjen.

- Internt i Statoil var en sikker på at daværende leder av energikomiteen på Stortinget, Jens Stoltenberg, jobbet for dem.

- Etter eget utsagn skal de ha vært i koma etter at partileder Thorbjørn Jagland kuppet møtet med overraskende å trekke seg som statsministerkandidat til fordel for Jens Stoltenberg.

- Informasjonsdirektør Finn Langeland har rett i at "den politiske utvikling har løpt fra at næringslivet skal finansiere partiene".

- Rent foreløpig var resultatet at Dag Danielsen skaffet seg flere stemmer i den interne nominasjonskamp med Hagens kandidat, nestleder Siv Jensen.

- I våre mange møter og lange brevveksling, både mens han satt inne og i tiden etterpå, har Butkov med styrke hevdet at var og er uskyldig i anklagene som ble rettet mot ham.

The choice of sentences was based on a speech sound excitation point of view. I wanted to use sentences that contained both *voiced*, *unvoiced*[1], and *plosive* sounds.

The input files are in WAV format, and sampled at 8000 Hz, 16 bit mono. The frame duration is 10 ms, and the LPC order is $P = 10$. The WAV format is described in Section 4.7.

5.3 Computation of the regressive cepstrum conversion

After the input files had been sampled in the appropriate way, I used the utility program *mp4enc* to produce the a_p parameters. These parameters are not written to file by default, so I had to make some minor changes to the program. These changes do not affect any of the computations, I just added a few lines to make the program store the a_p-parameters on disk. The program can be given several options that do affect the output parameters. The bitrate was set to 11000 kbit/s, LPC coding and vector quantization was turned on, and fine-rate control was turned off. These are the default settings when the file is sampled at 8 kHz.

I tested the program on three different tasks. First I simulated speaker verification by letting each person pronounce the same sentence five times, as mentioned above. I used the first four samples as a training set, and used the last utterance as the test sample.

Then I took another approach to the same speaker verification task, but now I used the next five samples. These samples were all different, but, as mentioned, every speaker pronounced the same five different sentences.

The third test is the real challenge. Here I use as a training set five different sentences, where all the sentences are different for all the speakers. No two sentences are the same. I then use a compound sentence, consisting of three speakers, as a test utterance, and the program will try to segment the utterance into speakers, and then recognize the different speakers. The sentences the speakers pronounced in the test sample, were all different from the sentences used in the training set.

I use the cepstrum coefficients in the speaker recognition process instead of using the a-parameters directly. The reason for this is that if we assume that the system is time invariant, the temporal average of the cepstral vectors issued from a single speaker does represent the cascade of his average vocal tract cepstral response and the channel cepstral response. Subtracting two

[1] Also called *fricative* sounds

averaged cepstra leads to terms which cancel out (same channel) and terms which possibly do not (vocal tract), resulting in a cepstral vector whose magnitude hopefully represents same kind of discrepancy between speakers.

To be more precise, let $\mathbb{S}$ denote the set of S speakers i, and let $\mathbb{L}_{i,k}$ denote the i'th speakers k'th utterance characterized by a set of cepstral vectors $\mathbf{C}_{i,k,l}$

$$\mathbb{S} = \{i | i \in [1, S]\}, \quad S = \text{number of speakers} \tag{5.1}$$

$$\mathbb{L}_{i,k} = \{\mathbf{C}_{i,k,l} | k \in [0, K_i - 1], l \in [0, L_{i,k} - 1]\}, \quad L \text{ is varying} \tag{5.2}$$

K denotes the number of utterances for speaker i. The varying length of L in equation (5.2) is explained by the fact that even in the case where the speakers pronounce the same sentences a number of times, the recordings will not be of exactly the same duration. Since L denote the number of cepstral coefficients representing the k'th utterance of speaker i, the L will not be a constant.

Now, the average cepstrum for speaker i taken over the k training utterances is computed by the following expression:

$$\mathbf{C}_i = \frac{1}{L_{i,k} K_i} \times \sum_{k \in [0, K_i - 1]} \sum_{l \in [0, L_{i,k} - 1]} \mathbf{C}_{k,l}, \quad i \in [1, S] \tag{5.3}$$

By using these definitions, I now measure in an Euclidean space the squared distance between a scalar $\mathbf{C}_i$, representing the averaged cepstrum of the training utterances for speaker i, and $\mathbf{C}$, the averaged cepstrum of an utterance from an unknown speaker in the following way:

$$d^2(i) = |\mathbf{C}_i - \mathbf{C}|^2, \quad i \in [1, S] \tag{5.4}$$

This distance is computed for all the 7 persons, and it is assumed that the unknown utterance represented by $\mathbf{C}$ belongs to the person who is represented by the averaged cepstrum $\mathbf{C}_i$ that minimizes the distance $d^2(i)$. In the case of speaker recognition, this is done in a similar way, except that I now split the speech sample into pieces and analyze every single one of them to find the person they correspond to.

5.4 Segmenting the speech sample when more than one speaker is present

The speaker recognition application also reveals the problem of segmenting the speech sample into different speakers. For this purpose I used a sort of an

endpoint detection algorithm. I used the first 100 millisecond of the utterance to determine the energy of the background noise, and used this together with the maximum peak value to compute a lower and upper threshold. These thresholds were used to determine whether or not an interval should be characterized as a pause between two speakers. The method used was found in [18]. This algorithm only deals with detecting endpoints in a single utterance, but the main principles can be used.

The speech "energy", $E(n)$ is defined as the sum of the magnitudes of 10 milliseconds of speech centered on the measurement interval,

$$E(n) = \sum_{i=-40}^{40} |s(n+i)| \tag{5.5}$$

It is assumed that the sampling frequency is 8 kHz, which explains the summation indices. This use of magnitude de-emphasizes large-amplitude speech variations and produces a smoother energy function. Some typical energy functions for the words *directive* and *multiply* is shown in Figure 5.1 and 5.2.

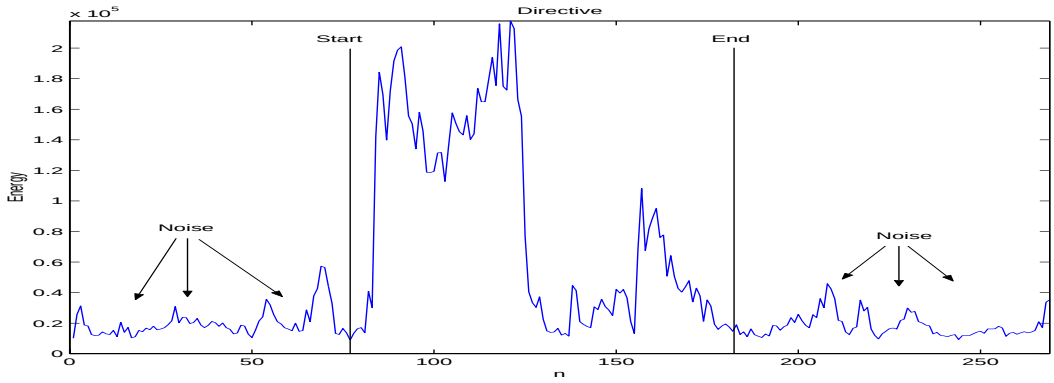

Figure 5.1: Energy plot for the word "directive" with markers indicating the noise, and the beginning and ending of the utterance.

As seen from these two figures, it is not as easy as it sounds to distinguish the noise from a weak utterance, especially if one of the speakers speaks very loudly in proportion to the other speakers in the same sample. This will affect the thresholds, and make the distinction difficult. In Figure 5.3 I have plotted a male and a female speaker in the same energy plot. We can imagine the difficulties if the difference is even greater, and still more background noise exists.

The computation of the thresholds is done in the following way: I let MXP denote the peak energy, that is, the maximum value of the energy plot,

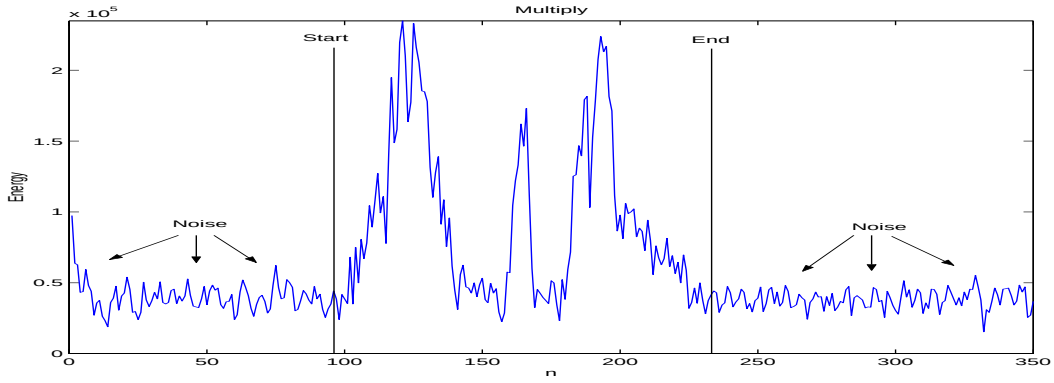

Figure 5.2: Energy plot for the word "multiply" with markers indicating the noise, and the beginning and ending of the utterance.

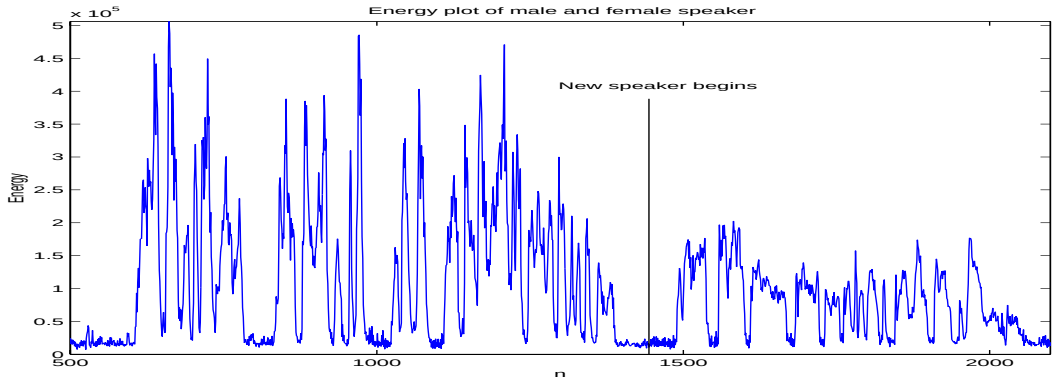

Figure 5.3: Energy plot of two different speakers. Notice the difference in energy. The first speaker is a male, the second a female.

and MNE the mean energy of the noise. MNE is computed from the first 100 ms of the speech sample, which consists of background noise. These values are then used to set the two threshold values, MXTV (Max Threshold Value), and MNTV (Minimum Threshold Value), according to the following rule:

$$I1 = 0.03 \times (MXP - MNE) + MNE \quad (5.6)$$

$$I2 = 4 \times MNE \quad (5.7)$$

$$MNTV = \min(I1, I2) \quad (5.8)$$

$$MXTV = 3 \times MNTV \quad (5.9)$$

Some explanation is required. Equation (5.6) shows $I1$ to be a level which is 3 percent of the peak energy, adjusted for the silence energy. Next, equation (5.7) makes $I2$ be a level set of four times the silence energy. The lower threshold value, MNTV, is set to the minimum of $I1$ and $I2$ in equation (5.8), and in equation (5.9) the upper threshold, MXTV, is set 3 times the lower threshold value. The value 3 in equation (5.9) turned out to be the best value after several tests.

The first step in the algorithm, after determining these thresholds, is to search the energy values for the beginning of the speech. The algorithm searches through the energy samples, $E(m)$, until $E(m)$ exceeds MNTV. This value of m is set as a temporary speech offset. Then the algorithm checks whether the energy falls below MNTV before it exceeds MXTV. If it does, it search for a new offset. If the energy does *not* fall below MNTV before it exceeds MXTV, the temporary offset really *was* the beginning of an utterance. This algorithm for detecting the starting point is depicted in Figure 5.4. In the last stage before *DONE*, *OFFSET+=1* is equal to *OFFSET = OFFSET+1*.

The detection of the endpoint of an utterance is a bit more complicated, and the algorithm is not guaranteed to be correct. When the starting point is found, what must be done is to find the first point where $E(n) < MNTV$. If $E(n)$ remains less that the upper threshold for some predefined interval, it is reasonable to assume that a change of speakers took place. The hard part here is to determine the interval for which $E(n)$ should stay below MXTV. This interval must depend on the length of the utterance, and after several tests I found that the following formula could be used:

$$\text{interval} = 0.0263 \times length(E) \tag{5.10}$$

It is important to note that if a person's sentence contains a comma, there will also be a pause, but not a new speaker. The program does a check to see whether the new speaker it detected is the same as the former one, but if the sentence only contains a few words after the comma, the possibility of suggesting the wrong speaker for this short part increases. This is because the average of the cepstrum coefficients are based on only a few points, and extreme values will have a more serious influence on the result. In Figure 5.5 I give an illustration of the detection of the speech endpoint.

If $i++ > length(E) - 1$ we have reached the end of the file, and the endpoint of the speech utterance has to be set to m.

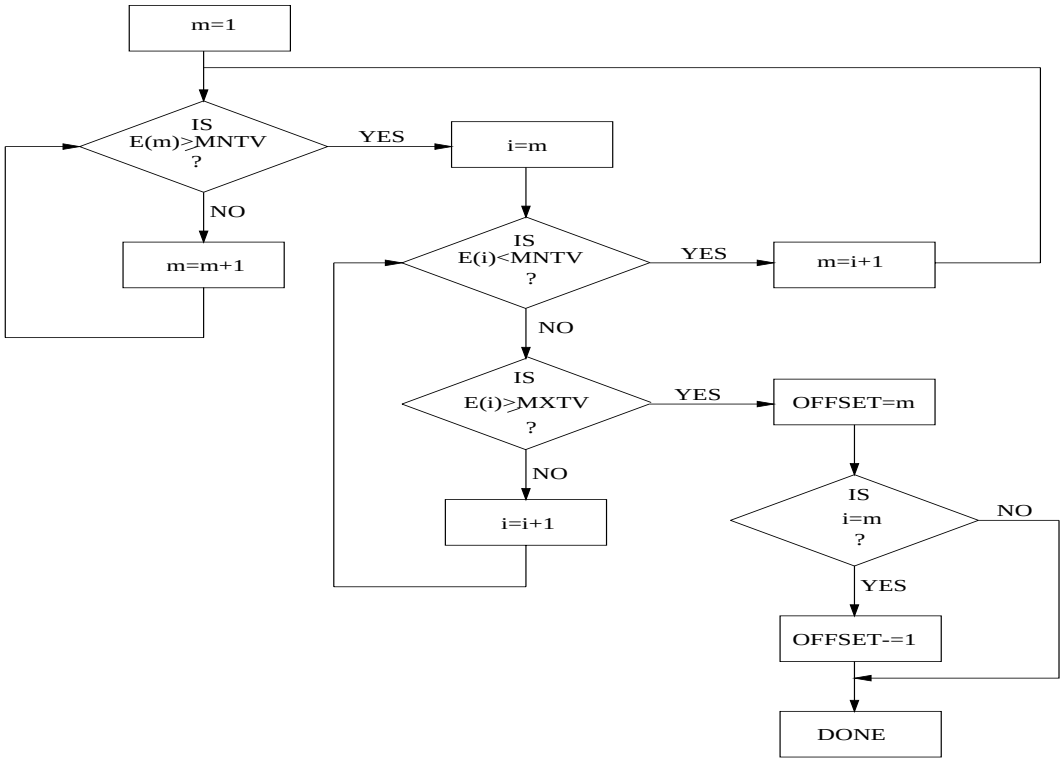

Figure 5.4: Flowchart illustration for the beginning point estimation of an utterance.

5.5 How the tests were carried out

Before I starte describing the details about how the tests were carried out, it is important to emphasize that for all the tests, the utterances that make up the training sets and the set of utterances that are used in the testing phase are mutually exclusive. This means that none of the utterances used in the testing phase has been used in the computation of $\mathbf{C}_i$ in equation (5.3).

The first test involves five utterances of the same sentence for each speaker. I wanted to observe what influence the number of speech samples in the training set had on the result, so the program was tested with four different training sets for each person. That is; first I used the first utterance of each person as a training vector for that person. Then I ran all the other utterances, 28 in number, through the program to see how many correct matches i got, and how many errors. Then, I used every person's second utterance as a training vector for that person, and tested against all the utterances that were not used in the training set, and so on. The same approach was followed

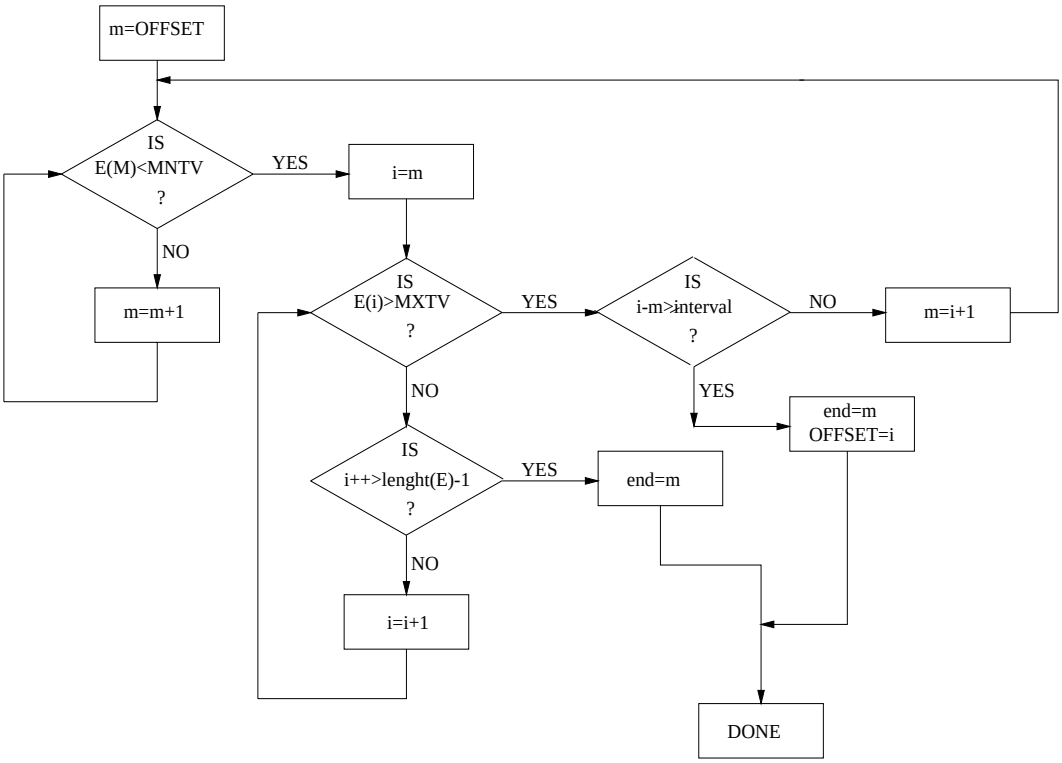

Figure 5.5: Flowchart illustration for the end point estimation of an utterance.

in the case of a training set consisting of two utterances, but now there were only 21 test utterances that was not a part of any persons training set.

Chapter 6

Results and Discussion

In this chapter I will give the results of the different tests followed by a discussion and explanation of the specific results. I will finish the presentation and the discussion related to one test before I move on to the next. As mentioned in the Method chapter, I have done three different test, and I will give a short summary of them before I present the results. The sentences that were uttered were listed in the Method section where I also gave an explanation for the choice of sentences, but I will nevertheless list the sentences in the introduction of each section for clarity. The results will be illustrated with tables and graphs. This, together with my explanations, will hopefully give an understanding of what the different results mean.

6.1 Text dependent speaker identification based on five equal sentences

In this first test, all the speakers pronounced the same sentence five times. The sentence used was the following, written and pronounced in Norwegian: *I tillegg sørget det administrasjonsstyrte Norsk Hydro for at den største eieren kom under 50 prosent, og nå er det politisk nær umulig for staten å kjøpe seg opp igjen.* The test was performed on four training sets of different length, and the results are shown in Table 6.1 and 6.2.

As seen from the tables, the error rate is very high. This was no surprise. First of all, the method I have used is very simple. It only considers the cepstrum coefficients, and does not take into account the pitch period of the speakers, which could be of great help in first determining the speaker's sex before trying to determine the specific speaker. For the results in the upper half of Table 6.1, the training set is built up of only one utterance. There might be quite a level of variation from utterance to utterance, which

Number of training samples pr. person = 1			
Training sequence	Num. of tests	Num. of errors	Error(%)
1	28	23	82
2	28	22	79
3	28	23	82
4	28	22	79
5	28	20	71
		Average error rate	79
Number of training samples pr. person = 2			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2	21	17	81
1, 3	21	15	71
1, 4	21	17	81
1, 5	21	16	76
2, 3	21	14	67
2, 4	21	14	67
2, 5	21	15	71
3, 4	21	17	81
3, 5	21	16	76
4, 5	21	14	67
		Average error rate	74

Table 6.1: A summary of the performance of the algorithm in a text dependent speaker recognition application. All the utterances were equal.

could lead to such a big difference between the training sequence and the test sample that the test sample is classified as the wrong person. In the second half of Table 6.1 the average error rate is lower than in the upper half, though it is still very high. A lower error rate should be expected, since the mean of *two* utterances is now used in the training process. This average will tend to minimize the effect of major deviation in one of the utterances, for example if a person halted or did not read correctly or hawked. With this in mind we should expect a further reduction in the error rate in the upper half of Table 6.2 where the training set is an average of three utterances. This is indeed true, though there is only a minor reduction. In the lower half, on the contrary, where we would expect the lowest average error rate, there is an increase. This is due to the few test utterances available. After using four out of five utterances for every speaker as training utterances, there were only one utterance left for each speaker to test. With a greater sample space

Number of training samples pr. person = 3			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2, 3	14	10	71
1, 2, 4	14	9	64
1, 2, 5	14	12	86
1, 3, 4	14	11	79
1, 3, 5	14	10	71
1, 4, 5	14	9	64
2, 3, 4	14	11	79
2, 3, 5	14	11	79
2, 4, 5	14	8	57
3, 4, 5	14	10	71
		Average error rate	72
Number of training samples pr. person = 4			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2, 3, 4	7	6	86
1, 2, 4, 5	7	4	57
1, 2, 3, 5	7	5	71
1, 3, 4, 5	7	6	86
2, 3, 4, 5	7	5	71
		Average error rate	74

Table 6.2: A summary of the performance of the algorithm in a text dependent speaker recognition application. All the utterances were equal.

utterances, we would have seen a reduced, or at least equal, error rate as the former test with three utterances in the training sets. This is provided the data is still recorded under the same circumstances.

Now, this explains the reduction, or in the last case, the lack of reduction, in the error rate, but I have still not completely explained *why* the error rates are as high as they are. To get a better understanding of the difficulties, I have plotted the five averaged cepstrum values, $\mathbf{C}_i$, representing each of the five utterances for every one of the seven persons in Figure 6.1

As seen, some of the graphs intersect, which make the task of differentiating between so many different speakers based on the cepstral coefficients only, a difficult one. But, the method is a good approach, because as we can see, at least three of the lines are clearly separated, and would yield a fairly good result, if these three speakers were the only one participating.

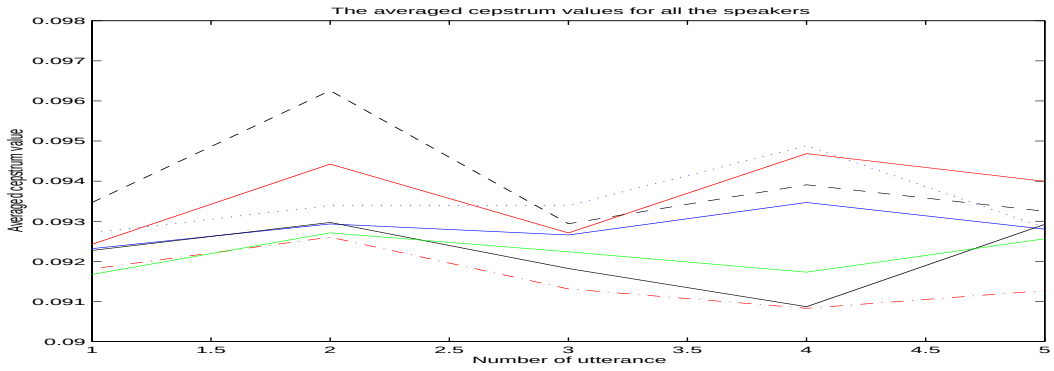

Figure 6.1: Plot of the averages of the cepstral values for each utterance. All sentences were equal.

6.2 Text dependent speaker identification based on five different sentences

This part is quite similar to the one above, in that there are still seven persons pronouncing five sentences, but now they pronounce five sentences that are all different. All the seven speakers pronounce the *same* five different sentences. The sentences, written and pronounced in Norwegian, are listed below

- Internt i Statoil var en sikker på at daværende leder av energikomiteen på Stortinget, Jens Stoltenberg, jobbet for dem.
- Etter eget utsagn skal de ha vært i koma etter at partileder Thorbjørn Jagland kuppet møtet med overraskende å trekke seg som statsministerkandidat til fordel for Jens Stoltenberg.
- Informasjonsdirektør Finn Langeland har rett i at "den politiske utvikling har løpt fra at næringslivet skal finansiere partiene".
- Rent foreløpig var resultatet at Dag Danielsen skaffet seg flere stemmer i den interne nominasjonskamp med Hagens kandidat, nestleder Siv Jensen.
- I våre mange møter og lange brevveksling, både mens han satt inne og i tiden etterpå, har Butkov med styrke hevdet at var og er uskyldig i anklagene som ble rettet mot ham.

As before, I did four tests with the difference being the number of utterances used when computing the training values. The results are presented in Table 6.3 and 6.4.

Number of training samples pr. person = 1			
Training sequence	Num. of tests	Num. of errors	Error(%)
1	28	24	86
2	28	19	68
3	28	21	75
4	28	20	71
5	28	17	61
		Average error rate	72
Number of training samples pr. person = 2			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2	21	15	71
1, 3	21	14	67
1, 4	21	14	67
1, 5	21	18	86
2, 3	21	11	52
2, 4	21	17	81
2, 5	21	14	67
3, 4	21	13	62
3, 5	21	12	57
4, 5	21	16	76
		Average error rate	69

Table 6.3: A summary of the performance of the algorithm in a text dependent speaker recognition application. All the sentences were different, but all the speakers pronounced the same five different sentences.

We first notice that the values in the upper half of Table 6.3 have, on the average, a less error rate than the results presented in the upper half of Table 6.1. This is perhaps not what we would expect. Intuitively we would expect the test where two equal utterances were compared, to give the best result. On the average it would, and in this case it may depend on the specific sentences pronounced, and is not a general property of the algorithm. It is clear that when the same sentence is uttered several times, the discrepancy in the averaged cepstrum values will tend to be less than the discrepancy observed when each sentence is different from all the others.

Number of training samples pr. person = 3			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2, 3	14	8	57
1, 2, 4	14	9	64
1, 2, 5	14	12	86
1, 3, 4	14	10	71
1, 3, 5	14	10	71
1, 4, 5	14	12	86
2, 3, 4	14	8	57
2, 3, 5	14	9	64
2, 4, 5	14	12	86
3, 4, 5	14	8	57
		Average error rate	70
Number of training samples pr. person = 4			
Training sequence	Num. of tests	Num. of errors	Error(%)
1, 2, 3, 4	7	3	43
1, 2, 3, 5	7	5	71
1, 2, 4, 5	7	7	100
1, 3, 4, 5	7	4	57
2, 3, 4, 5	7	4	57
		Average error rate	66

Table 6.4: A summary of the performance of the algorithm in a text dependent speaker recognition application. All the sentences were different, but all the speakers pronounced the same five different sentences.

Again we see that the method of looking at the cepstral coefficients would work fine if the sample space (number of speakers) was smaller. If, for example, the sample space consisted of the three speakers corresponding to the dashed black line, the blue line, and the solid black line, the method would work fine. From these two plots we can see that the averaged cepstrum coefficients do not result in a fixed slope characterizing a person. As different sentences are read, different sounds are pronounced, and the vocal tract assume different shapes. This again imply a change in the filter coefficients in the AR filter which leads to different cepstral coefficients. However, the placement in the plane relative to each other are preserved in some sense in the different plots, even though this quality is less illustrious when all speakers pronounce different sentences, as will be shown in the next section.

The second half of Table 6.3 also show a better result than the second half of Table 6.1, though the difference is less than in the two corresponding upper half. The error values also vary more than the values in Table 6.3. Some of the values are very low, while others are extremely high. If we look at the cepstral plots, we see that in Figure 6.1 the trend is that the different graphs try to follow the same path. Since the persons utter the same sentence every time, the difference in the averaged cepstrum will not be as large as when they pronounce different sentences all the time.

An interesting thing to note from the test results in Tables 6.3 and 6.4 is that the error rate seems to stabilize around 70 percent, regardless of the number of training utterances. The last test with four training values is an exception, but as pinpointed in the previous section, the number of test utterances is too low to jump to any conclusions. As the sentences to pronounce change for each pronunciation, averaging the values will not have the same effect on the error rate as in the first case.

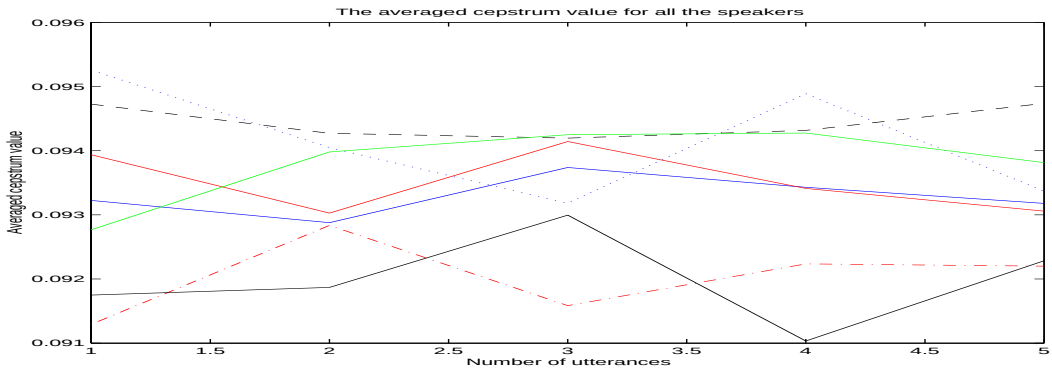

Figure 6.2: Plot of the averages of the cepstral values for each utterance in the second test. All speakers pronounced the same five different sentences.

6.3 Text Independent Speaker Recognition Based on Only Different Sentences

In this last test I simulated a text independent speaker recognition application. Every person pronounced eight sentences of varying duration, and all the sentences were different for all the persons. Taking into account the results from the former tests, I used three utterances as training data for each person. The data are presented a little different from above. I made 20

tests. I had 10 compound sentences, consisting of three speakers. I made tow different training sets, and tested both of them on the same 10 compounded sentences to make the number of tests equal 20. Since the utterances consists of three speakers, it is interesing to see if the program segmented the data the right way. I therefore use certain categories for the tests: did the program segment the data correctly, how many of the speakers did it recognize correctly, or did it not recognize any of them. For each test I also record the number of correct recognitions. This is what the number 1, 2 and 3 indicate. In the column labeled "1" is the number of test for which only one person were recognized correctly, 2 indicates the number of tests where two persons were recognized correctly, and 3 means that all the speakers were recognized. The result of the tests are presented in Table 6.5

#tests	#training utterances	#correct segmentation	1	2	3
20	3	11	9	4	1

Table 6.5: The results after testing the text independent speaker recognition application on 10 test samples with different training sets.

#correct segmentation is the number of times the program indicated that there were exactly three speakers. Notice that a person may pause, and then continue. The program performs a check to see if the new person is the same as the former, and if it is, the segmentation into more than three is not regarded as wrong. Moreover, even if the program segment for example the midle part of an utterance wrong, and classify these parts as wrong persons, it may still classify correctly the person that begin the sentence and/or the person that ends it. So, if the program segment an utterance incorrectly, it may still recognize correctly some of the speakers.

In Figure 6.5 I have plotted the averaged cepstral values for all the eight utterances for each of the seven persons. As we can see, it is not a simple task to distinguish and identify the speakers based on the cepstrum values. This is also indicated by the results in Table 6.5, where we just got a perfect match for one single test sentence. But, based on the observations in the two previous experiments, this was no supprise. This is a very simple method that tends to work best when there are only two or three speakers present. Nevertheless, the poor results in the last experiment can not be blamed on the cepstral speaker recognition method alone. The segmentation procedure, when characterizing the end of an /s/ pronounciation as silence, makes it difficult to the speaker recognition function to work properly. Figure 6.3 shows a plot of the waveform for the beginning of the word *six*. It is not

easy to differentiate between the background noise and the beginning of the utterance, but the frequency content of the speech is radically different from the frequency content of the background noise as manifested by the sharp increase in the zero crossing level.

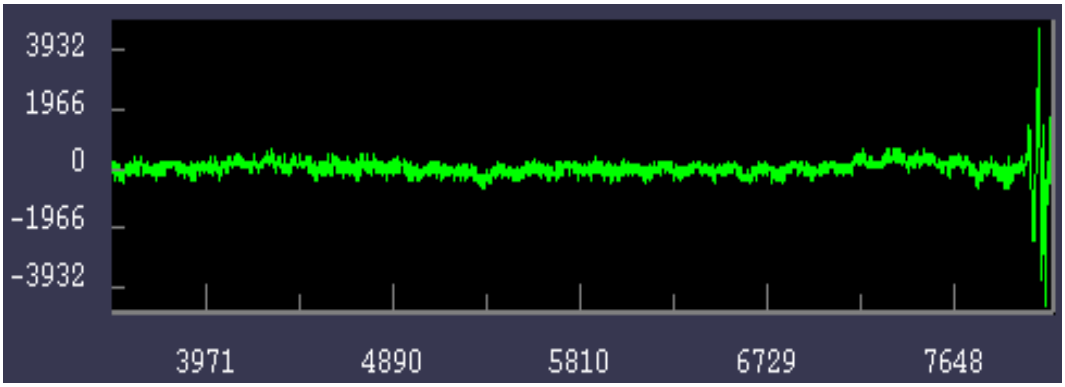

Figure 6.3: Waveform for the beginning of the word "six".

Another example of the difficulty in locating the endpoint, and it will not be detected by the energy measurement only, is shown in Figure 6.4. This figure shows the waveform for the end of the word "five". The final /y/ in "five" becomes devoiced and turns into an /f/, a weak fricative. Such weak fricatives are difficult to locate by eye, and sometimes even by ear.

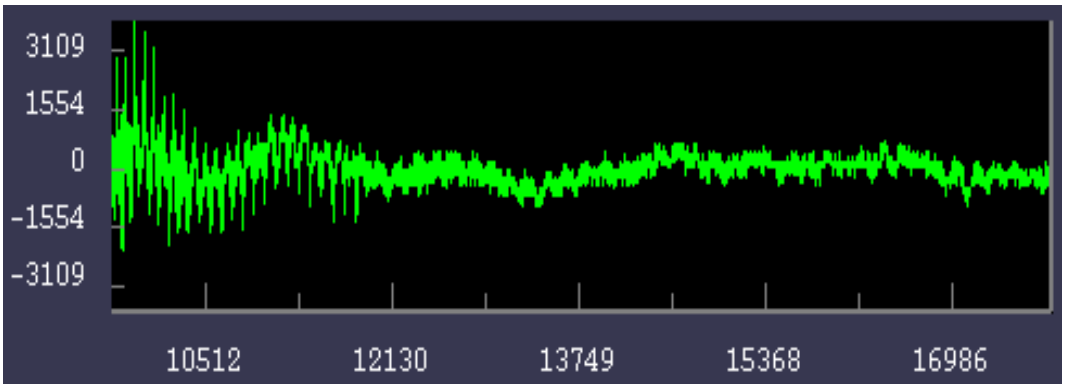

Figure 6.4: Waveform for the end of the word "five".

Sometimes, small regions are segmented and characterized as speech, and with few cepstral coefficients, the recognition procedure may very well fail. The combination of these two simple methods of energy measurement and

cepstral analysis in a text independent speaker recognition task is therefore of very little value. For advanced operations like this, what we need is more sophisticated methods that take into account other properties of speech as well as just the cepstrum values. On the other hand, for most of the tests the segmentation worked just fine. If the speakers are clearly distinguished, the method will find the correct number of speakers. A technique to improve the segmentation algorithm is suggested in Chapter 7. The use of the cepstral coefficients is not as bad as it seems either. Without some extensions it can not, at least not for any practical purposes, be used in conjunction with the energy base segmentation method to identify speakers in utterances where several speakers are present and with a large sample space, but it seems to be useful for speaker verification purposes with small sample spaces. Optimizations of the algorithm is suggested in Chapter 7.

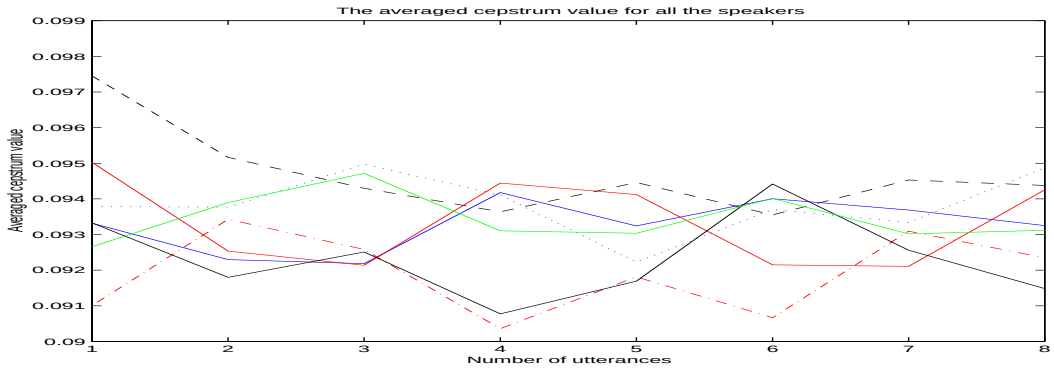

Figure 6.5: Plot of the averages of the cepstral values for each utterance in the second test. All utterances for all speakers were different.

6.4 Experiences with the use of Java in different signal processing applications

After working with Java for about two year as a tool for solving different signal processing tasks, the overall impression is tha Java is well suited for this purpose. As the language is growing, and more classes and APIs are added to it, the language is usefull for many other tasks then just web-programming. As this is written, a packet was newly released that contains different speech processing tools, but I have not been able to check it out. Of course, it cannon yet compete with Matlab as a scientific and engineering

programming language, but this is what Matlab is designed for, Java is so much more in addition.

It can perfectly well be used to take care of the visual user interface while integrated with some native language that takes care of some computational tasks. But, as new compilers are released, and the speed increases, I think it is just a matter of time before it can start to compete with languages as C and C++. Perhaps it will not be quite as fast, but the improvement over the last few years has been tremendous.

The APIs that I have been used, JNI and JMF, are packets that in a relatively simple way effectively enables integration with other languages, and streaming of media over the Internet. The object orientation makes it easy to keep control over even large programs, and since no functions or variables can lie outside any class, this makes it easier to split the program into several files. In the SpeechAnalyzer frame[1] I let every component be a class, that is, I extended the original classes for window components. The base class control user inputs, and delegates the operations to the different object. This makes it very easy to extend the program, you do not have to make major changes in the program, you just do the necessary changes in the particular object. Another advantage, in opposite to C and C++ is that debugging is much more easier. The compiler gives reasonably good error messages, and the pointers from C and C++, that caused many errors very difficult to discover, is dropped.

There are some drawbacks too. It is possible to do fancy layout in Java, but it is not always as easy as it sounds. In figure 6.6 I show the layout of my SpeechAnalyzer frame that I developed during the work with these thesis. It is not particulary good, but nevertheless it took quite an amount of time to make it look the way it should. However, for some time to come, it is the lack of speed that is the biggest problem. But despite some of the problems I had, Java is an easy language to master, at least if we compare with C or C++. As the language develope, the popularity increases,and with the experience I got from this work, I think Java will be a good and common tool for signal processing applications.

[1] A frame is the same thing as an applet, except that it is run from the command line. It also allows access to the local file system, something applets do not.

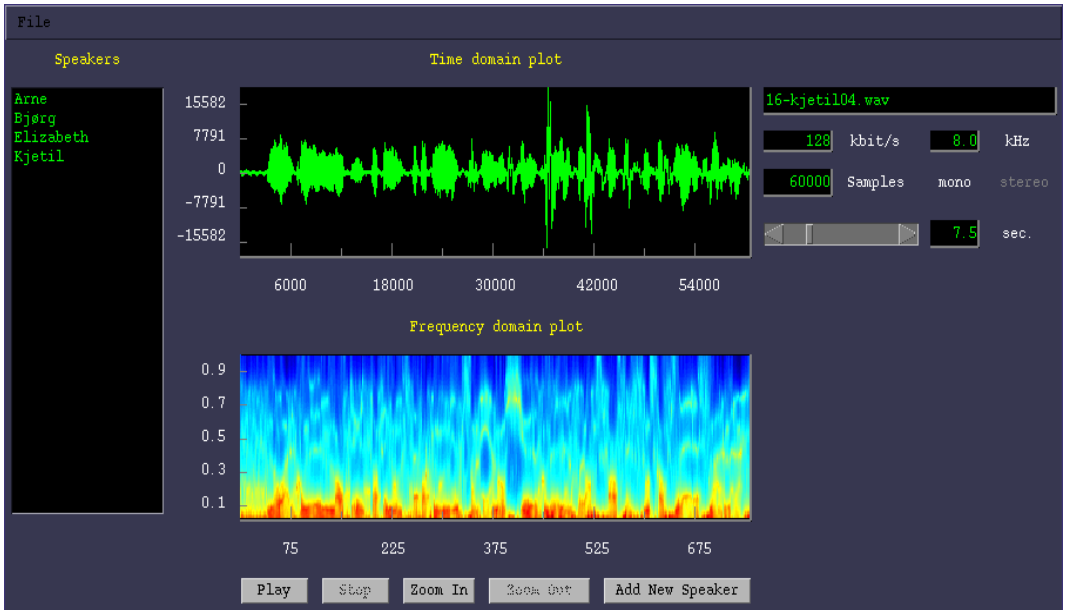

Figure 6.6: The SpeechAnalyzer frame

Chapter 7

Conclusion and further work

After a first glance at the results presented in Table 6.1 - 6.4, it may seem like the cepstrum approach of identifying speakers is a bad approach. I do not quite agree. First we have to take into account the number of speakers that were present. It is evident that the method did not work very well for these seven speakers, but in the plots is Figure 6.1 - fig:averagedCepstral2 we see that some of the cepstral graphs are clearly separated from each other, and with this in mind we can conclude that this approach will work better for small sample spaces, that is, about three or four persons. For this case, the algorithm provides a simple and efficient solution to the problem of identifying speakers, especially when we want to identify them by a single utterance. If this idea was to be extended further, there are a few things that could be implemented to improve the result. One approach is to take into account the peak period. Females usually have a higher peak frequency then males, so this could be used to determine if the unknown person we are trying to identify is a male or female. This is useful for narrowing the search for the correct person.

Furthermore, I have done a very coarse averaging of the cepstrum coefficients. One extension could perhaps be to use an other distance measure and averaging. We have ten parameters for each frame. Instead of averaging all the coefficients so that we end up with a single scalar, that necessarily leads to a grate amount of loss of information, what could be done is the following, assuming that the utterances are of exactly the same length: For each encoded file, we end up with a $10 \times$ number of frames matrix. We could then add, on a element by element basis, the elements in the matrices, and then multiply every element by a fraction of the number of matrices. Stated more mathematically; let $\mathbf{C}^j, \quad j = 0, \ldots, J-1$ denote one of the J encoded training utterances. Then an element $c_{i,j}^{mean}$ in the $10 \times$ number of frames averaged matrix $\mathbf{C}^{mean}$ could be computed in the following way.

$$c_{ij}^{mean} = \frac{1}{J} \times \sum_{k=0}^{J-1} c_{ij}^{k} \tag{7.1}$$

Now, given the encoded test utterance, $\mathbf{C}^{test}$, we might subtract the two matrices on an element by element basis to obtain a new matrix $\mathbf{C}^{diff}$, where $c_{i,j}^{diff}$ is computed by

$$c_{ij}^{diff} = c_{ij}^{mean} - c_{ij}^{test}, \quad i \in [1, 10], j \in [1, frames], \tag{7.2}$$

where 10 is the filter order, and frames is the number of frames in the encoded utterance. We could then use a norm measure on the resulting matrix, for example the *Frobenius norm* defined by

$$||A||_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2} \tag{7.3}$$

It could be interesting to see whether or not this is a better measure of distance. But it assumes that the matrices are of the same dimension.

When dealing with speaker recognition in a sample that consists of two or more speakers, it is important that the algorithm that segments the speech into different speakers is effective in the way that it does not segment too heavily - resulting in many short speech samples. The energy based approach I took in these theses worked all right, but some improvements could be done. It is for example possible to take into consideration the *zero-crossing rate*. As we saw in Figure 6.3, it is not that easy to distinguish the end of the utterance of the word "six" from the background noise. To automatically detect this we could, as mentioned in the previous chapter compute the zero crossing rate, that is significantly higher for unvoiced sounds than for background noise. The article [18] suggests to use the first 100 ms of the speech signal to estimate the zero crossing rate during silence, as the minimum of some predefined threshold value, the sum of the mean zero crossing rate during silence plus twice the standard deviation of the zero crossing rate during silence.

In various other papers that deals with speaker identification, a threshold is computed that indicates when a claimed person should be accepted, and when it should be rejected (see for example [19] and [20]). Perhaps this should be implemented as a security check in my algorithm too. This is because no person is able to pronounce the same sentence in exactly the same way twice. In my algorithm it is assumed that the person the unknown utterance belongs to is added in a list of speakers, and then it assigns the utterance to the person that has the closest match between his averaged cepstrum, and

the cepstrum of the test utterance. A more robust program could operate with a threshold and reject the input utterance if the differences between the averaged cepstrum values exceeds this threshold.

Appendix A

Selected parts of the Java source code

A.1 Source code for reading WAV data

```
/**
 * These selected files exists in the object that
 * take care of the handeling of WAV data.
 * The following procedures are used in reading
 * WAV data from file, and store the samples
 * as type INT. The data are stored as 8
 * bit byte on the file, so they need to be
 * converted.
 */


private int stereo, channelNumbers, sampleRate, bytesPrSec;
private int bytesPrSample, numOfDataSamp;
private int wavDataIndex;
private int offset = 44;
private int[] samples = null;
private float playLength, playLength2;
AudioStream audioStream = null;

/**
 * The constructor which sets the file name
 * and the path to the file we are going to read.
 * Then it calls 'readData()'
 * INPUT  : a File object spesifying the filename and path
 */
```

```
WavData(File file) {
  this.file = file;
  readData();
} /* End of constructor */

/**
 * Reads the data, and checks wether or not the
 * data read are valid wav-data.
 */
protected void readData() {

  try {
      fis = new FileInputStream(file);
      data = new byte[fis.available()]; /* available() gives the number of bytes
                                           that can be read without blocking */
      fis.read(data);
      checkValidity();
      if (isLegalFile()) {
        readHeaderInfo();
        readSamples();
      }
  }
  catch(IOException e) {
    System.out.println("Error in reading data!");
  }
} /* End of procedure 'readData()' */



/**
 * Concatenate every byte in the file to legal samples.
 */
private void readSamples() {
  int maxValue = 0;
  // Check to see wether this is a monofile sampled with
  // two bytes pr sample or just one, respectively.
  if (!isStereo()) {
    if (bytesPrSample == 2) {
      samples = new int[(int)Math.ceil((data.length - offset)/2)];
      int j = 0;
      for (int i = offset; i < (data.length-2); i = i+2, j++) {
        for (int k = 0; k < 2; k++)
          samples[j] = samples[j] + (data[i+k]<<(k*8));
```

```
      }
    }/* End of test for two bytes pr sample */
    else if (bytesPrSample == 1) {
      samples = new int[data.length - offset];
      int j = 0;
      for (int i = offset; i < data.length; i++, j++) {
        samples[j] = (int)data[i];
        if (samples[j] > maxValue)
          maxValue = samples[j];
      }
      // This part is necessary because in the 8 bit case,
      // the samples are really unsigned, but in Java no
      // 'unsigned int' exists.
      for (int i = 0; i < samples.length; i++) {
        int sign = (samples[i] < 0 ? -1 : 1);

        if (sign == 1)
          samples[i] = samples[i]-maxValue;
        else
          samples[i] = (-1)*(Math.abs(samples[i])-maxValue);
      }
    } /* End of test for one byte pr sample */
  } /* End of mono file test */
} /* End of procedure 'readSamples()' */



/**
 * Check to confirm that this is real wav-data
 */
protected void checkValidity() {
  String riff;
  String wave;

  riff = new String(data, 0, 4);
  wave = new String(data, 8, 4);
  if ((wave.equals("WAVE")) && riff.equals("RIFF"))
    legalFile = true;
  else
    legalFile = false;
} /* End of procedure 'isWaveFile()' */
```

```
/**
  * Reads the header information of the wav-file, and sets the
  * necessary variables.
  */
 protected void readHeaderInfo() {
   int RIFF_CHUNK = 0;
   int FMT_CHUNK = 12;
   int DATA_CHUNK = 36;

   stereo = getHeaderPortion(FMT_CHUNK+8, 2);
   channelNumbers = getHeaderPortion(FMT_CHUNK+10, 2);
   sampleRate = getHeaderPortion(FMT_CHUNK+12, 4);
   bytesPrSec = getHeaderPortion(FMT_CHUNK+16, 4);
   bytesPrSample = getHeaderPortion(FMT_CHUNK+20, 2);
   if (bytesPrSample == 2)
     numOfDataSamp = (data.length - 44)/2;
   else
     numOfDataSamp = data.length - 44;
 } /* End of procedure 'readHeaderInfo()' */


 /**
  * Get the specified part of the header bytes.
  * INPUT  : an offset in the header stream.
  *        : a spesific interval of data to read.
  * OUTPUT : the integer value contained on the interval.
  */
 int getHeaderPortion(int offset, int length) {
   int j = 0;
   int sum = 0;
   for (int i = offset; i < offset+length; i++, j++) {
     sum = sum + (Math.abs((int)data[i]) << 8*j);
   }
   return sum;
 } /* End of procedure 'getHeaderPortion()' */
```

A.2 Source code for segmentation of speech

```
/**
 * These selected files exists in the object that
 * take care of the segmentation part of the
```

```
 * speech.
 */

/**
 * This function tries to segment the speech sample into speakers.
 * It is assumed that there is a little pause between speakers, that is
 * the different speakers does not overlap.
 * INPUT  : an object containing the mp4 data and a number of
 *            procedures to manipulate its own data.
 *          : a list of the registrated speakers.
 *          : an object that 'add' and 'get' speakers.
 */
public void segmentSpeech(Mp4Data mp4Data, SpeakerList speakerList,
                          SpeakerOrganizer speakerOrganizer) {
  int[] E = computeEnergy();
  Vector offsetEnds = new Vector();
  int interval = (int)(0.02634921391138637*(double)E.length);

  //Compute statistics for the noise, the first 100 ms of the speech segment.
  int zeroCrossingsPr10Ms = computeZeroCrossing(getSubIntArray(0, 799));
  float meanSilenceEnergy = 0f;
  for (int i = 0; i < 10; i++)
    meanSilenceEnergy += E[i]/10;

  //Compute max energy in the whole speech segment.
  int maxEnergy = 0;
  for (int i = 0; i < E.length; i++) {
    if (E[i] > maxEnergy)
      maxEnergy = E[i];
  }

  float I1 = (float)(0.03*((float)maxEnergy-meanSilenceEnergy)+meanSilenceEnergy);
  float I2 = (float)4*meanSilenceEnergy;
  float ITL = I1 < I2 ? I1 : I2;
  float ITU = 3.0f*ITL;

  System.out.println("ITL = " + ITL + ", ITU = " + ITU);
  //Find those intevals where the energy falls below ITL, and does not raise above
  //ITU before a certain amount of time.
  //First, find the beginning of the utterance.
  int start = 0;
  while (E[start] < ITU)
    start++;
```

```
    int i = start;
    int[] limits = new int[2];
    limits[0] = start;

    //Now, search through the rest of E
    while (i < E.length) {
      while (E[i] > ITL && i < E.length-1)
        i++;
      int offset = i; //What might be the beginning of a silence interval.
      while (E[i] < ITU && i < E.length-1)
        i++;
      int end = i;
      if ((end - offset) >= interval) {
        //System.out.println("E.length = " + E.length);
        //System.out.println("x/E.length = " + (70d/E.length));
        limits[1] = offset; //Since offset of silence is the end of speech utterance.
        offsetEnds.addElement(new int[] {limits[0], limits[1]});
        if (i < E.length-1) {
          limits[0] = end;
          i++;
        }

      }
      i++;
    }

    String prevName = new String();
    System.out.println(offsetEnds.size());
    for (int j = 0; j < offsetEnds.size(); j++) {
      Vector aparams = findCorrespondingAparams((int[])offsetEnds.elementAt(j), mp4Data);
      double[][] cepstrals = computeCepstralCoeffs(aparams);
      String name = findSpeaker(cepstrals, speakerList, speakerOrganizer);
      if (!name.equals(prevName)) {
        prevName = name;
        System.out.println(name);
      }
    }
  } /* End of procedure 'segmentSpeech' */



  /**
   * A utility procedure that computes the energy of the speech samples
```

```
   * OUTPUT : the energy of the speach file
   */
  public int[] computeEnergy() {
    int lowerBound = 0, upperBound = 0;
    int n = 0; int[] E = null;

    //Compute the playlength
    float playLength;
    playLength = samples.length/(float)(sampleRate);
    playLength = (playLength/(float)(channelNumbers)) * 1000; //To get milliseconds

    //For each iteration we move 10ms in time, and we evaluate on a time
    //interval of 10 ms centered around n.
    E = new int[(samples.length/80)+1];
    for (int i = 0; i < samples.length; i += 80) {
      if (i < 40)
        lowerBound = i;
      else
        lowerBound = 40;
      int diff = (samples.length-1) - (i+40);
      if (diff >= 0)
        upperBound = 40;
      else
        upperBound = 40 + diff;
      for (int j = i - lowerBound; j <= i + upperBound; j++)
        E[n] += Math.abs(samples[j]);
      n++;
    }
    writeEnergySamplesToFile(E);
    return E;
  } /* End of procedure 'computeEnergy' */



/**
   * Find the a-parameters that correspond to the selected protion
   * of speech.
   * INPUT  : the limit that indicates which part of the speech file
   *          we are interested in.
   *        : an object containing the mp4 data and a number of
   *          procedures to manipulate its own data.
   * OUTPUT : a Vector object containing the energy samples.
   */
```

```
private Vector findCorrespondingAparams(int[] limits, Mp4Data mp4Data) {

  //Now, what interval in the vector containing the a-parameters does
  //this correspond to. Remember, a new frame is computed every 10 ms.
  float playLength;
  playLength = samples.length/(float)(sampleRate);
  playLength = (playLength/(float)(channelNumbers)) * 1000; //To get milliseconds

  double numOfSampsPrFrame = (double)samples.length/
    (double)(Integer.parseInt(mp4Data.getNumberOfFrames()));

  int xLimit = (int)((limits[0]*80)/numOfSampsPrFrame);
  int yLimit = (int)((limits[1]*80)/numOfSampsPrFrame);

  Vector aParams = mp4Data.getAparameters();
  Vector subSet = new Vector(yLimit-xLimit+1);
  for (int i = xLimit; i <= yLimit; i++)
    subSet.addElement((double[])aParams.elementAt(i));

  return subSet;
} /* End of procedure 'findCorrespondingAparams' */



/**
 * Based upon the cepstral coefficients, estimate the speaker of this
 * segment. We do this by averaging the cepstral coefficents we
 * get as input, and compare this value to the averaged cepstral
 * values of the speakers present in the list.
 * INPUT  : a matrix containing the cepstral data of the unknown
 *            input file.
 *        : a list of all the possible speakers.
 *        : an object that controll the speaker object.
 * OUTPUT : the name of the speaker that most probably
 *        : is present in the test speach.
 */
private String findSpeaker(double[][] cepstrals, SpeakerList speakerList,
                           SpeakerOrganizer speakerOrganizer) {
  double averageCForSegment = 0d;

  for (int i = 0; i < cepstrals.length; i++)
    for (int j = 0; j < cepstrals[i].length; j++)
      averageCForSegment += cepstrals[i][j];
```

```
    averageCForSegment /= (double)(cepstrals.length * cepstrals[0].length);

    String[] items = speakerList.getItems();
    double min = 999999d;
    int minIndex = 0;
    for (int i = 0; i < items.length; i++) {
      double tmpMin = Math.pow((Math.abs(averageCForSegment -
                                speakerOrganizer.getSpeakerObject(items[i]).getAverageC())
      if (tmpMin < min) {
        min = tmpMin;
        minIndex = i;
      }
    }

    return items[minIndex];
  } /* End of procedure 'findSpeaker' */
```

A.3 Source code for various filters

```
/**
 * The 'Filters' class that contains procedures
 * for computing the frequency responce and
 * for conversion of a-parameters into
 * cepstral parameters.
 */


/* ************************************* */
/* ************ CLASS FILTERS ************ */
/* ************************************* */
/**
 * This class contains implementations of different
 * filters that will be useful in the mail
 * program.
 */
class Filters extends Object {


  /**
   * Calls the function that evaluate the transfer function for a
   * specific frequence. The first two parameters are the filter
   * coefficients, the last one indicates in how many points on
   * det unit circle (between 0 and PI) we will evaluate the function
```

```
 * in.
 * INPUT  : the coeffisients of the nominator
 *        : the coeffisients of the denominator
 *        : the number of points in which we sample on the
 *          upper half of the unit circle.
 * OUTPUT : an arry of complex values containing the
 *          frequency response of the filter.
 */
public static Complex[] freqz(double[] bp, double[] ap, int points) {
  Complex[] returnArray = new Complex[points];
  double increment = (double)(Math.PI)/(double)points;

  for (int j = 0; j < points; j++) {
    returnArray[j] = evalFreqResp(bp, ap, j*increment);
  }
  return returnArray;
} /* End of procedure 'freqz()' */


/**
 * Evaluate the filter H(z) = (Sum(bp*z^(-1)))/(1-Sum(ap*z^(-1)))
 * in 'points' points on the unit circle in the interval 0-PI.
 * INPUT  : the coeffisients of the nominator
 *        : the coeffisients of the denominator
 *        : an angle value in which we evaluate the filter
 * OUTPUT : the value (complex) of the frequency response
 *          evaluated in the angle w
 */
private static Complex evalFreqResp(double[] bp, double[] ap, double w) {
  int N = bp.length;
  int M = ap.length;

  //Nominator
  double Rno = 0; double Ino = 0;
  for (int n = 0; n < N; n++) {
    Rno += bp[n]*Math.cos(n*w);
    Ino += bp[n]*Math.sin(n*w);
  }

  //Denominator
  double Rde = 1; double Ide = 0;
  for (int m = 0; m < M; m++) {
    Rde += (-1)*ap[m]*Math.cos((m+1)*w);
```

```
    Ide -= (-1)*ap[m]*Math.sin((m+1)*w);
  }

  // Split the transfer funktion evaluated in w in real and
  // imaginary part, sum and create a Complex object.
  double realPart, imaginaryPart;
  realPart = (Rno*Rde + Ino*Ide)/(Math.pow(Rde, 2) + Math.pow(Ide, 2));
  imaginaryPart = (Rno*Ide - Ino*Rde)/(Math.pow(Rde, 2) + Math.pow(Ide, 2));

  return new Complex(realPart, imaginaryPart);
} /* End of procedure 'evalFreqResp()' */


/**
 * Computation of the spectrogram values.
 * INPUT  : an array of complex values containing the
 *          value of the transfer function.
 * OUTPUT : a real array containing the spectrogram values.
 */
public static double[] specGram(Complex[] H) {
  double[] specGramValues = new double[H.length];
  for (int i = 0; i < H.length; i++)
    specGramValues[i] = 20*Math.log(Math.pow(H[i].real,2) +
                                    Math.pow(H[i].imaginary,2));
  return specGramValues;
} /* End of procedure 'specGram()' */


/**
 * Computation of the cepstral coefficients.
 * The procedure takes the a-parameters as input.
 * INPUT  : an array containing a-parameters
 * OUTPUT : an array containing the corresponding
 *          cepstral values.
 */
public static double[]  computeCepstralCoeffs(double[] ap) {
  double[] c = new double[ap.length];
  for (int k = 1; k <= ap.length; k++) {
    double sum = 0;
    for (int i = 1; i <= k-1; i++)
      sum += i*c[i-1]*ap[k-i];
    sum *= (1/(k));
    c[k-1] = ap[k-1] + sum;
```

```
    }
    return c;
  } /* End of procedure 'computeCepstralCoeffs()' */


} /* End of class 'Filters' */

/* ************************************************************************** */
/* ************************************************************************** */
/* ************************************************************************** */
```

Bibliography

[1] Janet Abbate. The internet challenge: Conflict and compromise in computer networking. *Westview Press*, pages 193–210, 1994.

[2] Larry L. Peterson and Bruce S. Davie. *Computer Networks - A System Approach.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[3] Waldemar Brøgge. *Cappelens Leksikon.* J. W. Cappelens Forlag a.s., Oslo, 1984.

[4] Hamid Krim and Mats Viberg. Two decades of array signal processing research. *IEEE Signal Processing Magazine*, pages 67–94, 1996.

[5] George F. Simmons. *Eifferential Equations With Applications And Historical Notes.* McGraw-Hill, 1991.

[6] J.M. Reid J.J. Wild. *Diagnostic use of Ultrasound, page 248-257.* Br J Phys Med, 1965.

[7] Monson H. Hayes. *Statistical Digital Signal Processing and Modelling.* John Wiley and Sons, Inc., 1996.

[8] Bjørn A. J. Angelsen. *Principles of medical ultrasound imaging and measurments.* Department of Physiology and Biomedical Engineering Norwegian University of Science and Technology, Trondheim, Norway, 1996.

[9] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals.* Prentice-Hall, Inc., 1978.

[10] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall International, Inc., 1989.

[11] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra.* SIAM - Society for Industrial and Applied Mathematics, 1997.

[12] John Scourias. Overview of the global system for mobile communications. *http://ccnga.uwaterloo.ca/ jscouria/GSM/gsmreport.html*, 1995.

[13] A. S. Spanias. Speech coding: A tutorial review. *http://www.eas.asu.edu/ spanias/papers/review.ps*, 1997.

[14] Brian P. Murray and Anthony D. Fagan. An efficient codebook structure for celp. *unknown*, 1994.

[15] Karl Home Michael P. Chin. *Incorporating Voice Telecommunication with VSATS*. Tenth International Conference on Digital Satellite Communication. IEE London UK, 1994.

[16] Erik Harborg. Speech coding - methods, standards and implementations. *SINTEF Telecom and Informatics, N-7034 Trondheim, NORWAY*, 1997.

[17] Moving Picture Experts Group. Iso/jtc 1/sc 29n2203celp - information technology - coding of audiovisual objects. *ftp://ftp.tnt.uni-hannover.de/pub/MPEG/audio/mpeg4/documents/w2203/w2203clp.pdf*, 1998.

[18] L. R. Rabiner and M. R. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal, vol. 54, No. 2*, pages 297–315, 1975.

[19] H. Hugli P. Thevenaz. Combining four text independent speaker recognition methods. *http://www-imt.unine.ch/grp-hu/www/publication/paper/1990/ThHu90.html*, 1990.

[20] H. Hugli P. Thevenaz. Usefulness of the lpc-residue in text-independent speaker verification. *http://www-imt.unine.ch/grp-hu/www/publication/paper/1995/ThHu95.html*, 1995.