Single channel speech dereverberation

Florensa, Arnau Torrent

Award date:
2005

Awarding institution:
Electrical Engineering

Eindhoven University of Technology
Department of Electrical Engineering
Signal Processing Systems

TU/e technische universiteit eindhoven

Single Channel Speech Dereverberation

by
Arnau Torrent Florensa

Master of Science thesis

Project period: February 2005 – June 2005
Report Number: 18-05

.Supervisors:
Name (TU/e): Dr.ir. P.C.W. Sommen en ir. E.A.P. Habets

Name Universitat Politècnica de Catalunya

Si plores perquè s'ha post el Sol,
les llàgrimes t'impediran veure les estrelles.

Rabindranat Tagore 1861 - 1941,
poeta, dramaturg i filòsof indi.

Abstract

Speech signals recorded with a distant microphone usually contain reverberation and noise, which degrades the fidelity and intelligibility of speech, and the recognition performance of automatic speech recognition system.

The dereverberation of speech may be useful under a variety of situations, including the enhancement of speech recorded monaurally in reverberant enclosures such as rooms, and the enhancement of speech transmitted over telephone lines which has become reverberant either before transduction by microphone.

The goal of this research has been the real time implementation of an algorithm which dereverberates the speech signal. Also an improvement of the algorithm is presented and discussed.

The algorithm which has been used and implemented in real time, suppresses late room reflections (reverberation) from speech signals, based on spectral subtraction. The algorithm is based on a statistical model of the Room Impulse Response (RIR). This model is used to obtain an estimation of the Power Spectral Density (PSD) of the reverberation. This estimate the spectral subtraction and obtain a dereverberated speech signal.

Acknowledgments

This work is the result of my Master's Thesis carried out at the Department of Signal Processing Systems of the Technical University of Eindhoven (Netherland).

I want to thank my supervisor Emanuël A. P. Habets who helped me to keep my thesis in the right direction, gave me valuable advices and answered my numerous questions. This endeavor could not have been accomplished without his help.

I would like to express my gratitude to all the people who taught me. In this way, I would also like to thank all the fellow members of the student room for kindly enduring all my questions and some listening tests and those friends who made life in Eindhoven so enjoyable. To my friends my apologies for whom I did not have as much time as I would have liked during this work.

I am grateful to my girlfriend for visiting me once in Netherland and for encouraging me to study abroad, even when she missed me a lot during these months. I would also like to thank my small brother to hold me on during this years and my old brother for his recommendations with my English. Finally, I wish to thank my parents to whom I am grateful for their constant care and support in every aspect of my life.

A special memory to all the dear beings who have left us during this way.

Contents

Abstract i

Acknowledgments iii

List of figures vii

List of tables x

1 Introduction 1

1.1 Effects of Reverberation on Speech 1

1.2 Overview of this thesis 2

2 Room Impulse Response 5

2.1 Introduction 5

2.2 Room Acoustics 6

2.2.1 Analyzing Room Acoustics 7

2.2.2 Room Transfer Function 8

2.2.3 Room Acoustic Modeling Techniques 9

2.3 Room Acoustical Atributes 10

2.3.1 Early Decay Time 12

2.3.2 Reverberation Time 12

2.3.3 Reverberation Distance 13

2.3.4 Energy Decay Curve 15

2.4 Effects of reverberation on speech 18

2.5 Statistical RIR model 19

3 Speech Dereverberation Algorithm **23**
3.1 Introduction 23
3.2 The reverberant signal 24
3.3 Spectral subtraction 27
3.4 Implementation 28
3.5 Estimation of reverberant PSD 30

4 Real Time Implementation **31**
4.1 Introduction 31
4.2 PortAudio 32
4.2.1 Callback Function 33
4.2.2 Stream Management 35
4.2.3 Error Handling 37
4.3 C++ Interface 37
4.4 Process Function 41

5 Experiments and discussions **47**
5.1 Introduction 47
5.2 Matlab implementation 48
5.3 Objective measures for performance evaluation 50
5.3.1 Reverberation Reduction 51
5.3.2 Speech Distortion 51
5.3.3 Reverberant Estimation Distance 52
5.4 Matlab Results 53
5.5 Real Time Implementation Results 54
5.6 Discussion 57
5.7 Improvement of the algorithm 58
5.7.1 Comparison with different Ts 62

6 Conclusions **65**

A List of abbreviations, notations and used excerpts **69**
A.1 List of abbreviations 69

A.2 List of symbols 71

B Short-Time Fourier Transform **73**

C C files **77**

C.1 PortAudio header 77

C.2 DSPLabDlg C++ 81

C.3 Process C++ 87

D Matlab files **95**

D.1 STFFT 95

D.2 ESTIMATION 96

D.3 SIGNAL 96

D.4 ISTFFT 97

Bibliography **99**

List of Figures

2.1 Room Impulse Response obtained with the image-source method . . . 11
2.2 Spatial dependence of direct and reverberant energy densities, w_d and w_r, respectively. 14
2.3 Energy Decay Curve (EDC). 17
2.4 Statistical model vs Image-source method. 21
2.5 Energy of the RIR of the statistical model, image-source method and the energy envelope of (2.15). 22

3.1 Overview of the algorithm. 29

4.1 Dialog Box 32
4.2 Real-Time Implementation 38
4.3 Processes relations 44
4.4 Relations between files and functions 45

5.1 Anechoic (a), reverberated (b) and processed (c) signals 49
5.2 Spectrograms 49
5.3 Speech Distortion as a function of distance 54
5.4 Reverberant Estimation Distance as a function of distance 55
5.5 Reverberation Reduction as a function of distance 56
5.6 Real time implementation results 57
5.7 Energy Decay Curve 58
5.8 Energy Decay Curve with a distance equal to 0.5 m 59
5.9 Speech Distortion improved algorithm 60

5.10 Reverberant Estimation Distance improved algorithm 61
5.11 Reverberation Reduction improved algorithm 62
5.12 Speech Dereverberation comparison 63
5.13 Reverberant Estimation Distance comparison 64
5.14 Reverberation Reduction comparison 64

B.1 Time-frequency plane. 75
B.2 Time-bandwidth product. 75

List of Tables

5.1 Comparison between $T = 40\ ms$ and $T = 60\ ms$ for short distances . 63

Chapter 1

Introduction

The communication of information via speech from one person to another involves many steps which are collectively known as the speech train [1]. These steps, in a "natural" context, include production of acoustic pressure patterns at the talker's mouth, transmission of the acoustic pressure to the listener's ears, and the perception of this acoustic pressure pattern by the listener as meaningful speech. In this context, interferences such as competing noise sources may occur at the transmission step which may hinder the listener's ability to perceive speech. Another source of interference may be reverberation, in which delayed copies of the speech acoustic waveform, called echoes, are received at the ears in addition to direct speech.

In this thesis, we study the problem of speech dereverberation. The dereverberation of speech may be useful under a variety of situations, including the enhancement of speech recorded monaurally in reverberant enclosures such as rooms, and the enhancement of speech transmitted over telephone lines which has become reverberant either before transduction by microphone, or during transmission over the telephone network.

1.1 Effects of Reverberation on Speech

In general, acoustics signal radiated within a room are linearly distorted by reflections from walls and other objects. These distortions degrade the fidelity and intelligibility of speech, and the recognition performance of automatic speech recognition

systems. Early room echoes mainly contribute to coloration, or spectral distortion, while late echoes contribute noise-like perceptions or tails to speech signals.

The Reverberation Time (RT) of a room specifies the duration for which a sound persists after it has been switched off. The persistence of sound is due to the multiple reflections of sound from the various surfaces within the room [2]. One important effect of reverberation on speech is overlap-masking, i.e. phonemes are smeared over time, thereby overlapping following phonemes [3]. Consequently, the RT of a room provides a measure of the listening quality of the room. This is of particular importance in speech perception where it has been noted that speech intelligibility reduces as the RT increases. The effect of reverberation is most noticeable when speech recorded by a microphone is played back via headphones.

Reverberation effects are more severe for hearing impaired listeners children, and the elderly than for normal listeners. For hearing impaired listeners, the reception of reverberant signals via the microphone of a hearing aid exacerbates the problem of listening in challenging environments.

1.2 Overview of this thesis

In this thesis, it is develop a real-time implementation of the algorithm presented in [3]. Also a small improvement of the algorithm for the situation when the source and the microphone are close.

This thesis is structured as follows. An overview of some of the theory behind the algorithm used in this work is given in chapter 2. Chapter 3 deals with the algorithm applied for the dereverbertion of speech. In chapter 4, it is described the real-time implementation. In chapter 5, the results are studied and discussed. Conclusions are drawn and directions for future work are given in chapter 6.

Chapter 2 describes the theoretical basis for the algorithm used in the rest of this work.

Room acoustics analysis techniques, room transfer function theory and room modeling techniques are reviewed followed by theoretical background of reverberation time and the methods used in its measurements and estimation.

Then it is discussed the effect of reverberation on speech and it is described the

statistical Room Impulse Response (RIR) model for late reflections (reverberation).

In **chapter 3**, it is discussed the algorithm used to dereverberate the speech.

First, the theory used to obtain the reverberant signal model is presented and it is explained the technique for the enhancement of noisy speech degraded, spectral subtraction. This is followed by the implementation of the algorithm and the estimation of reverberant Power Spectral Density (PSD).

Chapter 4 describes the real time implementation of the algorithm presented in chapter 3, which has been implemented by the author in the course of this study.

In **chapter 5**, a series of measurements are performed in order to evaluate the algorithm performance and to verify its results.

The simulation in Matlab is explained and the results obtained are given and analyzed. Then, an improvement of the algorithm for the closer distance between the source and the microphone is explained and also it is presented and discussed the results for this improvement.

The last part of this chapter is used to present the results on the real time implementation with subjective measures.

In **chapter 6**, conclusions are given on the results acquired using the real time implementation as well as for the improvement proposed over the algorithm.

Chapter 2

Room Impulse Response

2.1 Introduction

The impulse response of a linear system is the waveform that appears at the output of a system when a unit impulse (Dirac delta function) is presented at the input. The output $y(t)$ for arbitrary input $x(t)$ is obtained by convolving the input with the impulse response $h(t)$.

$$y(t) = x(t) \star h(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)\,d\tau \tag{2.1}$$

The lower limit for integration is set to zero for physically realizable causal systems. The system is said to be BIBO stable (Bounded-Input, Bounded-Output) if the output is bounded for every bounded input. If the impulse response $h(t)$ does not change with time, the system is time-invariant. Finally, if the superposition principle holds, the system is linear. Systems that fulfill the two previous conditions are called Linear Time-Invariant (LTI) systems. When moving to the world of discrete-time systems, the convolution integral in (2.1), changes into a convolution sum,

$$y(n) = x(n) \star h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{2.2}$$

where n is the discrete time index. The output sequence $y(n)$ is thus related to

the input sequence $x(n)$ by a linear combination of the past and future values, the weights being given by the unit sample response $h(n)$. For causal systems the lower limit for the sum is zero.

The Room Impulse Response (RIR) is defined as the acoustic pressure pattern induced at a particular point in a room, in response to a pressure impulse of unity magnitude at another point in the room. Between this to points, the room's acoustical properties, can be seen as a linear, time-invariant filter which produces a known acoustical output for every input. The acoustic echo property of a room is characterized by the reverberation time T_{60}, which is the time for the acoustical reverberant energy to decay 60 dB relative to the direct path energy.

Room acoustical attributes, such as reverberation time T_{60} can be derived from Energy Decay Curve (EDC) of a room. To compute this Energy Decay Curve room acoustic modeling techniques are necessary. Computer simulation allows to estimate an impulse response of a room which has not yet been built. Simulation requires a knowledge of the geometry and reflective characteristics of the surfaces in a room. The image method has been chosen to simulate room impulse responses in this master thesis. Also, a statistical model, which is presented at the end of this chapter, is used in the algorithm explained in chapter 3.

This chapter reviews some of the theory behind the algorithm developed in this work. Room acoustics analysis techniques, room transfer function theory and room modeling techniques are reviewed first, followed by theoretical background of reverberation time and the methods used in its measurements and estimation. The statistical Room Impulse Response (RIR) model for late reflections (reverberation) is introduced at the end of this chapter, after having talked about the effect of reverberation on speech.

2.2 Room Acoustics

The characteristics properties of a particular acoustic environment depend on the frequency components of the sound of interest; the various techniques which are used for the analysis of these properties are discussed below. It is also introduced the Room Transfer Function (RTF), which describes mathematically, the sound

propagation is introduced. By solving the wave equation, it can be obtained an impulse response from a source to a listener, but it can seldom be performed in an analytic form. Therefore, the solution must be approximated and it is explained three different approaches in computational modeling of room acoustics.

2.2.1 Analyzing Room Acoustics

There are many different techniques for analyzing the acoustics of a room and, in general, each of these techniques applies to a different frequency range of the audible spectrum; no single analytic or numerical tool can currently model the entire audible frequency range between 16 Hz and 15 kHz [4] and [5]. The audible spectrum can be divided into four regions, over each of which is appropriated a different analytical tool [6].

Very Low Frequencies

When the frequency of a sound source is below $f_\omega = \frac{c}{2L}$, where c is the speed of sound, and L is the largest dimension of the acoustic environment, there is no resonant support for the sound in the room. This frequency band can be analyzed using non-harmonic solutions to the wave equation (see equation (2.4)).

Comparable Room Dimensions and Wavelength: Wave Acoustics

The next region corresponds to frequencies for which the wavelength of the sound in consideration is comparable to the dimensions of the room. This region spans from the lowest resonant mode to the Schroeder cut-off frequency [4]:

$$f_g \approx 2000\sqrt{\frac{T_{60}}{V}}\ (Hz) \tag{2.3}$$

This is the frequency for which the separation of two adjacent frequency curve maxima is comparable with the average bandwidth of the room resonances, in other words the frequency at which modal overlap becomes important. In this region, wave acoustics are applicable for describing the acoustical properties of a room. Wave acoustics assume a harmonic sound source and are based on solutions of the

wave equation of equation (2.4). It is in this region where is valid the statistical model of the Room Impulse Response, which is explained in the last point of this chapter. For instance, in a small living room with dimensions $3\ m \times 5\ m \times 7\ m$ and a reverberation time of 0.5 *sec*, statistical theory would be relevant above 138 Hz [6].

High Sound Frequencies

The transition region consists of the frequency components between f_g and, approximately, $4f_g$, where f_g is given by equation (2.3). In this region, the wavelengths are often too short for accurate modeling using wave acoustics, and too long for geometric acoustics. Thus, in general, it is employed a statistical treatment. The boundary frequencies of this band for typical acoustic environments can be calculated using equation (2.3); for example, a small recording studio room of dimensions $6\ m \times 6\ m \times 3\ m$ and $T_{60} = 0.5\ sec$ gives a transition region of 138 Hz to 552 Hz, whilst a car compartment of volume $V = 2.5\ m^3$ and $T_{60} = 0.05\ sec$ gives a region of 282 Hz to 1131 Hz [4].

Very High Sound Frequencies: Geometrical Room Acoustics

At very high sound frequencies it is applied geometrical room acoustics. As in geometrical optics, geometrical room acoustics employs the limiting case of vanishingly small wavelengths. This assumption is valid if the dimensions of the room and its walls are large compared with the wavelength of the sound; a condition which is met for a wide-range of audio frequencies in standard rooms. Hence, in this frequency range, specular reflections and the sound ray approach to acoustics prevail. Geometrical acoustics usually neglect wave related effects such as diffraction and interference.

2.2.2 Room Transfer Function

In principle, any complex sound field can be considered as a superposition of numerous simple sound waves, e.g. plane waves [7], and their propagation within a room can be considered linear if the medium in which the waves travel is assumed to be

homogeneous, at rest, and independent of wave amplitude. Under these assumptions, when the acoustical power of the sound source is doubled, the sound pressure produced by the sound source at some distant point doubles in value; this property has its mathematical analogue in the linearity of the wave equation [7] and [8]:

$$\frac{\partial^2 p}{\partial t^2} = c^2 \nabla^2 p = c^2 \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} \right) \tag{2.4}$$

where p denotes the sound pressure and c the speed of sound.

In practice, the effects of temperature variations cause the velocity of sound to be a function of time and spatial position. Furthermore, the air is not completely at rest due to temperature differences and air conditioning and, therefore, solutions of the wave equation (2.4) will contain nonlinearities. However, the effects of these inhomogeneities are so small that they can be ignored.

It can be shown, using the solutions of the wave equation (2.4) for sound fields in a closed space that the transfer function between a sound source and receiver, with spatial coordinates denoted by position vectors $\mathbf{r}_s$ and $\mathbf{r}_0$ respectively, can be expressed in terms of the resonant frequencies, ω_i, and their eigenfunctions, $P_i(\mathbf{r})$, as [9]:

$$H_{(\mathbf{r}_s,\mathbf{r}_0)}(\omega) = G \sum_{i=1}^{\infty} \frac{P_i(\mathbf{r}_s) P_i(\mathbf{r}_0) j\omega}{\omega^2 - \omega_i^2 + \delta_i^2 - 2j\delta_i\omega_i} \tag{2.5}$$

where ω is the angular frequency, δ_i is the damping constant (corresponding to the Q-factor), and G is a gain constant. The parameters ω_i and δ_i are independent of the source and receiver positions, and their values are determined by the room size, wall reflection coefficient, and room shape. The frequency response of the room described by equation (2.5) leads to a Room Impulse Response, $h_{(\mathbf{r}_s,\mathbf{r}_0)}(t)$.

2.2.3 Room Acoustic Modeling Techniques

There are three different approaches in computational modeling of room acoustics [10]:

- Wave-based methods

- Ray-based methods
- Statistical models

Wave-based Methods

The most accurate results can be achieved with this method. An analytical solution for the wave equation can be found only in rare cases such as rectangular room with rigid walls. Therefore numerical wave-based methods must be applied. In this method, the most difficult part is the definition of the boundary conditions. Typically a complex impedance is required.

Ray-based Methods

The ray-based methods are the most often used modeling techniques. They are based on geometrical room acoustics, in which the sound is supposed to act like rays. This assumption is only valid when the wavelength of sound is small compared to the area of surfaces in the room and large compared to the roughness of surfaces.

The most commonly used ray-based methods are the ray-tracing and the image-source method. The basic difference between these methods is the way the reflection paths are typically calculated.

Statistical Models

The statistical modeling methods, are mainly applied in prediction of noise levels in coupled systems in which sound transmission by structures is an important factor. A statistical model for the Room Impulse Response is used in the algorithm presented in the next chapter.

2.3 Room Acoustical Atributes

The room transfer function of (2.5) yields a corresponding room impulse response. This impulse response can be divided into three components as can be observed in Figure 2.1; direct sound, early reflections and late reflections. Early reflections are

not perceived as a separate sound to the direct sound so long as the delay does not exceed a certain limit; although the direct sound is followed by multiple reflections, which would be audible in isolation, the first-arriving wavefront dominates many aspects of perception independent of the energy of these early reflections. As such, these early reflections are actually perceived to reinforce the direct sound and are therefore considered useful with regards to speech intelligibility. Reflections which arrive with larger delays w.r.t. the arrival of the direct sound are perceived either as separate echoes, or as reverberation; as such late reflections impair speech intelligibility.

Rooms may contain a large number of sources with different position and directivity patterns, each producing an independent signal. Fortunately , the statistical properties of late reflections do not changes significantly as a function of position. Thus, a point to point impulse response does characterize the late reflections of the room, although the early echo pattern is dependent on the positions and directivities of the source and receiver [11].

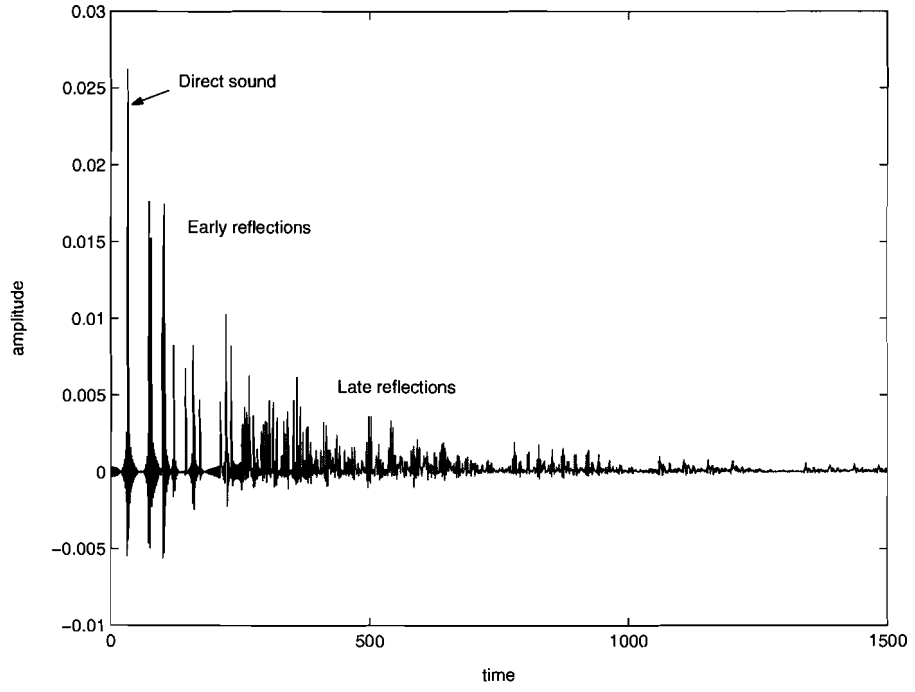

Figure 2.1 Room Impulse Response obtained with the image-source method

Reverberation is a phenomenon which play a major role in every aspect of room acoustics, and which yields the least controversial criterion for the judgement of the acoustical qualities of a room.

2.3.1 Early Decay Time

Early Decay Time (EDT) is defined as the time interval required for the sound energy level to decay 10 dB after excitation has stopped. To enable direct comparison with reverberation time, the result is scaled by a factor of 6. For an ideal exponential decay in a diffuse field, the expected value of the EDT equals T_{60}. Determination of T_{10} is performed in a similar manner with T_{60}, substituting 0 dB and 10 dB for the evaluation interval limits. EDT takes into account the subjective importance of the early part of the reverberation process. Due to local differences in early reflections, measured EDT values have greater variance than corresponding T_{60} values. Also, the short integration period makes the calculation of EDT sensitive to the accurate determination of the direct sound.

2.3.2 Reverberation Time

Reverberation Time (RT) is the time interval, T_{60}, in which the reverberating sound energy, due to decaying reflections, reaches one millionth of its initial value, i.e. the time interval it takes for the reverberation level to drop by 60 dB [7].

In a diffuse sound field, the ideal room decay process exhibits a purely exponential decay curve

$$p^2(t) = p_0^2 e^{-kt} \tag{2.6}$$

where p_0 is the sound pressure at zero time, and $p(t)$ is the sound pressure at time t. The decay parameter k is related to the room properties by

$$k = \frac{cA}{4V} \tag{2.7}$$

where $c = 340\ m/s$ is the velocity of sound, A the total absorption area in the room, and V the volume of the room. According to Sabine, the reverberation time T_{60} is

defined as

$$T_{60} = \frac{0.16V}{A} \tag{2.8}$$

The coefficient 0.16 is empirically determined, and shows some variance with temperature. Combining equations (2.7) and (2.8), the parameter k is related to the reverberation time by

$$k = \frac{13.6}{T_{60}} \tag{2.9}$$

We can also define the reverberation time T_{60} related with the average damping constant, α, of the resonant modes in the room as:

$$T_{60} = \frac{3ln(10)}{\alpha} \tag{2.10}$$

Although, the started point of the equations (2.9) and (2.10) is different, a relation between the average damping constant (α) and k can be observed: $2\alpha \approx k$.

2.3.3 Reverberation Distance

If a diffuse sound source continuously supplies acoustic energy into a room, there will be stationary sound energy throughout the room. Since reverberation can be considered as a statistical process with reflections of a large number of surfaces across the entire room, the energy density throughout the room due to reverberation will be constant. This is since, with a diffuse sound source, there is no reason why the reverberant energy in one part of a closed room should be higher than that in another part, save exceptional cases, due to the large number of reflections; the room is said to be filled with a diffuse sound field [7]. However, since a spherical sound wave is attenuated with distance as a result of the spherical spread of sound pressure, the energy density due to direct sound between the source and a point in the room falls off with distance from the source, as shown in Figure 2.2. The distance at which the steady state reverberant energy equals the direct sound energy is called the reverberation distance or radius and, for a point sound source, is given by:

$$r_d = 0.1\sqrt{\frac{V}{\pi T_{60}}}\ (m) \tag{2.11}$$

where V and T_{60} are, respectively, the volume (in m^3) and the reverberation time (in seconds) of the room. If an observer is within the reverberation distance of a source, the direct energy is greater than the reverberant energy while, if the observer is outside the reverberation distance, the reverberant energy will be dominant. The intelligibility of speech, for example, then depends greatly on whether the observer is near the source, or far from the source, and explains why reverberation has negligible effect on intelligibility when using normal telephones or equipment where the microphone can be placed near to the sound source. In a typical small office of volume 40 m^3, for example, with $T_{60} = 0.5$ sec, the reverberation distance is 0.5 m.

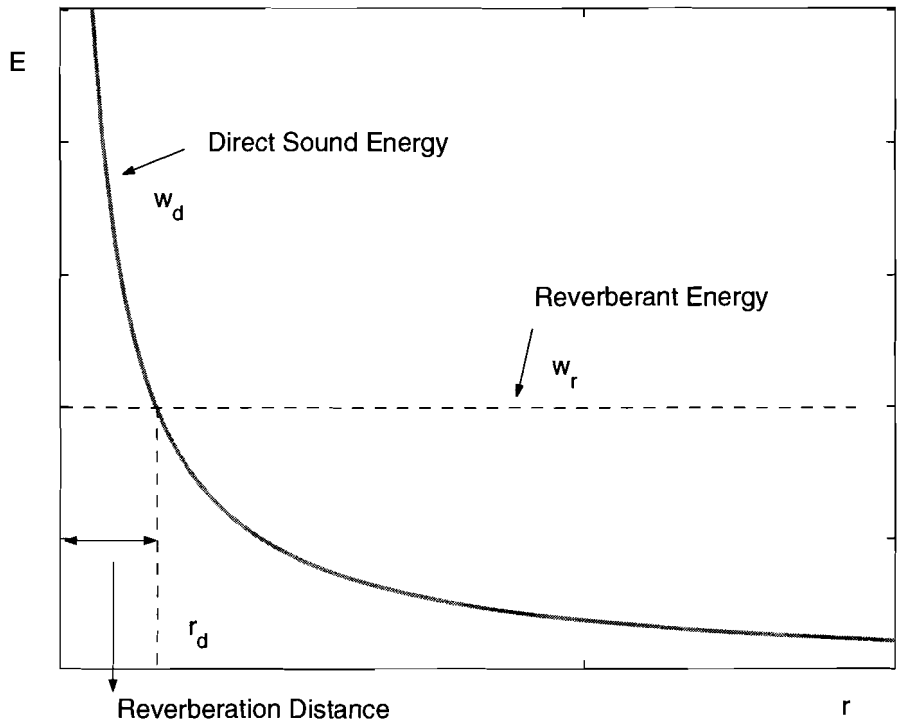

Figure 2.2 Spatial dependence of direct and reverberant energy densities, w_d and w_r, respectively.

2.3.4 Energy Decay Curve

The simplest way of calculating the reverberation time from a measured impulse response would be that of finding the time instant when the sound energy level falls below 60 dB from the peak level. Simply measuring the time interval directly from the squared pressure is usually not accurate enough. The usual procedure is to use linear regression to fit a straight line to the data, preceded by the Schroeder method.

Properties of decay curves

In practice, the properties of decay curves are not simply exponential. Background noise, natural room modes, and non-diffuse sound fields are the main factors for this feature. The exponential decay is always, in practice, obstructed by a constant level of background noise at some point. Natural room modes may cause the decay curve to warble, especially at low frequency bands where the mode density is low. In cases of an uneven distribution of absorption, the decay curve may bend from one slope to another, resulting in multiple levels of decay. This is more or less the case in almost all practical room spaces. Since the calculation of reverberation time is based on an assumption of exponential decay, it is not necessary to acquire a full 60 dB of decay by measurement, but a smaller portion of the available dynamic range may be evaluated and the result simply scaled to 60 dB of decay. By using a normalized decibel scale on the amplitude axis, the evaluation and scaling can be performed simply by finding the slope of the decay curve within a certain interval, and extending it down to -60 dB.

Decay slope

Calculating the reverberation time from a single measurement of the decay curve obtained with bandpass-filtered noise as an excitation, is not very accurate. This is due to the random nature of the measured signal. A method frequently used to minimize the effect of the fluctuations in decay curves on the measured reverberation time values is to repeat the reverberation experiment many times and to average the reverberation times (or decay rates). Since averaging over many realizations is quite laborious, a more elegant method would be preferred. The Schroeder method

yields a decay curve, in a single measurement, that is identical to the average over infinitely many decay curves that would be obtained using different noise signals [12].

$$E\{y^2(t)\} = \int_t^\infty |h(\tau)|^2 \, d\tau = \int_0^\infty |h(\tau)|^2 \, d\tau - \int_0^t |h(\tau)|^2 \, d\tau \tag{2.12}$$

where $y^2(t)$ is the squared received signal, $h(\tau)$ is an impulse response of the whole system. The ensemble average is indicated by $E\{\}$. This implies that the ensemble average of all individual decay curves can be acquired in a single measurement of a impulse response.

The Energy Decay Curve (EDC) equation more used and common can be defined as [11]:

$$EDC(t) = \int_t^\infty |h(\tau)|^2 \, d\tau \tag{2.13}$$

where $h(t)$ is a impulse response of the room which may be narrowband filtered to yield the EDC for some particular frequency. The integral (often called Schroeder integral) computes the energy remaining in the impulse response after time t.

In practice the upper limit of integration in equation (2.13) is set to a time instant at which the decay curve is still a little bit above the noise floor. The practical formula for obtaining the ensemble average of all decay curve then becomes [13]

$$EDC(t) = N \int_t^{T_i} h^2(\tau) \, d\tau \tag{2.14}$$

where N is a constant proportional to the Power Spectral Density of the noise on the frequency range measured and T_i is the upper limit of integration. According to [13], the choice of T_i should be made so that T_i is close to the point where the decaying signal "dives" into the noise floor. ISO 3382 standard specifies that T_i should be set to a point where the impulse response is 5 dB above the noise floor.

Converting levels to a decibel scale, the decay can be described by a linear equation $y = ax + b$, where the decay curve of slope a is at level y at time x. Offset b is usually equal to zero, as the curve is commonly normalized to begin at a level of 0 dB, thus passing through the origin.

Figure 2.3 shows an example of the Schroeder integration curve calculated from

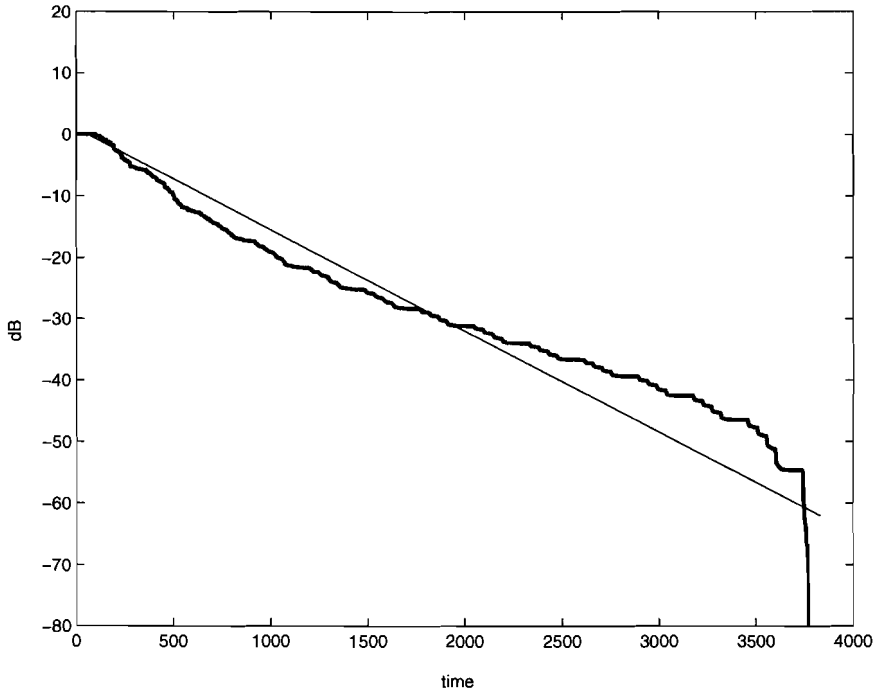

Figure 2.3 Energy Decay Curve (EDC).

a room impulse response. An example of straight-line fits to the first 30 dB.

2.4 Effects of reverberation on speech

Speech in enclosures such as rooms is subject to reverberation. Direct sound from the speaker to the listener is followed by reflections from walls and other surfaces which may arrive at delays from a few milliseconds to a few seconds [14]. The perceptual effects of reverberation are usually classified according to the delay time. Echoes which occur with delays up to a few tens of milliseconds modify the short-time spectrum by introducing periodically spaced nulls into the speech spectrum. This effect, known as spectral colouration, is particularly apparent in small rooms with highly reflective walls, because the echoes produced have high amplitudes and small delay times. Beyond a delay of perhaps 50 m, echoes may be perceived as distinct copies of the direct path speech and cause temporal rather than spectral distortion. Many studies have demonstrated that reverberation degrades the intelligibility of speech:

- Decrease overall understanding of speech
- Vowels interrupt the understanding of lower loudness consonants. Vowels tend to be $10 - 15$ dB louder than consonants.
- Silent intervals between syllables, sounds, and words are filled with reflected energy resulting in a smearing effect of the sound.

In a "live" situation a normal listener is able to use his binaural hearing and possibly other screening methods to compensate for reverberation. In a normal room, intelligibility between talkers and listeners of normal hearing is not affected severely by reverberation (at least in the absence of noise). However, whenever a single microphone is used in a room to record speech, the binaural hearing advantage is lost. In this case, reverberation reduces intelligibility. Most studies have found that long time echoes affect intelligibility more severely than shorter time echoes [14]. For a binaural environment, studies have shown that the earliest echoes actually improve intelligibility because they effectively increase the energy of the speech

signal. As long as the delay is less than the integration time of the ear, some of the reverberant energy may be useful. As the delay increases, the likelihood of interference between a phoneme and a succeeding phoneme increases and less of the reverberant energy is useful.

Reverberation effects are more severe for hearing impaired listeners, children, and the elderly than for normal listeners. For hearing impaired listeners who had bilaterally equal hearing, reverberation affected their speech perception under binaural hearing conditions where normal hearing listeners were not affected.

2.5 Statistical RIR model

The statistical RIR model for late reflections is presented after seeing some of the room acoustical attributes and how the reverberation affects to the speech. This model will help to get an approximation for the reverberant signal in order to estimate its Power Spectral Density.

The Room Impulse Response is modeled as the outcome of a non-stationary random process [3]:

$$h(t) = \begin{cases} b(t)e^{-\alpha t} & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2.15}$$

where $b(t)$ is a zero-mean Gaussian stationary noise, considered in first approximation to be white, and α is linked to the reverberation time T_{60} through:

$$\alpha \triangleq \frac{3ln(10)}{T_{60}} \tag{2.16}$$

The energy envelope of the RIR can be expressed as

$$E\{h^2(t)\} = \sigma^2 e^{-2\alpha t} \tag{2.17}$$

where σ^2 denotes the variance of $b(t)$ and $E\{\}$ denotes ensemble averaging over h, i.e. over different realizations of the stochastic process in (2.15).

It can be shown that different realizations of this stochastic process are obtained by varying the position of the receiver with a fixed source position, or by varying the

position of the source with a fixed receiver position (or of course by varying both positions). Notice that the same stochastic process will be observed, irrespective of position, provided that the time origin is defined in with reference to the signal emitted by the source and not w.r.t. the arrival time of the direct sound at the receiver.

The RIR can be split into two components, $h_d(t)$ and $h_r(t)$ so that

$$h(t) = \begin{cases} 0 & t < 0 \\ h_d(t) & 0 \leq t < T \\ h_r(t) & t \geq T \end{cases} \tag{2.18}$$

The value T is chosen such that $h_d(t)$ consists of the direct signal and a few early echoes and $h_r(t)$ consists of all later echoes, i.e. late reverberation. T usually ranges from 40 to 80 ms.

The next picture (Figure 2.4) shows the Room Impulse Response that has been just explained compared with the RIR created with the image-source method. It is important to remember that the statistical model of the RIR is only valid in the region from the lowest resonant mode to the Schroeder cut-off frequency.

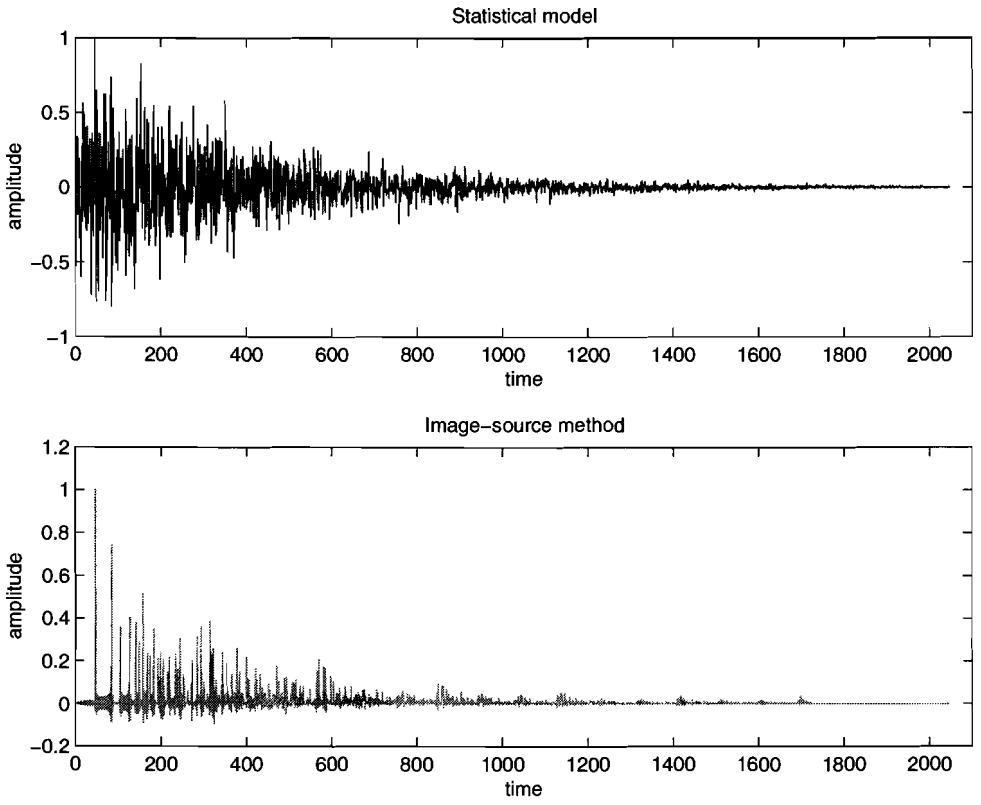

Figure 2.4 Statistical model vs Image-source method.

Figure 2.5 show the mean energy of both RIRs showed in the Figure 2.4 and compare them with the energy envelope of the statistical model, calculated with $E\{h^2(t)\} = \sigma^2 e^{-2\alpha t}$.

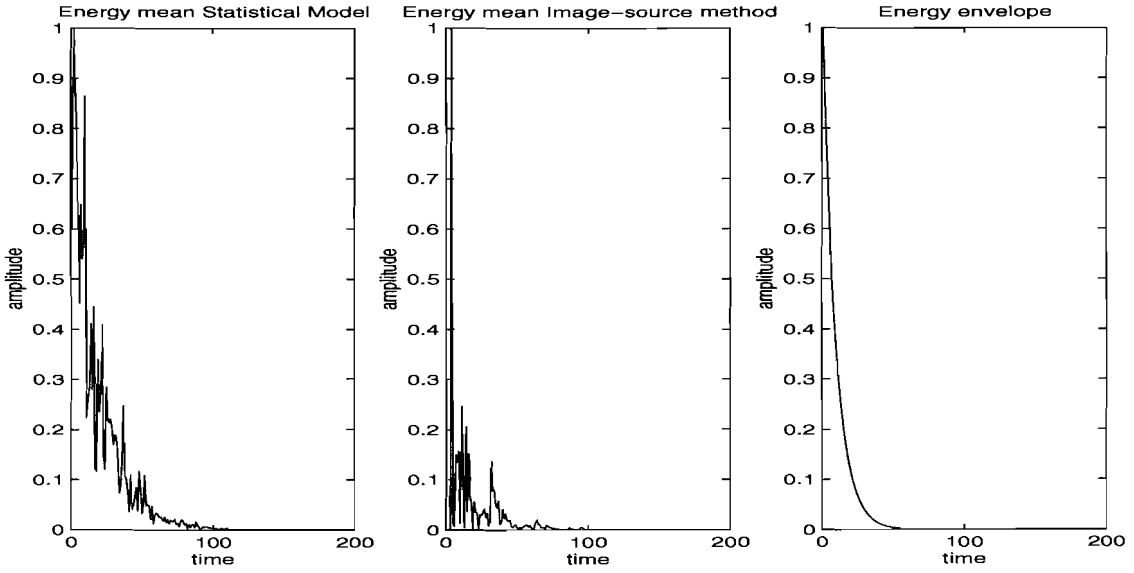

Figure 2.5 Energy of the RIR of the statistical model, image-source method and the energy envelope of (2.15).

Chapter 3

Speech Dereverberation Algorithm

3.1 Introduction

The algorithm which is next presented applies the signal processing techniques to improve the quality of speech distorted in an acoustic environment.

The algorithm dereveberates the signal with the aim to avoid the problems that presents receiving the speech signal with late reflections, as it has been explained in chapter 2, section 5. One of the used techniques for the enhancement of noisy speech degraded with uncorrelated additive noise is the spectral subtraction. This technique is used to eliminate the reverberation. For that is necessary to estimate the Power Spectral Density. The PSD is defined like the Fourier Transform of the autocorrelation in this case, the autocorrelation of the reverberant part of the signal, which is obtained from the model of the Room Impulse Response, explained in the last section of the previous chapter.

The outline of this chapter is as follows. The theory used to obtain the reverberant signal model is explained in the first point, followed by the technique for the enhancement of noisy speech degraded, spectral subtraction. The implementation of the algorithm is explained in the third point of this chapter and the estimation of reverberant Power Spectral Density (PSD) is introduced at the end of this chapter.

3.2 The reverberant signal

The reverberant signal results from the convolution of the anechoic speech signal $s(t)$ and the causal RIR $h(t)$ [15] and [3]:

$$x(t) = \int_{-\infty}^{\infty} s(\theta)h(t-\theta)\,d\theta \tag{3.1}$$

If we use

$$h(t) = b(t)e^{-\alpha t} \text{ for } t \geq 0, \tag{3.2}$$

where $b(t)$ is is a white zero-mean Gaussian stationary noise as is explained in chapter 2, the equation (3.1) can be expressed as:

$$x(t) = e^{-\alpha t}\int_{-\infty}^{t} s(\theta)b(t-\theta)e^{\alpha\theta}\,d\theta \tag{3.3}$$

because if $t-\theta \geq 0$ then $\theta \leq t$ that give us the limit of the integral.

The auto-correlation $r_{xx}(t,t+\tau) = E_x\{x(t)x(t+\tau)\}$ of the reverberant signal x at time t and lag τ for a fixed source-receiver configuration is

$$r_{xx}(t,t+\tau) = E\left\{\int_{-\infty}^{t}\int_{-\infty}^{t+\tau} s(\theta)s(\theta')h(t-\theta)h(t+\tau-\theta')\,d\theta d\theta'\right\}$$

Using the equation (3.2) of the statistical model of the RIR described in the last section of chapter 2, the auto-correlation can be written:

$$r_{xx}(t,t+\tau) = E\left\{\int_{-\infty}^{t}\int_{-\infty}^{t+\tau} s(\theta)s(\theta')b(t-\theta)b(t+\tau-\theta')e^{-\alpha t}e^{-\alpha(t+\tau)}e^{\alpha\theta}e^{\alpha\theta'}\,d\theta d\theta'\right\}$$

Since convolution is a linear operation performed on the signals $s(t)$ and $b(t)$, the expected value of the integral is equal to the integral of the expected value. Thus,

$$r_{xx}(t,t+\tau) = \int_{-\infty}^{t}\int_{-\infty}^{t+\tau} E\left\{s(\theta)s(\theta')b(t-\theta)b(t+\tau-\theta')\right\}e^{-\alpha t}e^{-\alpha(t+\tau)}e^{\alpha\theta}e^{\alpha\theta'}\,d\theta d\theta'$$

Then, if s and b are considered to be independent random processes:

$$r_{xx}(t,t+\tau) = \int_{-\infty}^{t}\int_{-\infty}^{t+\tau} E\{s(\theta)s(\theta')\}E\{b(t-\theta)b(t+\tau-\theta')\}e^{-\alpha t}e^{-\alpha(t+\tau)}e^{\alpha\theta}e^{\alpha\theta'}\,d\theta d\theta'$$

$$= e^{-2\alpha t}\int_{-\infty}^{t}\int_{-\infty}^{t+\tau} E\{s(\theta)s(\theta')\}E\{b(t-\theta)b(t+\tau-\theta')\}e^{\alpha(\theta+\theta'-\tau)}\,d\theta d\theta' \qquad (3.4)$$

Using the theory described in chapter 2, in the statistical Room Impulse Response model section, it follows that

$$E\{b(t-\theta)b(t+\tau-\theta')\} = \sigma^2\delta(\theta-\theta'+\tau) \qquad (3.5)$$

where $\delta()$ denotes the Dirac function and σ^2 the variance of $b(t)$. Equation (3.4) leads to

$$\begin{aligned} r_{xx}(t,t+\tau) &= e^{-2\alpha t}\int_{-\infty}^{t}\int_{-\infty}^{t+\tau} E\{s(\theta)s(\theta')\}\delta(\theta-\theta'+\tau)\sigma^2 e^{\alpha(\theta+\theta'-\tau)}\,d\theta d\theta' \\ &= e^{-2\alpha t}\int_{-\infty}^{t} E\{s(\theta)s(\theta+\tau)\}\sigma^2 e^{2\alpha\theta}\,d\theta \\ &= e^{-2\alpha t}\int_{t-T}^{t} E\{s(\theta)s(\theta+\tau)\}\sigma^2 e^{2\alpha\theta}\,d\theta \\ &+ e^{-2\alpha t}\int_{-\infty}^{t-T} E\{s(\theta)s(\theta+\tau)\}\sigma^2 e^{2\alpha\theta}\,d\theta \end{aligned}$$

The autocorrelation at time t can be divided into two terms. The first term depends on the direct signal between time $t-T$ and t, whereas the second depends on the reverberation signal and is responsible for overlap-masking. Let us consider the spatially averaged auto-correlation at time $t-T$

$$r_{xx}(t-T,t-T+\tau) = e^{-2\alpha(t-T)}\int_{-\infty}^{t-T} E\{s(\theta)s(\theta+\tau)\}\sigma^2 e^{2\alpha\theta}\,d\theta \qquad (3.6)$$

We can now see that the auto-correlation at time t can be expressed as

$$r_{xx}(t, t+\tau) = r_{x_d x_d}(t, t+\tau) + r_{x_r x_r}(t, t+\tau)$$

with

$$r_{x_d x_d}(t, t+\tau) = e^{-2\alpha t} \int_{t-T}^{t} E\{s(\theta)s(\theta+\tau)\}\sigma^2 e^{2\alpha\theta}\, d\theta$$

$$r_{x_r x_r}(t, t+\tau) = e^{-2\alpha T} r_{xx}(t-T, t-T+\tau) \tag{3.7}$$

Another interpretation of the two terms of the auto-correlation is possible [15]: let $h(t)$ be split into two components, $h_d(t)$ and $h_r(t)$, as can be seen in the chapter 2, so that:

$$h(t) = \begin{cases} 0 & t < 0 \\ h_d(t) & 0 \leq t < T \\ h_r(t) & t \geq T \end{cases} \tag{3.8}$$

Let $x_d(t)$ and $x_r(t)$ be the results of the convolution of $s(t)$ by respectively $h_d(t)$ and $h_r(t)$. If T is relatively small compared to the length of the RIR, $x_d(t)$ is made up of the direct signal and a few echoes. As a first approximation it can be considered as being the direct signal, whereas $x_r(t)$ corresponds to all the later echoes, that is reverberation.

In practice the signals can be considered as stationary over periods of time that are short compared to the reverberation time T_r. This is justified by the fact that the exponential decay is very slow, and that speech is quasi-stationary, Let T_s be the time span over which the speech signal can be considered stationary, which is usually around $20 - 40\ ms$. We consider that $T_s \leqslant T \ll T_r$.

The spectral characteristic of a stochastic signal is obtained by computing the Fourier Transform of the autocorrelation function. That is, the distribution of power with frequency is given by the function:

$$\gamma(f) = \int_{-\infty}^{\infty} r(\tau) e^{-j2\pi f\tau}\, d\tau$$

Under these assumptions, the counterparts of (3.6) and (3.7) in terms of the short-term Power Spectral Densitys are approximately:

$$\gamma_{xx}(t,f) = \gamma_{x_d x_d}(t,f) + \gamma_{x_r x_r}(t,f) \tag{3.9}$$

$$\gamma_{x_r x_r}(t,f) = e^{-2\alpha T}\gamma_{xx}(t-T,f) \tag{3.10}$$

Therefore, we can estimate the Power Spectral Density of the direct signal by spectral subtraction of the late reverberant PSD.

3.3 Spectral subtraction

The presence of background noise in speech significantly reduces the intelligibility of speech. Degradation of speech severely affects the ability of person, whether impaired or normal hearing, to understand what the speaker is saying. Noise reduction or speech enhancement algorithms are used to suppress such background noise and improve the perceptual quality and intelligibility of speech. Even though speech is perceptible in a moderately noisy environment, many applications like mobile communications, speech recognition and aids for the hearing handicapped drive the effort to build more effective noise reduction algorithms for better performance.

Numerous techniques for the enhancement of noisy speech degraded with uncorrelated additive noise have been proposed in literature. Among them the spectral subtraction methods are the most widely used due to the simplicity of implementation and the low computational load, which makes them the primary choice for real-time applications. A common feature of this technique is that the noise reduction process can be related to the estimation of a Short-Time Spectral Attenuation factor. Since the spectral components are assumed to be statistical independent, this factor is adjusted individually as a function of the relative local A Posteriori Signal to Noise Ratio on each frequency. The A Posteriori SNR is defined as

$$SNR_{post}(t,f) \triangleq \frac{|X(t,f)|^2}{\gamma_{x_r x_r}(t,f)} \tag{3.11}$$

The gain function related to the Magnitude Subtraction (MS) is given by

$$G(t,f) = 1 - \frac{1}{\sqrt{SNR_{post}(t,f)}} \tag{3.12}$$

The estimate of the amplitude spectrum of the signal is given by

$$|\widehat{S}(t,f)| = G(t,f)|X(t,f)| \tag{3.13}$$

which we can rewrite using equations (3.11) and (3.12) as:

$$|\widehat{S}(t,f)| = |X(t,f)| - \sqrt{\gamma_{x_r x_r}(t,f)} \tag{3.14}$$

In all frames it is however possible that for some frequencies the estimated amplitude of the noise spectrum is larger than the instantaneous amplitude of the noisy speech spectrum $|X(t,f)|$. Since this could lead to negative estimates for the amplitude of the clean speech spectrum $|\widehat{S}(t,f)|$, for these frequencies the gain function $G(t,f)$ is usually put to zero (i.e. half-wave rectification) or equal to a small noise floor value as proposed in [16]. Applying above modification to the gain function in (3.12) results in the following gain function

$$G(t,f) = \begin{cases} 1 - \frac{1}{\sqrt{SNR_{post}(t,f)}} & \text{if } |\widehat{S}(t,f)| \geq \lambda |X(t,f)| \\ \lambda & \text{otherwise} \end{cases} \tag{3.15}$$

where λ denotes the threshold value.

3.4 Implementation

The signals are digitized with a sampling rate of 8 kHz. In the following, the discrete time and frame indices will be denoted by n and m, respectively, and the discrete frequency index by k. An overview of the complete algorithm is presented in Figure 3.1.

The reverberated signal $x(n)$ is decomposed into a Short-Time Fourier Transform (STFT) filter bank which is one of the ways that can be performed the Time Frequency (TF) analysis. For more information about the STFT see appendix

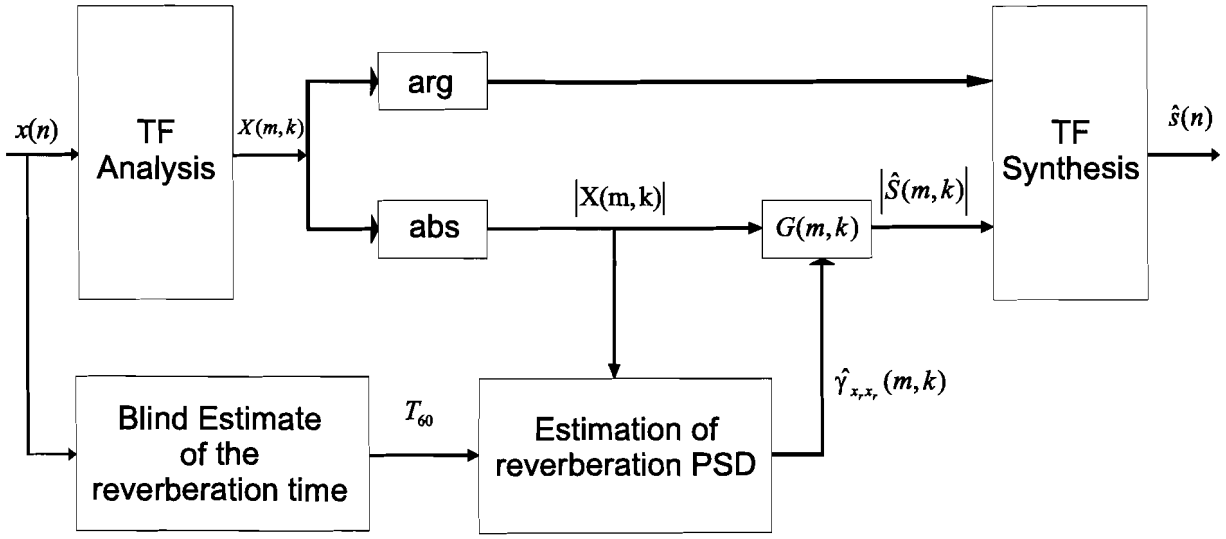

Figure 3.1 Overview of the algorithm.

B. Although this analysis results in a constant time-frequency bandwidth product, it performs well and has a low computational complexity. The analysis window is a 128 point Hamming window, and the overlap between two successive windows is set to 75%. Each frame is zero padded to 256 points to avoid wrap around errors. The power spectral density of the reverberation noise $\gamma_{x_r x_r}$ is estimated according to (3.10), as detailed in the next section. The square root of this estimate is then subtracted from the amplitude spectrum of the reverberated signal, $|X(m,k)|$, yielding an estimate of the amplitude spectrum of the dereverberated signal, $\widehat{S}(m,k)$. This is in practice realized by a short-term spectral attenuation, equivalent to spectral subtraction. This modification is detailed in the last section but using the discrete versions. The RIR model does not incorporate the direct delay caused by the source-receiver distance between the source and receiver. This will ensure that the received signals are time-aligned w.r.t. the direct speech signal. The threshold λ in (3.15) was set to 0.1, corresponding to a maximum attenuation of 20 dB. The estimated dereverberated signal $\widehat{s}(n)$ is then reconstructed from its estimated amplitude spectrum and the noisy phase, through the overlap-add technique [17].

3.5 Estimation of reverberant PSD

In order to estimate the reverberant Power Spectral Density we need to estimate the reverberation time of the room. As has been explained in the chapter two, for evaluating purposes we have used the average reverberation time measured directly from the synthetic Room Impulse Responses using Schroeder's method.

It can be seen that two terms need to be estimated to obtain an estimate of the reverberation PSD: the parameter of the model α (or equivalently the reverberation time T_{60}) and the PSD of the past reverberated signal.

Then

$$\hat{\gamma}_{x_r x_r}(m,k) = e^{-2\alpha T}\hat{\gamma}_{xx}(m-T',k) \tag{3.16}$$

and then, the Power Spectral Density of the past reverberated signal is estimated by averaging the periodograms of the signal. This is done by a running average according to:

$$\hat{\gamma}_{xx}(m,k) = \beta\hat{\gamma}_{xx}(m-1,k) + (1-\beta)|X(m,k)|^2 \tag{3.17}$$

If β is close to 1, the variance of the estimate of the PSD is small, but the equivalent averaging duration long. The averaging time should be kept small since the signal is non stationary. β should be chosen as a compromise so that the variance of the estimate is as small as possible while the assumption of quasi-stationarity is respected. The values are: $\beta = 0.9$ and $T' = \lfloor \frac{Tf_s}{32} \rfloor$, T' give us the delay in frames (not in samples), fs is the sample rate and 32 is shift of each window and can be calculated with: $32 = 128\frac{100-75}{100}$. Where 128 is the length of the Hamming window and 75 is the amount of overlap in %. For example, if we talk about 40 ms, this time in samples (with $f_s = 8000\ Hz$) is 320 but in frames if the overlap is 75% is 10.

Chapter 4

Real Time Implementation

4.1 Introduction

In order to be able to dereverberate the signal in real time and then, to be able to make measures in a real enviroment, the algorithm presented in the previous chapter has been developed in Visual C++. For that, first it was first implemented in Matlab to verify its performance.

The main difference from Matlab at the moment of implementing the algorithm in C++ is the necessity to divide the signal in blocks. In Matlab all the speech signal, which is saved in a wave file, is batch processed. In a real time environment it is not possible to wait until all the signal has been radiated and received. The signal must be processed at the same time it is received, so this implies the segmentation in blocks.

Some already written functions were used for C++ algorithm development. These functions can be found in a library called libtsp [18]. This package is a library of routines for signal processing which also includes a number of general purpose routines (useful for development programming).

The PortAudio library [19], which is explained in the first point of this chapter, has been used in order to develop the algorithm in C++. A Visual C++ dialog interface and the C++ code for the dialog are presented in the second point of this chapter. The interface let us to select the wave file to be radiated in the room and also to specify the Reverberation Time (T_{60}) and the parameter T. The Figure 4.1

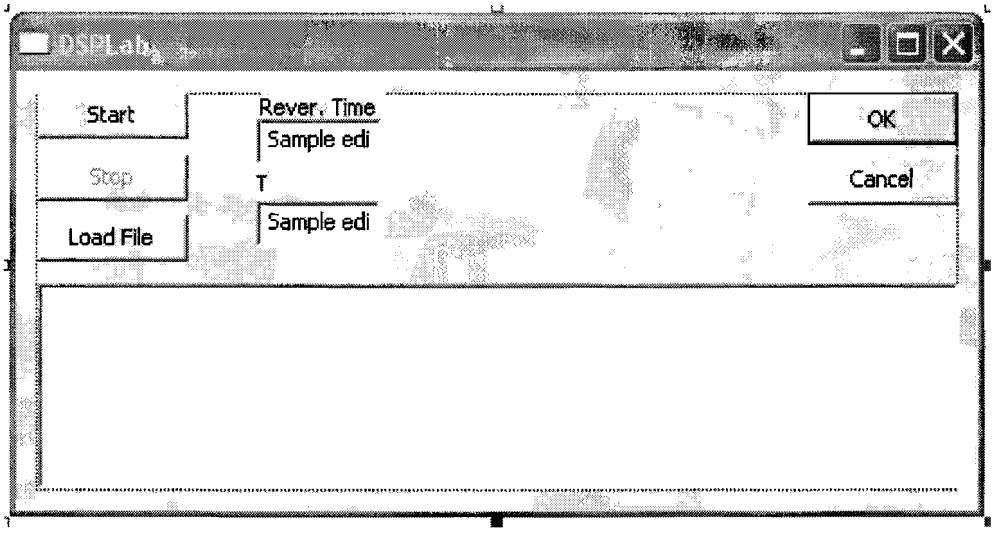

Figure 4.1 Dialog Box

shows an image of the dialog box.

In the last section of this chapter, the process developed to dereverberate the signal is presented.

4.2 PortAudio

PortAudio is a library that provides streaming audio input and output. It is a cross-platform Application Programming Interface (API) that works on Windows, Macintosh, Unix running OSS, SGI, BeOS, and perhaps other platforms. This means that a simple 'C' program to process or generate an audio signal can be written, and that program can run on several different types of computer just by recompiling the source code. To compile PortAudio for Windows, three audio-interfaces are supported:

- DirectSound API
- Windows MultiMedia Extensions (WMME) API
- Steinberg's Audio Stream Input/Output (ASIO) API

in our case ASIO has been chosen because is a low latency API

The PortAudio architecture includes two main abstractions: Audio Devices and Audio Streams.

Audio devices represent audio input and/or output ports on the host platform. The PortAudio API provides functions for enumerating available devices and querying them for properties such as available sampling rates, number of supported channels and supported sample formats.

Audio streams manage active audio input and output through at least one input device and one output device - streams may be full duplex or half duplex. A PortAudio implementation manages buffers internally and requests audio processing from the client application via a callback that is associated with a stream when it is opened.

A variety of sample formats are supported by PortAudio including 16 and 32 bit integer and 32 bit floating point. Where necessary PortAudio manages conversion between the requested buffer formats and those available natively. If requested, PortAudio can clamp out of range samples and/or dither samples to a native format.

Some steps must be followed to write a PortAudio application. Firstly, a callback function must be written, this function will be called by PortAudio when audio processing is needed. Next, the PortAudio library has to be initialized with Pa_Initialize(), this will trigger a scan of available devices which can be required later. The next step is to open a stream for audio I/O (Pa_OpenStream() or Pa_OpenDefaultStream()). Then, start the stream with Pa_StartStream(), the callback function will be now called repeatedly by Port Audio in the background. The callback function audio data can be read and/or written from the inputBuffer to the outputBuffer. After, stop the stream by calling the stop function Pa_StopStream(). When the library is no longer required a Pa_Terminate() function should be called.

4.2.1 Callback Function

To write a program using PortAudio, the "portaudio.h" file must be included. The "portaudio.h" which is included in the appendix C contains a complete description of the PortAudio functions and constants. Next, callback function must be written.

The callback function is called by the PortAudio engine whenever it has captured audio data, or when it needs more audio data for output, as is often called by an interrupt, or low level process, any complex system activities like allocating memory, or reading or writing file, or printf() should not be done. The callback function must return an int and accept the exact parameters specified in this typedef:

```
1  typedef int (PortAudioCallback)(
2                  void *inputBuffer, void *outputBuffer,
3                  unsigned long framesPerBuffer,
4                  PaTimestamp outTime, void *userData);
```

The callback function used in our real-time implementation is:

```
1  static int paPlaybackCallback(
2                  void *inputBuffer, void *outputBuffer,
3                  unsigned long framesPerBuffer,
4                  PaTimestamp outTime, void *userData)
5  {
6          paData          *pData = (paData*) userData;
7          float           *in = (float*) inputBuffer;
8          float           *out = (float*) outputBuffer;
9          int             finished=0;
10         unsigned int    i;
11
12         switch(pData->Mode & 1)
13         {
14         case 0:         /* Silence */
15                 for(i=0; i<pData->SamplesPerBufferOut; i++)
16                 {
17                         *out++ = 0;
18                 }
19                 break;
20         case 1:         /* Process Data */
21                 for(i=0; i<pData->SamplesPerBufferOut/
                       NUMBER_OF_OUTPUT_CHANNELS; i++)
22                 {
23                         *out++ = pData->pOutSamplesChan[i];
24                         *out++ = pData->pInSamplesChan[i];
25                 }
26                 for(i=0; i<pData->SamplesPerBufferIn; i++)
27                 {
```

```
28                          pData->pInSamplesChan[i] = *in++;
29                  }
30                  break;
31          }
32          PostMessage(pData->FHandle, WM_USER_PD, 0, 0);
33          return finished;
34  }
```

In case of silence the output is written by zero. In case of processing the signal two outputs are used; one to radiated the signal in the room (line 23) and the other to hear the output signal after dereverberation process (line 24). Also the signal received in the microphone is saved in the inputBuffer (line 28). The signals must be between -1.0 and $+1.0$.

The instruction in line 32 is a message which is interpreted as the order to start the function "LRESULT CDSPLabDlg::OnProcessData". this function, which is explained in the next section, contains the called to the function "Process", which implements the algorithm presented in chapter 3. The function "Process" is explained in section 4 of this chapter.

4.2.2 Stream Management

As has been explained before, PortAudio streams can be opened with either the Pa_OpenStream() or Pa_OpenDefaultStream() functions, both of which return an opaque handle to a PortAudioStream object.

Pa_OpenStream() allows specification of: input and output devices; sample format; number of channels and a device specific information block for input and output; sample rate; number and size of i/o buffers; a set of implementation defined stream flags; the user callback function and a user specified data pointer which is passed to the callback function. The following lines show how the stream used by us is opened.

```
1  err = Pa_OpenStream(
2    &stream,
3    Pa_GetDefaultInputDeviceID(),          /* default input device */
```

```
4    NUMBER_OF_INPUT_CHANNELS,                /*number of input channels
         = 1*/
5    paFloat32,                               /*32 bit floating point
         input*/
6    NULL,
7    Pa_GetDefaultOutputDeviceID(),           /*default output device*/
8    NUMBER_OF_OUTPUT_CHANNELS,               /*number of output channels
         = 2*/
9    paFloat32,                               /*32 bit floating point
         output*/
10   NULL,
11   SAMPLE_RATE,                             /*in the real enviroment
         16000 is used*/
12   FRAMES_PER_BUFFER,                       /*2048*/
13   0,                                       /*number of buffers, if
         zero then use default minimum*/
14   paClipOff,                               /*we won't output out of
         range samples so don't bother clipping them*/
15   paPlaybackCallback,                      /*specify our custom
         callback*/
16   pData );                                 /*pass our data through to
         callback*/
```

Pa_OpenDefaultStream() provides a simplified interface for opening a stream using the default device(s).

The Pa_StartStream() and Pa_StopStream() functions are used to begin and to end the processing on a stream. Pa_AbortStream() may be used to immediately abort playback on a stream rather than waiting for queued samples to finish playing.

When the stream is started, audio I/O begins and the user callback function is repeatedly called with a pointer to a full input buffer -that is to be read-, and a pointer to an empty output buffer -that is to be filled-. It is also passed a time-stamp that is the number of sample frames generated so far. The time-stamp of the currently playing sample can also be queried which allows the audio output to be synchronized with Musical Instrument Digital Interface (MIDI) or video events. The callback function may return a non-zero value to indicate that the use of the stream has been completed. The main program can determine whether a stream has been completed using the Pa_StreamActive() function.

The Pa_CloseStream() function should be used to close a stream when it is no longer needed.

Other functions that can be handled are:

- Pa_StreamTime() returns the current playback time of a stream. It is intended for use as a time reference when synchronizing audio to MIDI.
- Pa_GetCPULoad() returns a floating point value ranging from zero to one which indicates the fraction of total CPU time being consumed by the stream's audio processing. This gives the programmer an opportunity to modify their synthesis techniques, or to reduce the number of voices, in order to prevent excessive loading of the CPU.

4.2.3 Error Handling

Most PortAudio functions return an error code of type *PaError*. The function Pa_GetErrorText(err) may be used to retrieve textual error information. PortAudio also provides the Pa_GetHostError() function for retrieving host specific error codes.

4.3 C++ Interface

PortAudio library has been used in a Visual C++ environment. An easy useful interface has been created. This interface allow us to select the wave file and then, clicking start, the signal is radiated in the room, received by the microphone, processed and the signal dereverberated is radiated again (not in the room) in the headphones to hear the real time dereverberation as can be seen in the Fig. 4.2. Also, the signal from the microphone and the signal dereverberated are recorded into wave files for later measures.

The next functions, written in C++, are the most important implemented in the "DSPLabDlg.cpp" file. These functions define the behavior of the dialog box and all the connections between the files, the callback function and the stream. The complete code of "DSPLabDlg.cpp" can be found in the appendix C.

BOOL CDSPLabDlg::OnInitDialog()

This function is called when the dialog box is started. First of all the Mode of the structure paData is initialized to zero. This Mode, which contains the code followed

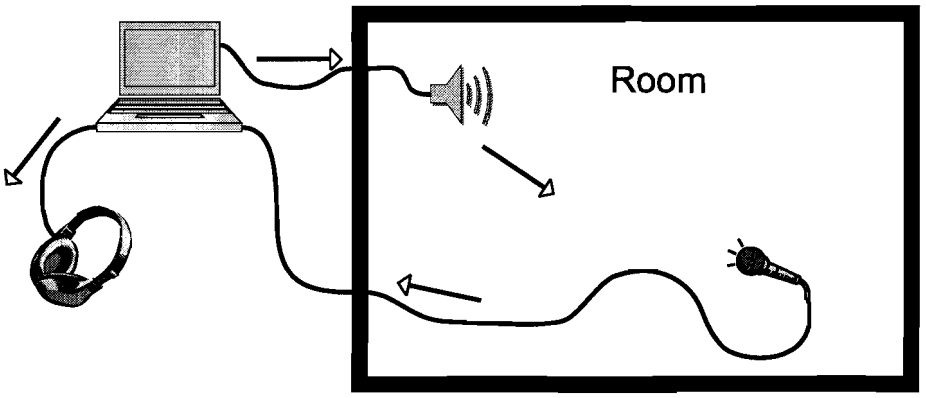

Figure 4.2 Real-Time Implementation

by the function OnProcessData, decides which case of the switch in the function OnProcessData is executed.

Then, the stream is initialized and also the program verifies if the process has been made without errors. If all is correct, then a message appears in the dialog box, explaining that the PortAudio I/O stream has being initialized.

The next step is to open the PortAudio stream, -the code to do this is showed in the last section-. After this, the number of samples per buffer of the structure paData is calculated and the buffers are allocated and initialized to zero. The samples per buffer are the result of the number of channels multiplied by the number of frames per buffer which is a constant defined in "DSPLabDlg.cpp" file.

The function ends with the start of the stream and also with the verification that none error has occurred. If everything is right the function returns *TRUE*.

void CDSPLabDlg::OnClose()

This function contains what has the program to make when the close button of the dialog box is clicked. Firstly, the PortAudio stream is stopped and before closing it the program verifies if the stream is really inactive. In both functions, stop and close, it is made a verification of errors.

Then, the PortAudio is terminated and the data buffers are released. The file used to read the signal radiated in the room is closed.

LRESULT CDSPLabDlg::OnProcessData (WPARAM wParam, LPARAM lParam)

This function describes the basic process of the data. First, the data is read from the input file (lines 3-6). Secondly, the data from the microphone is adjusted between the float range, in order to save this signal in a wave file (this is the reverberated signal)(lines 8-10). Then, after saving the signal (lines 12-14), it is called the process function which tries to dereverberate the microphone signal (line 17). The line 16 calls a function which returns a Hamming window used in the process of the signal. The result of this process is also saved into a file (lines 18-20). The last thing is to readjust the amplitude of this processed signal because it is radiated again but in the headphones so as to listen the effect of the program (lines 21-24). As can be seen when the complete code is observed (Appendix C), here it is explained only one of the values which can get the Mode of the structure paData, this is the case executed when the button start is clicked.

```
1     if ((InputFile.AFp != NULL))
2     {
3         AFreadData (InputFile.AFp, InputFile.Counter, pData->
              pOutSamplesChan, pData->SamplesPerBufferOut/
              NUMBER_OF_OUTPUT_CHANNELS);
4         InputFile.Counter += pData->SamplesPerBufferOut/
              NUMBER_OF_OUTPUT_CHANNELS;
5         if (InputFile.Counter >= InputFile.Nsamp)
6             InputFile.Counter = 0;
7     }
8     for (i=0 ; i<pData->SamplesPerBufferIn; i++)
9     {
10        pData->pInSamplesChan[i] = pData->pInSamplesChan[i] *
              32768;
11    }
12    if ((OutputFile.AFp != NULL))
13    {
14        AFwriteData (OutputFile.AFp, pData->pInSamplesChan, pData->
              SamplesPerBufferIn);
15    }
16    FIwinHamm (Win, LW, awindow);
17    Process (pData->pInSamplesChan, pData->SamplesPerBufferIn, Win,
          m_RT, m_T);
18    if ((OutputFile2.AFp != NULL))
19    {
20        AFwriteData (OutputFile2.AFp, pData->pInSamplesChan, pData
```

```
            ->SamplesPerBufferIn);
21      }
22      for (i=0 ; i<pData->SamplesPerBufferIn; i++)
23      {
24          pData->pInSamplesChan[i] = pData->pInSamplesChan[i] /
                32768;
25      }
26      for (i=0 ; i<pData->SamplesPerBufferOut/
            NUMBER_OF_OUTPUT_CHANNELS; i++)
27          pData->pOutSamplesChan[i] = (InputFile.Volume * pData->
                pOutSamplesChan[i]) / (32768 * 100);
28      break;
```

void CDSPLabDlg::OnBnClickedStart()

When the button start is clicked in the dialog box, the next steps follows. Firstly, the mode data is put to 9. Then, the program verify if there is a file to read the data. And finally the buttons start and stop are actualized: the button stop is enabled and the start is disabled. Also a message appears in the dialog box explaining that the process of the data has started.

void CDSPLabDlg::OnBnClickedStop()

This function is executed when the button stop is clicked in the dialog box. The important thing here is to close the output files, which are the files used to save the signals. Also the buttons are actualized back like in the previous function. And a message is presented explaining that the process data has stopped

void CDSPLabDlg::OnBnClickedLoad()

When the button load is clicked, before starting the program, executes the next code. The first step is to clear the buffers pInSamplesChan and pOutSamplesChan. Then the files are opened: the input file is opened only in the read mode and the other two (the file to save the signal received in the microphone and the file to save the signal dereverberated) are opened in write mode. Also the message "Loading wav-file..." is presented.

4.4 Process Function

The process function is the real time implementation of the algorithm described in the last chapter. With the aim of dereverberate the signal some mathematical functions, as FT, are needed. This functions can be found in the package called libtsp as it has been explained in the introduction.

The process function is called in the function explained in the last section (OnProcessData(...)):

```
1       Process (pData->pInSamplesChan, pData->SamplesPerBufferIn, Win,
            m_RT, m_T);
```

All the code of this process can be found in the appendix C.

Some considerations must be done before explaining the code. This considerations refers to the length of the received and processed data. The length of the data received data is SamplesPerBufferIn, but the length of the processed data is $FRAMES\_PER\_BUFFER + LW - SHIFT$. SamplesPerBufferIn has the same length as $FRAMES\_PER\_BUFFER$. LW is the length of the Hamming window used to do the Short-Time Fourier Transform. And $SHIFT$ is the amount of overlap in samples. Therefore, the length of processed data is larger than the received data.

In this way, the last samples from the last call are saved in the first part of the vector called NewSamples, followed by the new samples just arrived. That is, because for a given number of samples, it is not possible to process all of them, and the last samples need the data that will arrive in the next block to get an output (dereverberated signal).

When the data process starts, some initializations should be done. These are different depending on the dereverberation process state, wether the process is already started or it is the first time it is executed. When it is the first time, as no information about the last block is available, the vector must be filled up with zeros. This leads to a delay in the signal between the received signal in the microphone and the signal dereverberated.

After these initializations, five functions are called in order to dereverberate the signal:

- STFFT (Nsamples, Win);
- PhaseModule ();
- Estimation (DELAY);
- Signal();
- ISTFFT (Nsamples, Win);

At the end of the process, the last data of the vector received from the "DSPLabDlg.cpp" file (**Samples**) are saved in order to be used in the next process. After, the dereverberated data obtained in the vector NewSamples is copied in the vector Samples (used in the "DSPLabDlg.cpp" file), and as the vectors have different size, really only the first $FRAMES\_PER\_BUFFER$ samples of the dereverberated signal are really copied.

STFFT (int Nsamples, float Win[])

The STFFT is a function which receives the number of samples and the vector with the Hamming window values. The purpose of this function is to make the Short-Time Fourier Transform. The Hamming window is multiplied with the first LW (which is the length of the Hamming window) samples of the vector NewSamples. The result is stored in a vector **u** which has the length double that the used window, this is to improve the information in frequency. The next step is to call the function SPfRFFT from the library libtsp which does the Fast Fourier Transform (FFT) of the vector **u**. The result is stored in a column of a matrix.

This process is repeated for all the data of the vector NewSamples with the Hamming window sliding. The window only moves SHIFT samples with the consequent overlapping. SHIFT is calculated with: $LW - \frac{LW*OVERLAP}{100}$ where LW is the length of the window as has been explained above and OVERLAP is the amount of overlap wished in percentile.

At the end, a matrix with the FT in each column is obtained.

PhaseModule()

The purpose of this function is to get the phase and the module from the matrix obtained in the STFFT. The module is calculated as the square root of the real part to the square plus the imaginary part to the square. The phase is obtained as the arctangent of the imaginary part divided by the real part. Both, phase and module, are saved in matrix, **P** and **M** respectively.

Estimation (const int DELAY)

The estimation function takes the module matrix, **M**, and uses it to estimate the short-term PSD explained in chapter 3 with the expression:

$$\hat{\gamma}_{xx}(m,k) = \beta\hat{\gamma}_{xx}(m-1,k) + (1-\beta)|X(m,k)|^2$$

In the function, an estimation is made for all blocks. The problem is that, as can be seen in the equation

$$\hat{\gamma}_{x_r x_r}(m,k) = e^{-2\alpha T}\hat{\gamma}_{xx}(m-T',k)$$

there is a shift in the estimation of the reverberation. Thus some columns of the estimation of the last block are needed and some frames of the new estimation are also needed. The rest of the frames of the actual estimation which are not used in this block are saved for the next block. The number of frames, so the shift, are determined by T' which is defined as: $T' = \lfloor \frac{Tf_s}{32} \rfloor$, as has been explained in the last section of chapter 3.

The result of this function is a matrix with the estimation for the current block **E**, and another matrix with the estimation for the next block **Resere**.

Signal()

With the estimation, we apply the theory described in the section 3 of chapter 3 to dereverberate and reconstruct the signal before the inverse Fourier Transform. First the gain matrix **G** is calculated as in the equation (3.12), that we can express as

$\mathbf{G} = 1 - \sqrt{\frac{\mathbf{E}}{\mathbf{M}}}$. Secondly, infinitive values in the gain matrix are reset to one.

Then the module matrix $\mathbf{S}$ of the estimation signal is obtained multiplying the gain matrix with the module matrix $\mathbf{S} = \mathbf{G} * \mathbf{M}$. After this, if the estimated signal is minor than λ multiplied by the module matrix, the value of the estimated signal is substituted by $\lambda * \mathbf{M}$ and the gain function is modified by λ. The operations become for each position of the matrices.

The last thing before the inverse FT is to join phase and module.

ISTFFT (int Nsamples, float Win[])

This function take each column of the matrix obtained in the last function and does the inverse Fourier Transform. The result of each column is used to reconstruct the signal with the Hamming window and the corresponding overlapping.

The next figures show the relations between the most important files and functions explained in all this chapter and also the processes.

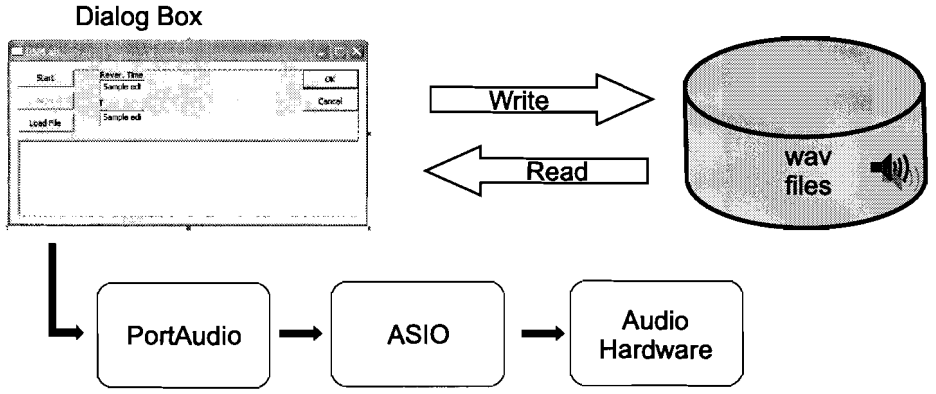

Figure 4.3 Processes relations

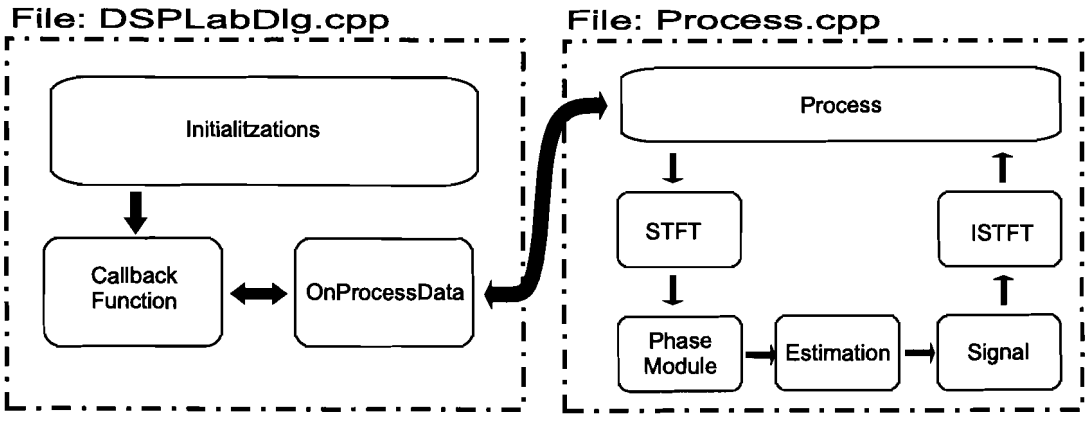

Figure 4.4 Relations between files and functions

Chapter 5

Experiments and discussions

5.1 Introduction

Approach to dereverberation of speech through the estimation of the Power Spectral Density of the reverberant signal and its removal through spectral subtraction were presented in the previous chapters. The experiments described in this chapter test the system depicted in Figure 3.1.

The evaluation of a speech enhancement algorithm is not simple. While objective quality assessment methods can indicate an improvement or degradation in speech quality based on mathematical measures, the human listener needs more than a mathematical error criterion. Therefore, it is also required to study subjective measurements of intelligibility and quality.

It is important to verify the functionality of the algorithm. The second section describes the Matlab implementation of the algorithm presented in chapter 3. Section 3 explains the objective measures that were used to evaluate the algorithm. In section 4, the results obtained in Matlab are presented. Real-time implementation results are presented in section 5. Section 6 discusses the results obtained in the previous two sections. The section 7, proposes and evaluates an improvement for the algorithm.

5.2 Matlab implementation

The image method has been used in Matlab to get a Room Impulse Response. This impulse response is needed to create a reverberant signal, and to determine de Reverberation Time.

The RT is obtained by generating synthetics RIRs many times changing the source position with a fixed receiver position. From all these RIRs are obtained RTs which values are quite similar because the statistical properties of late reflections do not change significantly as a function of position. All these RTs are averaged and the value obtained is used as the unique RT of the room.

The reverberant signal, which is the signal that would be received in a microphone in a real enclosure, is obtained by the convolution of the clean signal with the RIR.

With both, RT and reverberant signal, we apply the algorithm presented in the chapter 3, which uses all the reverberant signal and dereberverates it without dividing it in blocks. The algorithm catches all the signal, applies the Short-Time Fourier Transform, so it is obtained a matrix with the frames (in columns) and then the output is divided in phase and module. The module is used to estimate the reverberant Power Spectral Density. Then the spectral subtraction and the output are joined with the phase to do the inverse Fourier Transform. The signal is reconstructed windowing each frame by the Hamming window and overlapping the result with the same overlap used to do the STFT. The most important functions implemented in Matlab are showed in appendix D.

Figure 5.1 shows an example of the reverberated, the processed and the anechoic signals. The signal used was an anechoic female voice of 7 seconds sampled at 8 kHz. The distance between the source and the microphone was two meter in a room with sizes 6 m × 6 m × 3 m (l x w x h). The RT used for the room was $T_{60} = 0.448$ sec and $T = 40$ ms. In Figure 5.2, it is depicted the spectrograms of the anechoic, reverberated and processed signals. It can be seen clearly that overlap-masking are reduced.

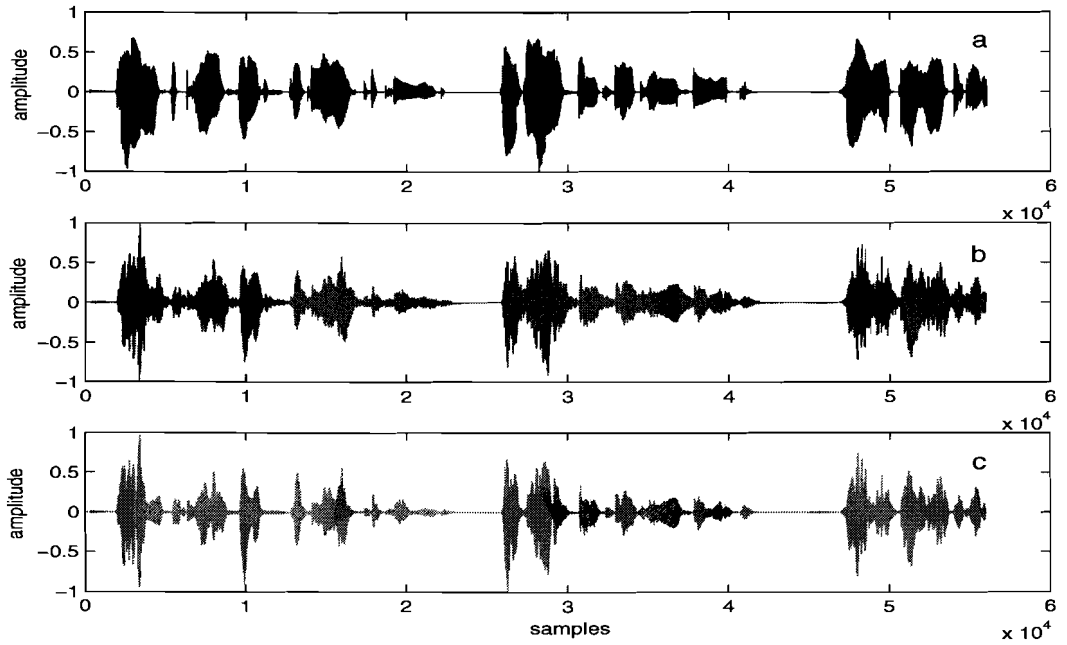

Figure 5.1 Anechoic (a), reverberated (b) and processed (c) signals

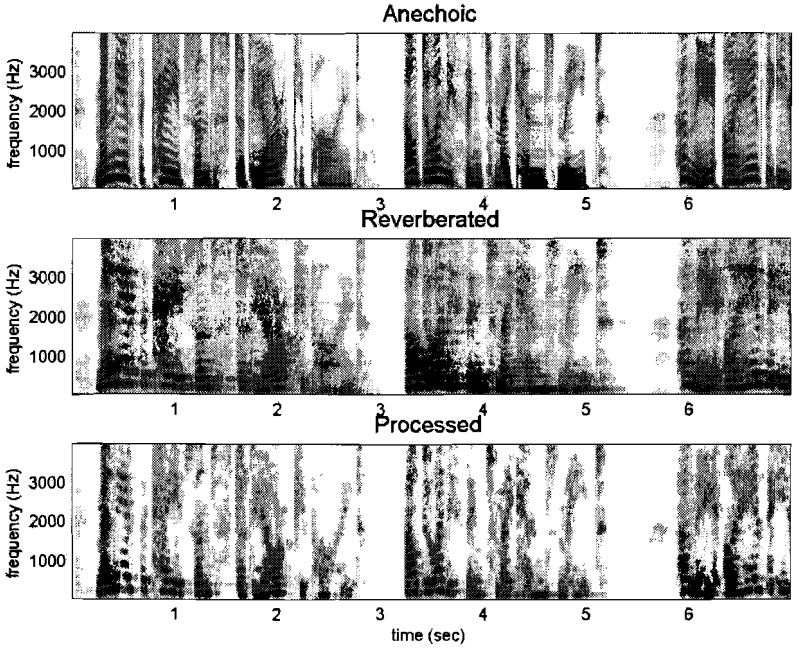

Figure 5.2 Spectrograms

5.3 Objective measures for performance evaluation

In the evaluation of speech enhancement algorithms, it is necessary to identify the similarities and differences in perceived quality and subjectively measured intelligibility. Speech quality is an indicator of the naturalness of the processed speech signal. Intelligibility of speech signals is a measure of the amount of speech information present in the signal that is responsible for conveying what the speaker is saying. The interrelationship between perceived speech and intelligibility is not clearly understood. While unintelligible speech may not be considered to be of high quality, the converse may not be true. For human listeners, it is important for the speech signal to be intelligible, even at the expense of some degradation in speech quality. For example, human end-users could actually prefer a less aggressive enhancement method that may not completely remove all of the interfering noise, to a more aggressive algorithm that may completely remove the noise component but also reduce the speech intelligibility.

Performance evaluation tests can be done by subjective quality measures or objective quality measures. Subjective measures are discussed in section 4. While subjective measures provide a broad measure of performance since a large difference different in quality is necessary to be distinguishable to the listener. Hence, it becomes difficult to get a reliable measure of changes due to algorithm parameters. On the other hand, objective measures provide a measure that can easily be implemented and reliably reproduced.

Objective measures are based on a mathematical comparison of the original and processed speech signals. The majority of objective quality measures quantify speech quality in terms of a numerical distance measure.

Objective measurements were used for the reverberation reduction, speech distortion, and reverberation distance. The reverberant signal of the microphone was decomposed into the sum of a direct signal $d_{in}(n)$ and a reverberant part $r_{in}(n)$, obtained by convolving the anechoic signal with the first 6 ms (measured w.r.t. the arrival time of the direct sound) of the Room Impulse Response, and with the RIR minus this part. The delay in the direct sound was calculated looking for all the positions where the RIR was ten times minor than the maximum amplitude

of the RIR. When all the positions were found, the first one was selected as the moment in which the direct signal arrived. While the complete reverberant signal was being processed, the time-varying signal dependent gain function $G(t, f)$ was recorded. The recorded gain was then applied separately to the reverberant part, giving $r_{out}(n)$.

5.3.1 Reverberation Reduction

When no speech was present in the anechoic signal the global reverberation reduction was calculated using

$$RR = 10log_{10}\left(\frac{\sum_{n\in\Omega_{silence}} r_{in}^2(n)}{\sum_{n\in\Omega_{silence}} r_{out}^2(n)}\right)$$

The separation between speech and silence zones was made through a function. This function uses the anechoic signal $x(n)$ to create a new signal with:

$$\hat{x}(n) = \beta\,\hat{x}(n-1) + (1-\beta)|x(n)|$$

with $\beta = 0.85$.

Whit this signal, the silence is defined as the points where $\hat{x}(n) < 0.001$. The function also verifies the duration of silence, if the silence doesn't have a minimum duration, it is not considered silence.

5.3.2 Speech Distortion

The Log Spectral Density (LSD) between the direct signal $d_{in}(n)$ and the output signal $\hat{s}(n)$ is used as a measure of distortion. The distance is calculated using:

$$LSD(m,k) = |10log_{10}\hat{S}'(m,k) - 10log_{10}D'_{in}(m,k)|\ (dB)$$

where

$$\hat{S}'(m,k) \triangleq max\{|\hat{S}(m,k)|^2, \delta\}$$

is the spectral power, clipped such that the log-spectrum dynamic range is confined to about 50 dB, i.e.

$$\delta = 10^{-50/10} max_{m,k}\{|\hat{S}(m,k)|^2\}$$

where $max_{m,k}$ is the maximum value over all the frequencies and frames.

The spectral power D'_{in} is defined similarly. The LSD is averaged over all frequencies and finally averaged over all frames containing speech.

This measure is compared with the LSD between the signal received at the microphone $x(n)$ and the direct signal. The equation is:

$$LSD(m,k) = |10log_{10}X'(m,k) - 10log_{10}D'_{in}(m,k)| \ (dB)$$

where $X'(m,k)$ is defined as has been explained before.

5.3.3 Reverberant Estimation Distance

The Log Spectral Density between the estimation of the reverberant signal $\hat{\gamma}_{x_r x_r}(m,k)$, explained in the second section of chapter 3, and the estimation of the PSD of the reverberant part $r_{in}(n)$ is used as a measure of reverberant estimation distance. The estimation of $\hat{\gamma}_{r_{in}r_{in}}$ is calculated as:

$$\hat{\gamma}_{r_{in}r_{in}}(m,k) = \beta\hat{\gamma}_{r_{in}r_{in}}(m-1,k) + (1-\beta)|R_{in}(m,k)|^2$$

The distance is defined as before:

$$LSD_\gamma(m,k) = |10log_{10}\hat{\gamma}'_{x_r x_r}(m,k) - 10log_{10}\hat{\gamma}'_{r_{in}r_{in}}(m,k)| \ (dB)$$

where $\hat{\gamma}'_{x_r x_r}(m,k)$ and $\hat{\gamma}'_{r_{in}r_{in}}(m,k)$ are calculated with

$$\begin{aligned} \hat{\gamma}'_x(m,k) &\triangleq max\{|\hat{\gamma}'_x(m,k)|^2, \delta\} \\ \delta &= 10^{-50/10} max_{m,k}\{|\hat{\gamma}'_x(m,k)|^2\} \end{aligned}$$

in the same way explained in the speech distortion.

5.4 Matlab Results

In order to repeat the measurements, a function was created. This function changes the source position in the room and leaves the receiver in a fixed position. The room dimensions used in all the Matlab experiments are 6 $m \times 6\ m \times 3\ m$ (l x w x h) and the Reverberation Time is $T_{60} = 0.448\ sec$, obtained as has been explained in the section 2 of this chapter.

The results are obtained using a female voice of 7 seconds with a sample rate of 8 kHz. The graphs are depicted in function of distance and with $T = 40\ ms$. As has been explained, the function changes the source position by all the room in steps of 5 cm. Once, all the measures are obtained, the results obtained for the same distances are averaged and for distances greater than reverberation distance (see section 3 of chapter 2), the results are averaged, each 25 centimeters one average result is given. The reverberation distance calculated for this room is:

$$r_d = 0.1\sqrt{\frac{V}{\pi T_{60}}} = 0.876\ meters \tag{5.1}$$

Speech Distortion

Figure 5.3 shows the speech distortion of the signal received in the microphone and the signal obtained after the dereverberated process. The comparison is made with the direct signal which has been explained in section 2 of this chapter.

As can be seen, the plot shows an improvement in the amount of the speech distortion. Only in the first half meter the result is up side down, the dereverberated signal has more distortion than the microphone received at the microphone.

Reverberant Estimation Distance

The reverberant estimation distance is showed in the Figure 5.4. This distance is calculated with the estimation of the Power Spectral Density of the reverberant part (used in the reverberation reduction) and the estimation of PSD of the reverberant signal model (section 2 chapter 3).

The plot show more or less a flat distance between the estimation of the PSD

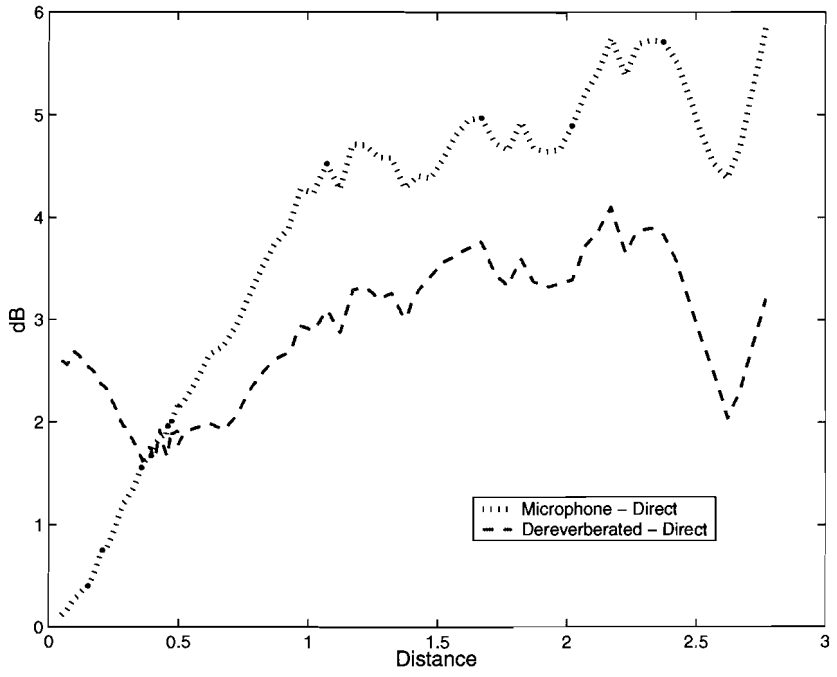

Figure 5.3 Speech Distortion as a function of distance

reverberant signal $\hat{\gamma}_{x_r x_r}(m,k)$ and the estimation of the $\hat{\gamma}_{r_{in} r_{in}}$ for distances greater than the reverberation distance (0.876 m). In distances smaller, the distance become larger until more than 50 dB

Reverberation Reduction

Figure 5.5 shows the reverberant reduction of the signal obtained at the output of the dereverberated process. The input signal to the process is only the reverberant part, obtained by convolving the anechoic signal with the Room Impulse Response minus the first 6 ms (measured w.r.t. the arrival time of the direct sound) of the RIR.

5.5 Real Time Implementation Results

Subjective measures are used in order to show the good operation of the real time implementation. The tests are made for different distance, for each distance, each

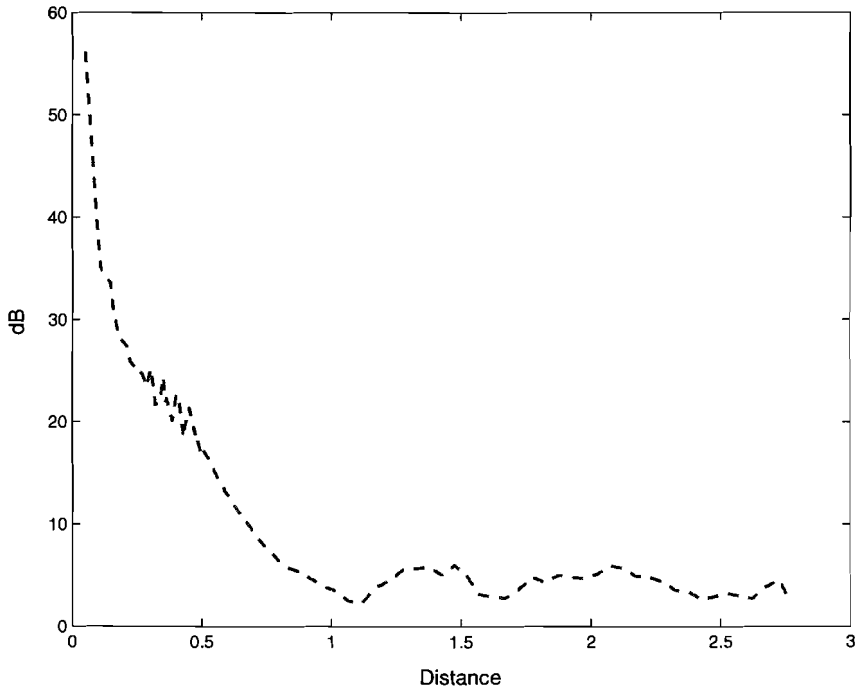

Figure 5.4 Reverberant Estimation Distance as a function of distance

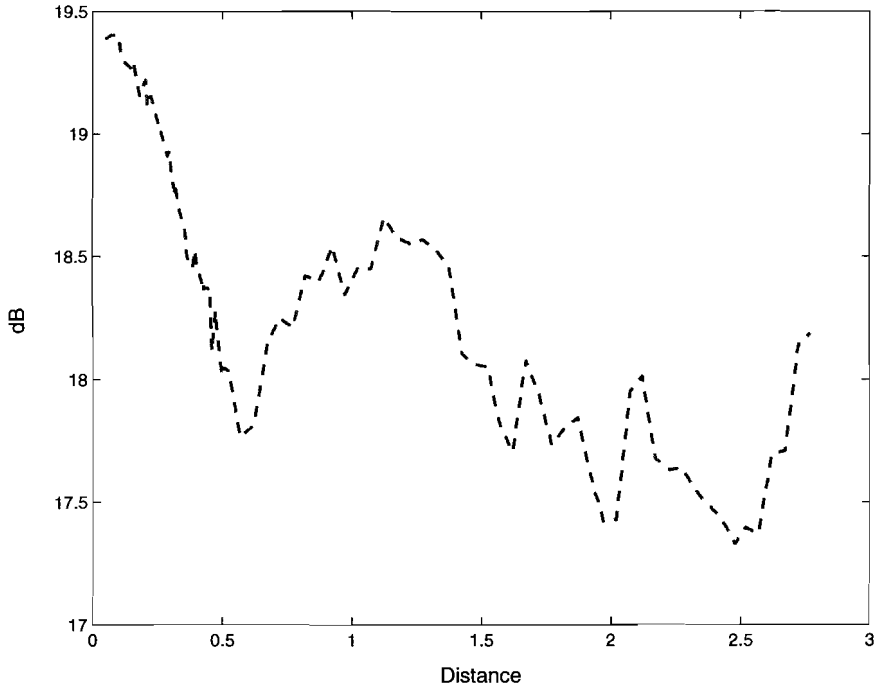

Figure 5.5 Reverberation Reduction as a function of distance

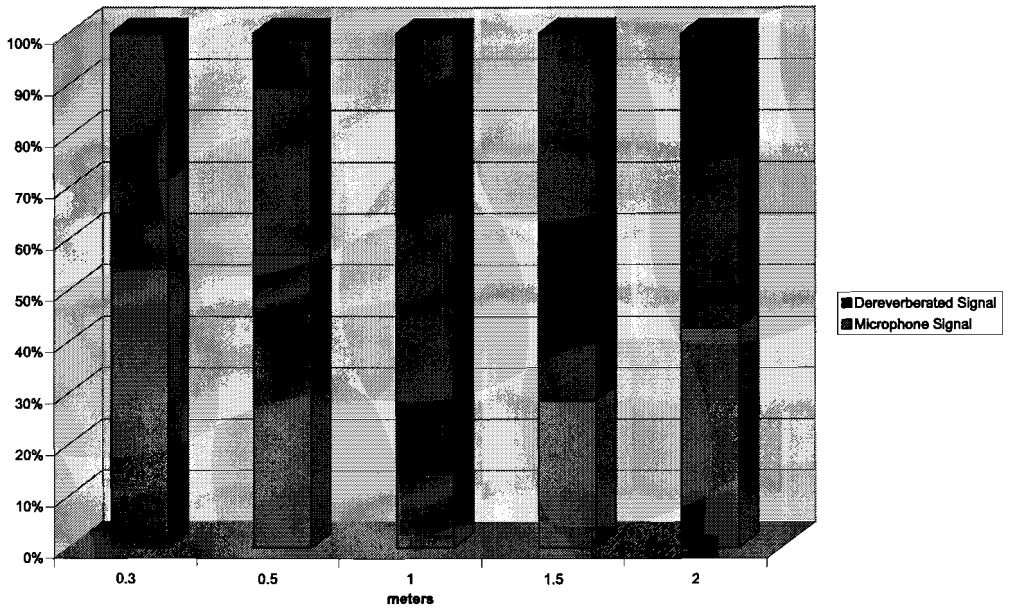

Figure 5.6 Real time implementation results

listener chose what speech signal sounds better, the dereverberated signal or the signal received at the microphone. The speech signal is the same used before (female voice of 7 seconds with a sample rate of 8 kHz).

Figure 5.6 shows the result of 7 listeners in %. As can be seen, at least the 50% of the listeners prefer the dereverberated signal when the distance is longer than one meter. When the distance is smaller than one meter then the listeners prefer the signal received at the microphone.

5.6 Discussion

As can be seen in Matlab results and real time results, there is something wrong for small distances. Matlab results show a big amount of distortion and a large reverberant estimation distance for distances smaller than the reverberation distance.

When the distance between the source and the microphone are smaller than the reverberation distance the Energy Decay Curve change the normal shape and a step appears just in the beginning. In Figures 5.7 and 5.8 two different EDC

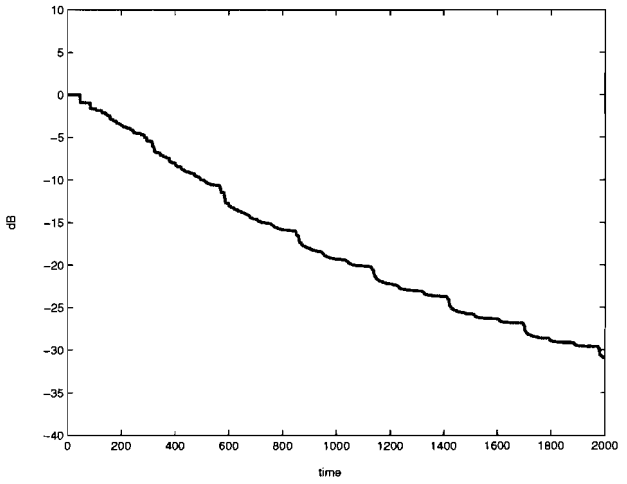

Figure 5.7 Energy Decay Curve

can be compared, the first one is one of the EDC used to get a RT close to the Reverberation Time obtained at the end ($T_{60} = 0.448\ sec$), the second one is a EDC obtained with a distance between the source and the microphone equal to 0.5 m.

The step can be observed at the beginning of the EDC of the Figure 5.8. This step is translated in a big amount of distortion in the dereverberated signal and then it is reduced the intelligibility of the signal.

5.7 Improvement of the algorithm

It has been proposed the next modification for the Matlab implementation in order to improve the speech distortion when the source and the microphone are closer one to each other. The results of this improvements are showed below.

The change proposed is to use the value of the step to change the equation

$$\hat{\gamma}_{x_r x_r}(m,k) = e^{-2\alpha T}\hat{\gamma}_{xx}(m-T',k)$$

explained in chapter 3, section 5 into:

$$\hat{\gamma}_{x_r x_r}(m,k) = \frac{e^{-2\alpha T}}{\xi}\hat{\gamma}_{xx}(m-T',k)$$

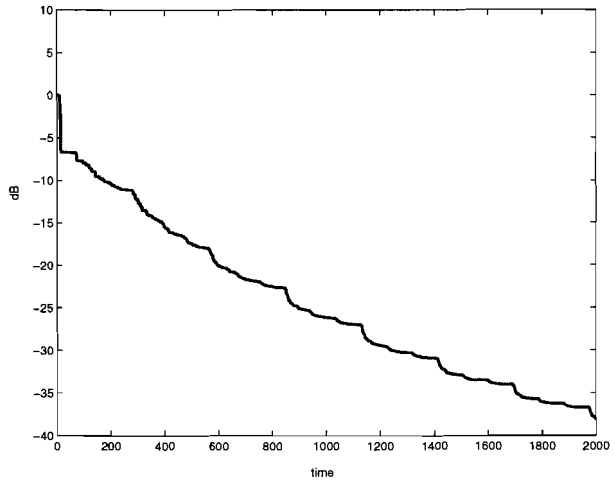

Figure 5.8 Energy Decay Curve with a distance equal to 0.5 m

where ξ is calculated as:

$$\xi = 10^{\frac{step\_value\_in\_dB}{10}}$$

The *step_value_in_dB* is calculated in Matlab from the Energy Decay Curve of each realization. For each position, the synthetic RIR is generated to create the reverberant signal, as has been done before, but now also the EDC is calculated. This is used to obtain the step which will be introduced in the estimation of the PSD. The results of this improvement are shown below.

Speech Distortion

The comparison between the algorithm and the improved algorithm in terms of speech distortion is showed in Figure 5.9. Also, the speech distortion between the signal received in the microphone and direct signal is depicted.

As can be seen, the speech distortion for distances greater than half meter are approximately the same without the improvement, but for smaller distances the algorithm reduce the speech distortion and follows the speech distortion of the signal received at the microphone which decrease quickly.

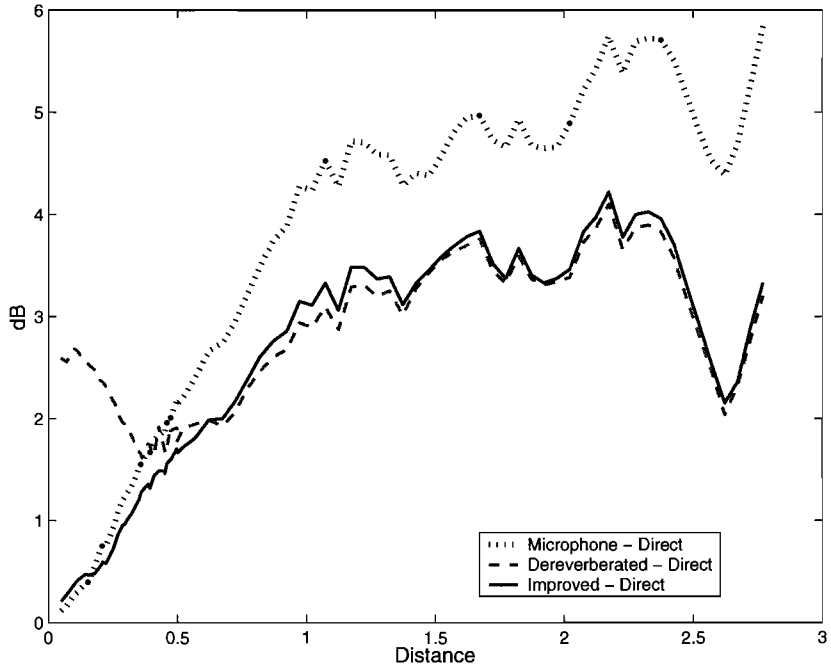

Figure 5.9 Speech Distortion improved algorithm

Reverberant Estimation Distance

The reverberant estimation distance is showed in the Figure 5.10. The figure also shows the graph obtained without the improvement.

The results for the reverberant estimation distance show and improvement for the distance smaller than the reverberation distance.

Reverberation Reduction

Figure 5.11 shows the reverberant reduction of the signal obtained at the output of the dereverberated process. The figure also shows the graph obtained without the improvement.

Now the reverberation reduction is smaller than before. The results shown a big variance in the smaller distance, this is because for the small distance no average has been done.

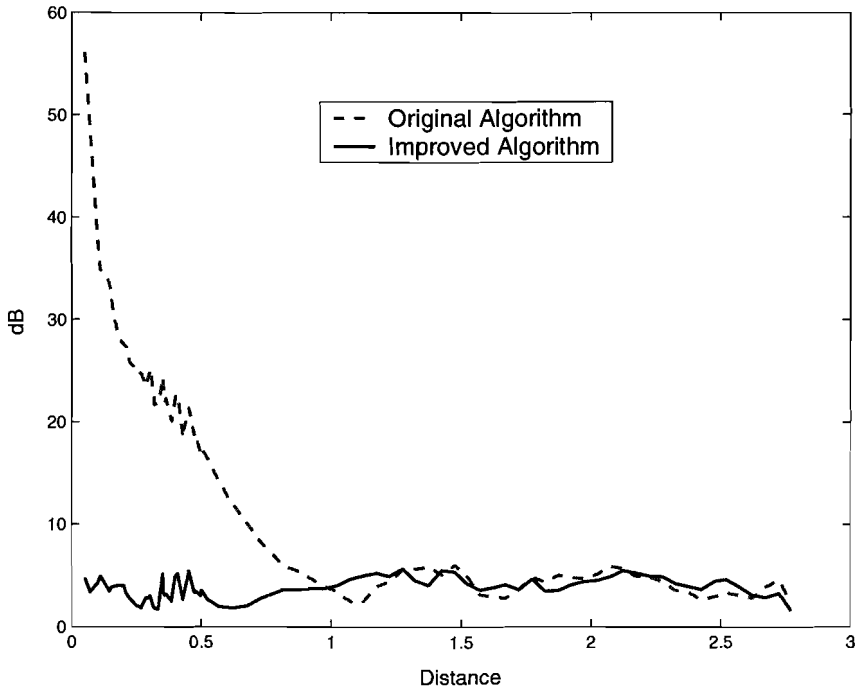

Figure 5.10 Reverberant Estimation Distance improved algorithm

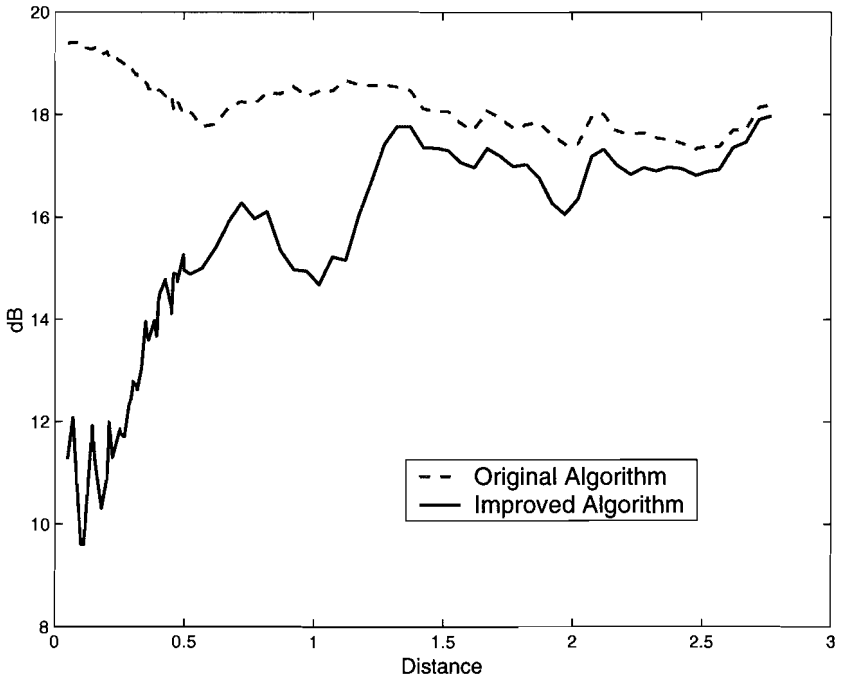

Figure 5.11 Reverberation Reduction improved algorithm

5.7.1 Comparison with different Ts

As can be seen in all the results, for the short distance, the improvement proposed works quite well.

The Table 5.1 shows the results for the improvement algorithm, in function of distance and for two different T, $T = 40\ ms$ and $T = 60\ ms$.

As can be seen, the values are quite similar. In the reverberation reduction, the results are better in $T = 40\ ms$, but then the results of the speech distortion are not so good as the results for $T = 60\ ms$. The most important thing is that, as it is possible to view, the algorithm works properly for both T, $T = 40\ ms$ and $T = 60\ ms$.

The Figures 5.12, 5.13 and 5.14 depicted the values showed in the table.

DISTANCE	Reverberation Reduction (dB)		Speech Distortion (dB)		Reverberant Estimation Distance (dB)	
	$T = 40\ ms$	$T = 60\ ms$	$T = 40\ ms$	$T = 60\ ms$	$T = 40\ ms$	$T = 60\ ms$
0.1	9.4649	8.8113	0.39225	0.30742	5.2959	10.333
0.15	11.47	10.587	0.46025	0.33357	3.8297	8.8304
0.20	11.213	10.276	0.54538	0.62755	3.5902	8.583
0.25	12.45	11.456	0.72643	0.82823	1.9493	6.9231
0.30	13.692	12.522	1.0071	1.1314	2.3882	3.7978
0.35	15.083	13.9	1.242	1.3898	6.046	0.95311
0.40	15.05	13.96	1.412	1.5766	6.4029	1.3147
0.45	15.349	14.354	1.5085	1.7013	6.5092	1.4213
0.50	15.504	14.258	1.597	1.8307	5.4108	0.96347
0.55	15.285	14.245	1.4938	1.7523	3.0635	3.1492
0.60	16.199	14.942	1.4425	1.783	0.87971	5.2329
0.65	16.115	14.865	1.4689	1.7655	1.2463	6.2062
0.70	16.771	15.418	1.5529	1.7451	2.6357	7.6108
0.75	16.391	14.632	1.5826	1.7672	2.5992	7.5552
0.80	15.462	13.442	1.7106	1.8797	3.2875	8.2517

Table 5.1 Comparison between $T = 40\ ms$ and $T = 60\ ms$ for short distances

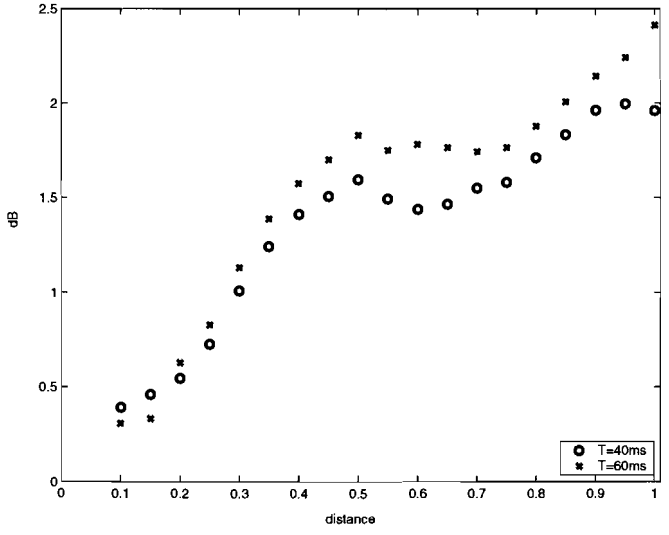

Figure 5.12 Speech Dereverberation comparison

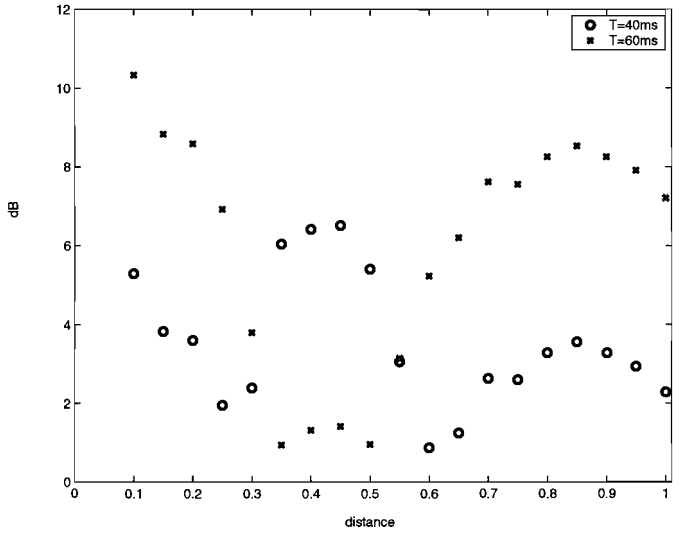

Figure 5.13 Reverberant Estimation Distance comparison

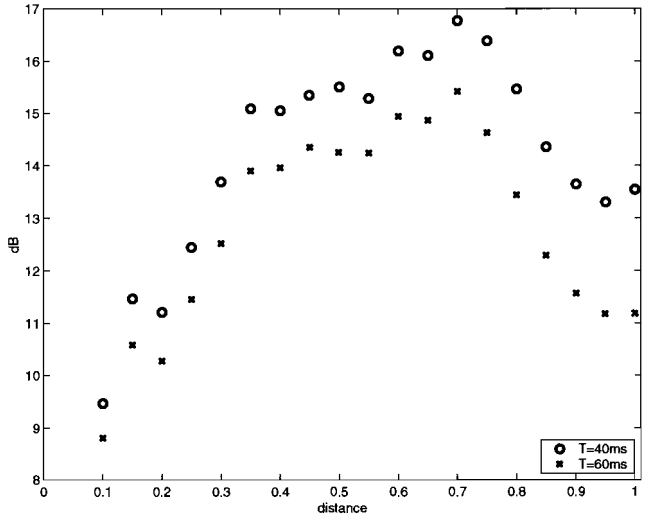

Figure 5.14 Reverberation Reduction comparison

Chapter 6

Conclusions

It has been shown by many researchers that reverberation degrades the intelligibility of speech, in particular under monaural conditions. It has also been noted that reverberation affects the naturalness of speech, lending it a "hollow" sound. The goal of this research was to develop a real time implementation of an algorithm which dereverberates the speech recorded with one microphone in a reverberant environment.

The algorithm which has been used and implemented in real time, suppresses late room reflections (reverberation) from speech signals, based on spectral subtraction. The algorithm is based on a statistical model of the Room Impulse Response (RIR). This model is used to obtain an estimation of the Power Spectral Density (PSD) of the reverberation. This estimate the spectral subtraction and obtain a dereverberated speech signal.

In Matlab simulations the synthetic Room Impulse Response was generated with the image method in order to construct the reverberant signal. The difference between the two methods (real time and Matlab) was the use of signal segmentation for the real time implementation. In Matlab, the author chose to process all the signal at the same time. For all the signal, the Short-Time Fourier Transform was made, all the estimation was calculated and all the dereverberated signal was given without mattering in the length of the speech signal. The segmentation was necessary to give and output without a big amount of delay.

The real time implementation was developed in C++. The PortAudio library which provides streaming audio input and output was used. PortAudio is a cross-platform Application Programming Interface (API) that works on Windows and other platforms. Also a Visual C++ dialog interface has been used. The dialog interface can be used to select a wave file which will be radiated in the room. It can also be used to specify the Reverberation Time (RT) (T_{60}) and other parameters.

A prior knowledge of the Reverberation Time is needed in the real time implementation. This Reverberation Time was obtained from the Energy Decay Curve, using the well-known Schroeder method, of the Room Impulse Response measured before running the program. The program uses this Reverberation Time to dereverberate the speech signal in the room. It does not matter where the source and the microphone are situated because as was explained, a point to point impulse response does characterize the late reflections of the room, although the early echo pattern is dependent on the positions and directivities of the source and receiver.

During the measures done in real time, a particular situation was founded when the distance between the source and the microphone was smaller than the reverberation distance, explained in section 3 of chapter 2. When this occurs, the objective measures show a large distortion and also an increase of the reverberation reduction and reverberant estimation distance. Then, an improvement of the algorithm was proposed. This improvement is based on the measure of the step which appears in the Energy Decay Curve when the distance between the microphone and the source is smaller than the reverberation distance.

The objective and subjective measures showed proper operation of the algorithm implemented in real time for all source-receiver distances larger than the reverberation distance. Once the Reverberation Time is known, which is a characteristic of the room not of the position, the results means that for applications such as distant sound pickup automatic speech recognition, the user can move around the room while dictating to the speech recognizer without considering the position with the exception that the distance between the speaker and the microphone is smaller than the reverberation distance.

The objective measures (as defined in chapter 5) showed that the proposed improvement works independently from the parameter T. This parameter can be used

to control the time instant (measured with respect to the arrival time of the direct sound) from which the received reflections will be considered as late reflections, i.e. reverberation.

Direction for future work could be the extension of this algorithm for all frequencies. As has been explained in chapter two, the statistical model for the Room Impulse Response is only valid in the region from the lowest resonant mode to the Schroeder cut-off frequency. It could be interesting to integrate into the model differences in Reverberation Times along the frequency.

Another direction for future work could be the improvement of the speech distortion. In chapter 5, an improvement for the speech distortion, when the distance between the microphone and the signal is smaller than the reverberation distance, has been presented, but for distance larger it could be possible to reduce the speech distortion in order to hear a dereverberated signal more similar to the anechoic signal.

Appendix A

List of abbreviations, notations and used excerpts

A.1 List of abbreviations

API Application Programming Interface

ASIO Audio Stream Input/Output

BIBO Bounded-Input, Bounded-Output

CPU Central Processing Unit

EDC Energy Decay Curve

EDT Early Decay Time

FFT Fast Fourier Transform

FT Fourier Transform

LSD Log Spectral Density

LTI Linear Time-Invariant

MIDI Musical Instrument Digital Interface

MS Magnitude Subtraction

PSD Power Spectral Density

RIR Room Impulse Response

RT Reverberation Time

RTF Room Transfer Function

SNR Signal to Noise Ratio

STFT Short-Time Fourier Transform

TF Time Frequency

WMME Windows MultiMedia Extensions

A.2 List of symbols

$\star$	convolution
$\hat{\cdot}$	estimate
$\triangleq$	defined as
$\approx$	approx
$\lvert \cdot \rvert$	absolute value
$\lfloor x \rfloor$	biggest integer less than or equal to x
γ	log spectral density
c	speed of sound
δ_i	damping constant
f	temporal frequency
f_s	sampling frequency
f_g	Schroeder cut-off frequency
$h(t)$	impulse response function
$H(\omega)$	frequency response function
ln	natural logarithm function
p	sound pressure
r_d	reverberation distance
t	time variable
T_{10}	early decay time
T_{60}	reverberation time
T_i	upper limit of Schroeder integration
T_s	time span
w	angular frequency

Appendix B

Short-Time Fourier Transform

The Short-Time Fourier Transform (STFT) is a natural extension of the Fourier Transform. To be specific, let $g(t)$ denote a signal that is assumed to be "stationary" when viewed through a temporal window $w(t)$ that is of limited extent; in general, $w(t)$ is complex valued. The Short-Time Fourier Transform of the signal $g(t)$ is then dfefined as the Fourier Transform of the windowed signal $g(t)w^*(t-\tau)$, where τ is the center position of the window and the asterisk denotes complex conjugation. That is, we have

$$G_{STFT}(\tau, f) = \int_{-\infty}^{\infty} g(t)w^*(t-\tau)e^{-j2\pi ft}\,dt \tag{B.1}$$

which is linea in the signal $g(t)$. Accordingly, the Short-Time Fourier Transform maps the one-dimensional function $g(t)$ into the two dimensional function $G_{STFT}(\tau, f)$. The parameter f plays a role similar to that of frequency in the ordinary Fourier Transform. Moreover, for a given $g(t)$, the result obtained by computing $G_{STFT}(\tau, f)$ is dependent on the choice of the window $w(t)$. Many different window shapes are used in practice. Typically, they are symmetric, unimodal, and smooth.

Many of the properties of the Fourier Transform are carried over to the Short-Time Fourier Transform. In particular , we may mention the following two signal-preserving properties:

1. The Short-Time Fourier Transform preserves time shifts, except for a linear modulation. That is, if $G_{STFT}(\tau, f)$ is the Short-Time Fourier Transform of

the signal $g(t)$, then the Short-Time Fourier Transform of the time-shifted signal $g(t - t_0)$ is given by $e^{-j2\pi f t_0} G_{STFT}(\tau - t_0, f)$.

2. The Short-Time Fourier Transform preserves frequency shifts. That is, if $G_{STFT}(\tau, f)$ is the Short-Time Fourier Transform of the signal $g(t)$, then the Short-Time Fourier Transform of the modulated signal $g(t)e^{-j2\pi f_0 t}$ is given by $G_{STFT}(\tau, f - f_0)$

For another interpretation of the Short-Time Fourier Transform, we may rewrite equation (B.1) in the equivalent form

$$G_{STFT}(\tau, f) = e^{-j2\pi f\tau}[g(\tau) \star w^*(-\tau)e^{-j2\pi f\tau}] \tag{B.2}$$

where $\star$ denotes convolution, and $w^*(-\tau)e^{j2\pi f\tau}$ represents a linearly modulated impulse response. Recognizing that convolution in the time domain is transformed into multiplication in the frequency domain, we may go one step further and redefine the Short-Time Fourier Transform of the signal $g(t)$ as

$$G_{STFT}(\tau, f) = e^{-j2\pi f\tau} \int_{-\infty}^{\infty} G(\nu)W^*(\nu - f)e^{j2\pi\nu\tau}\, d\nu \tag{B.3}$$

where the frequency ν is a dummy variable, $G(\nu)$ is the Fourier Transform of $g(t)$, and the spectral window $W(\nu)$ is the Fourier Transform of the temporal window $w(t)$. Equation (B.3) shows that, except for the linear phase factor $e^{-j2\pi f\tau}$, the Short-Time Fourier Transform of the signal $g(t)$ can also be defined as the inverse Fourier Transform of the windowed spectrum $G(\nu)W^*(\nu - f)$.

The "time-domain" computational interpretation of the Short-Time Fourier Transform may be viewed as the dual of the "frequency-domain" computational interpretation. The nature of this duality is illustrated graphically in Figure B.1. According to the picture, the Short-Time Fourier Transform basis is often illustrated by a tiling of the time-frequency plane, where each tile represents a particular basis element.

The height and width of a tile represent the spectral and temporal widths of the basis element, respectively, and the position of a tile represents the spectral and temporal centers of the basis element. Note that, while the tiling diagram (Figure B.1) suggests that the STFT uses a discrete set of time/frequency shifts, the STFT

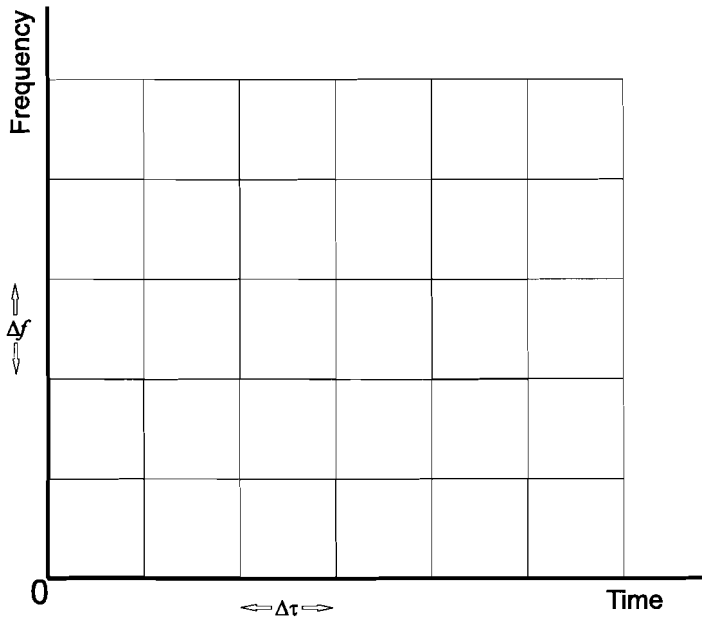

Figure B.1 Time-frequency plane.

basis is really constructed from a continuum of time/frequency shifts.

Note that we can decrease spectral width Δf at the cost of increased temporal width $\Delta\tau$ by stretching basis waveforms in time, although the time-bandwidth product $\Delta f\,\Delta\tau$ (i.e., the area of each tile) will remain constant (Figure B.2).

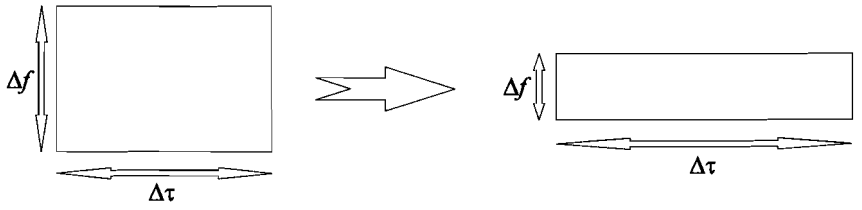

Figure B.2 Time-bandwidth product.

Can be observed:

- The time resolutions and frequency resolutions of every STFT basis element will equal those of the window $w(t)$. (Al STFT tiles have the same shape).
- The use of a wide window will give good frequency resolution but poor time resolution, while the use of a narrow window will give good time resolution

but poor frequency. (When tiles are stretched in one direction they shrink in the other).

- The combined time-frequency resolution of the basis, proportional to $\frac{1}{\Delta t \Delta f}$, is determined not by window width but by window shape. Of all shapes, the Gaussian $w(t) = \frac{1}{\sqrt{2\pi}} e^{-(\frac{1}{2}t^2)}$ gives the highest time-frequency resolution, although its infinite time-support makes it impossible to implement. (The Gaussian window results in tiles with minimum area).

Appendix C

C files

C.1 PortAudio header

```
#ifndef PORT_AUDIO_H
#define PORT_AUDIO_H

#ifdef __cplusplus
extern "C"
{
#endif /* __cplusplus */

/*
 * $Id: portaudio.h,v 1.5 2002/03/26 18:04:22 philburk Exp $
 * PortAudio Portable Real-Time Audio Library
 * PortAudio API Header File
 * Latest version available at: http://www.audiomulch.com/portaudio/
 *
 * Copyright (c) 1999-2000 Ross Bencina and Phil Burk
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files
 * (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * Any person wishing to distribute modifications to the Software is
 * requested to send the modifications to the original developer so that
 * they can be incorporated into the canonical version.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

typedef int PaError;
typedef enum {
    paNoError = 0,

    paHostError = -10000,
    paInvalidChannelCount,
    paInvalidSampleRate,
    paInvalidDeviceId,
    paInvalidFlag,
    paSampleFormatNotSupported,
    paBadIODeviceCombination,
    paInsufficientMemory,
    paBufferTooBig,
    paBufferTooSmall,
```

```
        paNullCallback,
        paBadStreamPtr,
        paTimedOut,
        paInternalError,
59      paDeviceUnavailable
    } PaErrorNum;

    /* Pa_Initialize() is the library initialisation function - call this before using the library.
    */
64
    PaError Pa_Initialize( void );

    /* Pa_Terminate() is the library termination function - call this after using the library.
    */
69
    PaError Pa_Terminate( void );

    /* Pa_GetHostError() returns a host specific error code.
    This can be called after receiving a PortAudio error code of paHostError.
74  */

    long Pa_GetHostError( void );

    /* Pa_GetErrorText() translates the supplied PortAudio error number into a human readable
        message.
79  */

    const char *Pa_GetErrorText( PaError errnum );

    /* Sample formats
84   These are formats used to pass sound data between the callback and the stream. Each device
        has a "native" format which may be used when optimum efficiency or control over
        conversion is required.
     Formats marked "always available" are supported (emulated) by all PortAudio implementations.
     The floating point representation (paFloat32) uses +1.0 and -1.0 as the maximum and minimum
        respectively.
     paUInt8 is an unsigned 8 bit format where 128 is considered "ground"
    */
89
    typedef unsigned long PaSampleFormat;
    #define paFloat32       ((PaSampleFormat) (1<<0)) /* always available */
    #define paInt16         ((PaSampleFormat) (1<<1)) /* always available */
    #define paInt32         ((PaSampleFormat) (1<<2)) /* always available */
94  #define paInt24         ((PaSampleFormat) (1<<3))
    #define paPackedInt24   ((PaSampleFormat) (1<<4))
    #define paInt8          ((PaSampleFormat) (1<<5))
    #define paUInt8         ((PaSampleFormat) (1<<6))
    #define paCustomFormat  ((PaSampleFormat) (1<<16))
99
    /* Device enumeration mechanism.
     Device ids range from 0 to Pa_CountDevices()-1.
     Devices may support input, output or both.
    */
104
    typedef int PaDeviceID;
    #define paNoDevice -1

    int Pa_CountDevices( void );
109
    typedef struct
    {
        int structVersion;
        const char *name;
114     int maxInputChannels;
        int maxOutputChannels;
        /* Number of discrete rates, or -1 if range supported. */
        int numSampleRates;
        /* Array of supported sample rates, or {min,max} if range supported. */
119     const double *sampleRates;
        PaSampleFormat nativeSampleFormats;
    }
    PaDeviceInfo;

124 /* Pa_GetDefaultInputDeviceID(), Pa_GetDefaultOutputDeviceID() return the default device ids
        for input and output respectively, or paNoDevice if no device is available. The result
        can be passed to Pa_OpenStream().
     On the PC, the user can specify a default device by setting an environment variable.
     For example, to use device #1.
            set PA_RECOMMENDED_OUTPUT_DEVICE=1
     The user should first determine the available device ids by using the supplied application "
        pa_devs".
129 */

    PaDeviceID Pa_GetDefaultInputDeviceID( void );
    PaDeviceID Pa_GetDefaultOutputDeviceID( void );

134 /* Pa_GetDeviceInfo() returns a pointer to an immutable PaDeviceInfo structure for the device
        specified.
     If the device parameter is out of range the function returns NULL.
```

```
     PortAudio manages the memory referenced by the returned pointer, the client must not
         manipulate or free the memory. The pointer is only guaranteed to be valid between calls
         to Pa_Initialize() and Pa_Terminate().
    */

139 const PaDeviceInfo* Pa_GetDeviceInfo( PaDeviceID device );

    /*PaTimestamp is used to represent a continuous sample clock with arbitrary start time that
         can be used for syncronization. The type is used for the outTime argument to the
         PortAudioCallback and as the result of Pa_StreamTime()
    */

144 typedef double PaTimestamp;

    /*PortAudioCallback is implemented by PortAudio clients.

     inputBuffer and outputBuffer are arrays of interleaved samples, the format, packing and
         number of channels used by the buffers are determined by parameters to Pa_OpenStream() (
         see below).
149
     framesPerBuffer is the number of sample frames to be processed by the callback.

     outTime is the time in samples when the buffer(s) processed by this callback will begin
         being played at the audio output.

154  See also Pa_StreamTime()

     userData is the value of a user supplied pointer passed to Pa_OpenStream() intended for
         storing synthesis data etc.

     return value:
159  The callback can return a non-zero value to stop the stream. This may be useful in
         applications such as soundfile players where a specific duration of output is required.
         However, it is not necessary to utilise this mechanism as StopStream() will also
         terminate the stream. A callback returning a non-zero value must fill the entire
         outputBuffer.
     NOTE: None of the other stream functions may be called from within the callback function
         except for Pa_GetCPULoad().
    */

    typedef int (PortAudioCallback)(
164     void *inputBuffer, void *outputBuffer,
        unsigned long framesPerBuffer,
        PaTimestamp outTime, void *userData );

    /*Stream flags
169  These flags may be supplied (ored together) in the streamFlags argument to the Pa_OpenStream
         () function.
    */

    #define   paNoFlag          (0)
    #define   paClipOff         (1<<0)   /* disable default clipping of out of range samples */
174 #define   paDitherOff       (1<<1)   /* disable default dithering */
    #define   paPlatformSpecificFlags (0x00010000)
    typedef   unsigned long PaStreamFlags;

    /*A single PortAudioStream provides multiple channels of real-time input and output audio
         streaming to a client application.
179  Pointers to PortAudioStream objects are passed between PortAudio functions.
    */

    typedef void PortAudioStream;
    #define PaStream PortAudioStream
184
    /*Pa_OpenStream() opens a stream for either input, output or both.

     stream is the address of a PortAudioStream pointer which will receive a pointer to the newly
          opened stream.

189  inputDevice is the id of the device used for input (see PaDeviceID above). inputDevice may
         be paNoDevice to indicate that an input device is not required.

     numInputChannels is the number of channels of sound to be delivered to the callback. It can
         range from 1 to the value of maxInputChannels in the PaDeviceInfo record for the device
         specified by the inputDevice parameter. If inputDevice is paNoDevice numInputChannels is
          ignored.

     inputSampleFormat is the sample format of inputBuffer provided to the callback function.
         inputSampleFormat may be any of the formats described by the PaSampleFormat enumeration
         (see above). PortAudio guarantees support for the device's native formats (
         nativeSampleFormats in the device info record) and additionally 16 and 32 bit integer
         and 32 bit floating point formats. Support for other formats is implementation defined.
194
     inputDriverInfo is a pointer to an optional driver specific data structure containing
         additional information for device setup or stream processing. inputDriverInfo is never
         required for correct operation. If not used inputDriverInfo should be NULL.

     outputDevice is the id of the device used for output (see PaDeviceID above). outputDevice
         may be paNoDevice to indicate that an output device is not required.
```

```
199     numOutputChannels is the number of channels of sound to be supplied by the callback. See the
            definition of numInputChannels above for more details.

        outputSampleFormat is the sample format of the outputBuffer filled by the callback function.
            See the definition of inputSampleFormat above for more details.

        outputSampleFormat is the sample format of the outputBuffer filled by the callback function.
            See the definition of inputSampleFormat above for more details.
204
        outputDriverInfo is a pointer to an optional driver specific data structure containing
            additional information for device setup or stream processing. outputDriverInfo is never
            required for correct operation. If not used outputDriverInfo should be NULL.

        sampleRate is the desired sampleRate. For full-duplex streams it is the sample rate for both
            input and output

209     framesPerBuffer is the length in sample frames of all internal sample buffers used for
            communication with platform specific audio routines. Wherever possible this corresponds
            to the framesPerBuffer parameter passed to the callback function.

        numberOfBuffers is the number of buffers used for multibuffered communication with the
            platform specific audio routines. If you pass zero, then an optimum value will be chosen
            for you internally. This parameter is provided only as a guide - and does not imply
            that an implementation must use multibuffered i/o when reliable double buffering is
            available (such as SndPlayDoubleBuffer() on the Macintosh.)

        streamFlags may contain a combination of flags ORed together. These flags modify the
            behaviour of the streaming process. Some flags may only be relevant to certain buffer
            formats.
214
        callback is a pointer to a client supplied function that is responsible for processing and
            filling input and output buffers (see above for details.)

        userData is a client supplied pointer which is passed to the callback function. It could for
            example, contain a pointer to instance data necessary for processing the audio buffers.

219     return value:
        Upon success Pa_OpenStream() returns PaNoError and places a pointer to a valid
            PortAudioStream in the stream argument. The stream is inactive (stopped).
        If a call to Pa_OpenStream() fails a non-zero error code is returned (see PaError above) and
            the value of stream is invalid.
    */

224 PaError Pa_OpenStream( PortAudioStream** stream,
                          PaDeviceID inputDevice,
                          int numInputChannels,
                          PaSampleFormat inputSampleFormat,
                          void *inputDriverInfo,
229                       PaDeviceID outputDevice,
                          int numOutputChannels,
                          PaSampleFormat outputSampleFormat,
                          void *outputDriverInfo,
                          double sampleRate,
234                       unsigned long framesPerBuffer,
                          unsigned long numberOfBuffers,
                          PaStreamFlags streamFlags,
                          PortAudioCallback *callback,
                          void *userData );
239
    /*Pa_OpenDefaultStream() is a simplified version of Pa_OpenStream() that opens the default
        input and/or output devices. Most parameters have identical meaning to their
        Pa_OpenStream() counterparts, with the following exceptions:
     If either numInputChannels or numOutputChannels is 0 the respective device is not opened.
        This has the same effect as passing paNoDevice in the device arguments to Pa_OpenStream
        ().
     sampleFormat applies to both the input and output buffers.
    */
244
    PaError Pa_OpenDefaultStream( PortAudioStream** stream,
                                  int numInputChannels,
                                  int numOutputChannels,
                                  PaSampleFormat sampleFormat,
249                               double sampleRate,
                                  unsigned long framesPerBuffer,
                                  unsigned long numberOfBuffers,
                                  PortAudioCallback *callback,
                                  void *userData );
254
    /*Pa_CloseStream() closes an audio stream, flushing any pending buffers.
    */

    PaError Pa_CloseStream( PortAudioStream* );
259
    /*Pa_StartStream() and Pa_StopStream() begin and terminate audio processing.
     Pa_StopStream() waits until all pending audio buffers have been played.
     Pa_AbortStream() stops playing immediately without waiting for pending buffers to complete.
    */
264
    PaError Pa_StartStream( PortAudioStream *stream );

    PaError Pa_StopStream( PortAudioStream *stream );
```

```
PaError Pa_AbortStream( PortAudioStream *stream );

/* Pa_StreamActive() returns one (1) when the stream is active (ie playing or recording audio)
     , zero (0) when not playing, or a negative error number if the stream is invalid. The
     stream is active between calls to Pa_StartStream() and Pa_StopStream(), but may also
     become inactive if the callback returns a non-zero value. In the latter case, the stream
     is considered inactive after the last buffer has finished playing.
*/

PaError Pa_StreamActive( PortAudioStream *stream );

/* Pa_StreamTime() returns the current output time in samples for the stream. This time may be
      used as a time reference (for example synchronizing audio to MIDI).
*/

PaTimestamp Pa_StreamTime( PortAudioStream *stream );

/* Pa_GetCPULoad() returns the CPU Load for the stream. The "CPU Load" is a fraction of total
     CPU time consumed by the stream's audio proccssing routines including, but not limited to
     the client supplied callback.
 A value of 0.5 would imply that PortAudio and the sound generating callback was consuming
      roughly 50% of the available CPU time.
 This function may be called from the callback function or the application.
*/

double Pa_GetCPULoad( PortAudioStream* stream );

/* Pa_GetMinNumBuffers() returns the minimum number of buffers required by the current host
      based on minimum latency.
 On the PC, for the DirectSound implementation, latency can be optionally set by user by
      setting an environment variable.
 For example, to set latency to 200 msec, put:
     set PA_MIN_LATENCY_MSEC=200
 in the AUTOEXEC.BAT file and reboot.
 If the environment variable is not set, then the latency will be determined based on the OS.
      Windows NT has higher latency than Win95.
*/

int Pa_GetMinNumBuffers( int framesPerBuffer, double sampleRate );

/* Pa_Sleep() puts the caller to sleep for at least 'msec' milliseconds.
 You may sleep longer than the requested time so don't rely on this for accurate musical
      timing.

 Pa_Sleep() is provided as a convenience for authors of portable code (such as the tests and
      examples in the PortAudio distribution).
*/

void Pa_Sleep( long msec );

/* Pa_GetSampleSize() returns the size in bytes of a single sample in the supplied
     PaSampleFormat, or paSampleFormatNotSupported if the format is no supported.
*/

PaError Pa_GetSampleSize( PaSampleFormat format );

#ifdef __cplusplus
}
#endif /* __cplusplus */
#endif /* PORT_AUDIO_H */
```

C.2 DSPLabDlg C++

```
// DSPLabDlg.cpp : implementation file
//
#include "pa_host.h"
#include "libtsp.h"
#include "stdafx.h"
#include "DSPLab.h"
#include "DSPLabDlg.h"
#include "portaudio.h"
#include "Process.h"
#include ".\dsplabdlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

//-----------------------------------------------------------------------

#define MEMO_ADD(text)                                  m_lbMemo.AddString(_T(text));
    m_lbMemo.SetScrollPos(SB_VERT, m_lbMemo.GetCount());
```

```
#define WM_USER_PD WM_USER+1

#define SAMPLE_RATE                     (16000)
#define FRAMES_PER_BUFFER               (2048)
#define NUMBER_OF_INPUT_CHANNELS        (1)
#define NUMBER_OF_OUTPUT_CHANNELS       (2)

#define LW                                                              (128)
#define awindow                                                 (0.46)

#ifndef M_PI
#define M_PI  (3.14159265)
#endif

//-----------------------------------------------------------------------

struct paData
{
    float*                      pInSamplesChan;
    float*                      pOutSamplesChan;
    unsigned int                SamplesPerBufferIn;
    unsigned int                SamplesPerBufferOut;
    char                        Mode;
    HWND                        FHandle;
};

struct DSPLabFile
{
    AFILE*                                              AFp;
    long int                    Nsamp;
    long int                    Nchan;
    float                                               Sfreq;
    int                         Volume;
    long int                    Counter;
};

//-----------------------------------------------------------------------

struct paData                   Data;
struct paData*                  pData = &Data;
PortAudioStream*                stream;

DSPLabFile                      InputFile;
DSPLabFile                      OutputFile;
DSPLabFile                      OutputFile2;

//-----------------------------------------------------------------------

static int paPlaybackCallback(    void *inputBuffer, void *outputBuffer,
                                  unsigned long framesPerBuffer,
                                  PaTimestamp outTime, void *userData )
{
    paData              *pData = (paData*) userData;
    float               *in = (float*) inputBuffer;
    float               *out = (float*) outputBuffer;
    int                 finished=0;
    unsigned int        i;

    switch(pData->Mode & 1)
    {
    case 0:     /* Silence */
                for( i=0; i<pData->SamplesPerBufferOut; i++ )
                {
                        *out++ = 0;
                }
                break;
    case 1:     /* Process Data */
                for( i=0; i<pData->SamplesPerBufferOut/NUMBER_OF_OUTPUT_CHANNELS; i++ )
                {
                        *out++ = pData->pOutSamplesChan[i];
                                        *out++ = pData->pInSamplesChan[i];
                        }
                        for( i=0; i<pData->SamplesPerBufferIn; i++ )
                {
                        pData->pInSamplesChan[i] = *in++;
                }
                break;
    }
    PostMessage(pData->FHandle, WM_USER_PD, 0, 0);
    return finished;
}

//-----------------------------------------------------------------------

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
```

```
        CAboutDlg();

// Dialog Data
        enum { IDD = IDD_ABOUTBOX };

        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// CDSPLabDlg dialog

CDSPLabDlg::CDSPLabDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDSPLabDlg::IDD, pParent)
        , m_RT(0.446)
        , m_T(0.04)
{
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDSPLabDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        DDX_Control(pDX, IDC_LOAD, m_Load);
        DDX_Control(pDX, IDC_START, m_Start);
        DDX_Control(pDX, IDC_STOP, m_Stop);
        DDX_Control(pDX, IDC_LIST, m_lbMemo);
        DDX_Text(pDX, IDC_EDIT2, m_RT);
        DDX_Text(pDX, IDC_EDIT3, m_T);
}

BEGIN_MESSAGE_MAP(CDSPLabDlg, CDialog)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        //}}AFX_MSG_MAP
        ON_WM_CLOSE()
        ON_BN_CLICKED(IDC_START, OnBnClickedStart)
        ON_BN_CLICKED(IDC_STOP, OnBnClickedStop)
        ON_BN_CLICKED(IDC_LOAD, OnBnClickedLoad)
        ON_MESSAGE(WM_USER_PD, OnProcessData)
        ON_BN_CLICKED(IDCANCEL, OnBnClickedCancel)
END_MESSAGE_MAP()


// CDSPLabDlg message handlers

BOOL CDSPLabDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        // Set the icon for this dialog.  The framework does this automatically when the
             application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon
```

```
        PaError err;

    // Initialize process mode
    pData->Mode = 0;
201     pData->FHandle = this->m_hWnd;

    err = Pa_Initialize();
    if( err != paNoError ) goto error;

206 MEMO_ADD("Initializing PortAudio I/O stream...");

        // Open Portaudio stream
    err = Pa_OpenStream(
            &stream,
211         Pa_GetDefaultInputDeviceID(),           /* default input device */
            NUMBER_OF_INPUT_CHANNELS,               /* number of input channels = 1*/
            paFloat32,                                      /* 32 bit floating
                point input */
            NULL,
            Pa_GetDefaultOutputDeviceID(),  /* default output device */
216         NUMBER_OF_OUTPUT_CHANNELS,              /* number of output channels = 2*/
            paFloat32,                                      /* 32 bit floating
                point output */
            NULL,
            SAMPLE_RATE,                                    /* in the real
                enviroment 16000 is used*/
            FRAMES_PER_BUFFER,                      /* 2048 */
221         0,                                                      /* number of
                buffers, if zero then use default minimum */
            paClipOff,                                      /* we won't output
                out of range samples so don't bother clipping them */
            paPlaybackCallback,                     /* specify our custom
                callback */
            pData );                                        /* pass our data
                through to callback */
    if( err != paNoError ) goto error;
226
    // Calculate number of samples per buffer.
    pData->SamplesPerBufferIn = NUMBER_OF_INPUT_CHANNELS * FRAMES_PER_BUFFER;
    pData->SamplesPerBufferOut = NUMBER_OF_OUTPUT_CHANNELS * FRAMES_PER_BUFFER;

231 // Allocate data buffers
    pData->pInSamplesChan = (float*) malloc(pData->SamplesPerBufferIn * sizeof(float));
    pData->pOutSamplesChan = (float*) malloc(pData->SamplesPerBufferOut * sizeof(float));

        for (unsigned int i=0 ; i<pData->SamplesPerBufferIn ; i++)
236         pData->pInSamplesChan[i] = 0;
        for (unsigned int i=0 ; i<pData->SamplesPerBufferOut ; i++)
            pData->pOutSamplesChan[i] = 0;

    err = Pa_StartStream( stream );
241 if( err != paNoError ) goto error;

        return TRUE;  // return TRUE  unless you set the focus to a control

error:
246 Pa_Terminate();
    MEMO_ADD("Error: An error occured while using the portaudio stream !");
#ifdef _DEBUG
    TRACE("An error occured while using the portaudio stream\n" );
    TRACE("Error number: %d\n", err );
251 TRACE("Error message: %s\n", Pa_GetErrorText( err ) );
#endif
        return FALSE;
}

256 void CDSPLabDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
261         dlgAbout.DoModal();
        }
        else
        {
            CDialog::OnSysCommand(nID, lParam);
266     }
}

// If you add a minimize button to your dialog, you will need the code below to draw the icon
    . For MFC applications using the document/view model, this is automatically done for you
     by the framework.

271 void CDSPLabDlg::OnPaint()
{
        if (IsIconic())
        {
            CPaintDC dc(this); // device context for painting
276
            SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
```

```
                    // Center icon in client rectangle
                    int cxIcon = GetSystemMetrics(SM_CXICON);
281                 int cyIcon = GetSystemMetrics(SM_CYICON);
                    CRect rect;
                    GetClientRect(&rect);
                    int x = (rect.Width() - cxIcon + 1) / 2;
                    int y = (rect.Height() - cyIcon + 1) / 2;
286
                    // Draw the icon
                    dc.DrawIcon(x, y, m_hIcon);
            }
            else
291         {
                    CDialog::OnPaint();
            }
    }

296 // The system calls this function to obtain the cursor to display while the user drags the
        minimized window.
    HCURSOR CDSPLabDlg::OnQueryDragIcon()
    {
            return static_cast<HCURSOR>(m_hIcon);
    }
301
    void CDSPLabDlg::OnClose()
    {
        PaError err;

306     MEMO_ADD("Terminating PortAudio I/O stream...");

        /* Stop Portaudio stream. */
        err = Pa_StopStream( stream );
        if( err != paNoError ) goto error;
311
        /* Wait until Portaudio stream is done. */
        while (Pa_StreamActive(stream)==1);

        /* Close Portaudio stream. */
316     err = Pa_CloseStream( stream );
        if( err != paNoError ) goto error;

        /* Terminate Portaudio. */
        Pa_Terminate();
321
        /* Free data buffers. */
        free(pData->pInSamplesChan);
        free(pData->pOutSamplesChan);

326         /* Close input files. */
            if (InputFile.AFp != NULL)
        {
            AFclose (InputFile.AFp);
        }
331
            CDialog::OnClose();

        error:
            Pa_Terminate();
336         MEMO_ADD("Error: An error occured while using the portaudio stream !");
    #ifdef _DEBUG
                    TRACE("An error occured while using the portaudio stream\n" );
            TRACE("Error number: %d\n", err );
            TRACE("Error message: %s\n", Pa_GetErrorText( err ) );
341 #endif
                    CDialog::OnClose();
    }

    LRESULT CDSPLabDlg::OnProcessData(WPARAM wParam, LPARAM lParam)
346 {
        unsigned int i;
            double maxim;
            float Win[LW];

351     switch(pData->Mode)
        {
        case 3:     // Input -> Output.
                    for( i=0; i<pData->SamplesPerBufferOut; i++ )
                    {
356                         pData->pOutSamplesChan[i] = pData->pInSamplesChan[i];
                    }

                    break;

361     case 9:    // Real stereo wav file.
                    if ((InputFile.AFp != NULL))
                    {
                            AFreadData (InputFile.AFp, InputFile.Counter, pData->pOutSamplesChan,
                                    pData->SamplesPerBufferOut/NUMBER_OF_OUTPUT_CHANNELS);
```

```
366                             InputFile.Counter += pData->SamplesPerBufferOut/
                                    NUMBER_OF_OUTPUT_CHANNELS;
                                if (InputFile.Counter >= InputFile.Nsamp)
                                        InputFile.Counter = 0;
                        }
371                             for (i=0 ; i<pData->SamplesPerBufferIn; i++)
                                {
                        pData->pInSamplesChan[i] = pData->pInSamplesChan[i] * 32768;
                                }
376                             /* Save File Reverberation */
                                if ((OutputFile.AFp != NULL))
                        {
                                        AFwriteData (OutputFile.AFp, pData->
                                            pInSamplesChan, pData->SamplesPerBufferIn
                                            );
                                }
381
                                FIwinHamm (Win, LW, awindow);
                                Process (pData->pInSamplesChan, pData->SamplesPerBufferIn,
                                     Win, m_RT, m_T);

                                /* Save File ProcessOut */
386                             if ((OutputFile2.AFp != NULL))
                        {
                                        AFwriteData (OutputFile2.AFp, pData->
                                            pInSamplesChan, pData->SamplesPerBufferIn
                                            );
                                }
391                             for (i=0 ; i<pData->SamplesPerBufferIn; i++)
                                {
                        pData->pInSamplesChan[i] = pData->pInSamplesChan[i] / 32768;
                                }
396                     /* Adjust amplitude. */
                        for (i=0 ; i<pData->SamplesPerBufferOut/NUMBER_OF_OUTPUT_CHANNELS; i++)
                                pData->pOutSamplesChan[i] = (InputFile.Volume * pData->
                                     pOutSamplesChan[i]) / (32768 * 100);

                                break;
401     }

            return(0);
    }
    void CDSPLabDlg::OnBnClickedStart()
406 {
            UpdateData(TRUE);

        //pData->Mode = (char) pow(2,Mode->ItemIndex) + 1;
            pData->Mode = 9;
411
        if (pData-> Mode == 9 && InputFile.AFp == NULL)
        {
            MEMO_ADD("Error: No input file selected !");

416         return;
        }

        //Mode->Enabled = false;
            m_Stop.EnableWindow(TRUE);
421     m_Start.EnableWindow(FALSE);

        MEMO_ADD("Start processing data.");
    }

426 void CDSPLabDlg::OnBnClickedStop()
    {
        pData->Mode = 0;

        //Mode->Enabled = true;
431         m_Stop.EnableWindow(FALSE);
        m_Start.EnableWindow(TRUE);

            /*Close write files*/
            if (OutputFile.AFp != NULL)
436         {
                    AFclose (OutputFile.AFp);
            }
            if (OutputFile2.AFp != NULL)
            {
441                 AFclose (OutputFile2.AFp);
            }
        MEMO_ADD("Stop processing data.");
    }

446 void CDSPLabDlg::OnBnClickedLoad()
    {
            CFileDialog m_ldFile(TRUE);
```

```
        CString         sResult;

451     // Show the File open dialog and capture the result
        if (m_ldFile.DoModal() == IDOK)
        {
                // Clear buffers
                for (unsigned int i=0 ; i<pData->SamplesPerBufferIn ; i++)
456                     pData->pInSamplesChan[i] = 0;
                for (unsigned int i=0 ; i<pData->SamplesPerBufferOut ; i++)
                        pData->pOutSamplesChan[i] = 0;

                // Get the filename selected
461             sResult = m_ldFile.GetFileName();

                // Update the dialog
                UpdateData(FALSE);

466     // Open wav file and display File Information.
            FILE* fpinfo = NULL;

                InputFile.AFp = AFopenRead(sResult , &InputFile.Nsamp, &InputFile.Nchan, &
                    InputFile.Sfreq, fpinfo);
                OutputFile.AFp = AFopenWrite("rever.wav", 517, NUMBER_OF_OUTPUT_CHANNELS,
                    SAMPLE_RATE, fpinfo);
471             OutputFile2.AFp = AFopenWrite("out.wav", 517, NUMBER_OF_OUTPUT_CHANNELS/2,
                    SAMPLE_RATE, fpinfo);
        InputFile.Volume = 50;
        MEMO_ADD("Loading wav-file...");
        }
    }
```

C.3 Process C++

```
// Process.cpp : implementation file
//
4 #include "pa_host.h"
#include "libtsp.h"
#include "Process.h"
#include <math.h>
#include <stdio.h>
9 #include <stdlib.h>

//


#define SAMPLE_RATE                     (16000)
14 #define FRAMES_PER_BUFFER              (2048)
#define NUMBER_OF_INPUT_CHANNELS        (1)
#define NUMBER_OF_OUTPUT_CHANNELS       (2)

#define OVERLAP         (75)
19 #define BETA           (0.9)
#define LANDA          (0.1)
#define LW             (128)
#define awindow        (0.46)
#define TF             (256)
24
const int SHIFT = LW-(LW*OVERLAP)/100;
const int Nrow = (FRAMES_PER_BUFFER)/SHIFT;
```

```
#ifndef M_PI
#define M_PI  (3.14159265)
#endif

float A[Nrow][TF];                  //Matrix where the frames are saved
    after the FT
float Reserv[LW-SHIFT];             //Vector with some samples for the
   reconstruction of the FT
float ReserS[LW-SHIFT];             //Samples saved for the next block
float M[Nrow][(TF/2)+1];            //Matrix with the module of the FT
float P[Nrow][(TF/2)+1];            //Matrix with the phase of the FT
float E[Nrow][(TF/2)+1];            //Matrix with for the estimation
float GAIN[Nrow][(TF/2)+1];         //Matrix with the gain
float S[Nrow][(TF/2)+1];            //Matrix with the estimated signal
float NewSamples[FRAMES_PER_BUFFER+LW-SHIFT];   //Vector with the
   samples of the last block and the new samples to process all
float e[Nrow][(TF/2)+1];            //Used in the estimation
float Resere[Nrow][(TF/2)+1];       //Used in the estimation
float m_RT;                         //Reverberation Time
float m_T;                          //T = 40 or T = 60
int value = 0;


void Process (float Samples[], int NSamples, float Win[], float
   m_rt, float m_t)
{
    int i;

    m_RT = m_rt;
    m_T = m_t;
        //SHIFT OF THE WINDOW
    const int DELAY = (m_T*SAMPLE_RATE)/32;
        //INITIALIZATIONS
    if (value == 0)
    {
        VRfZero (Reserv, LW-SHIFT);
        VRfZero (ReserS, LW-SHIFT);

        for (i=0; i<(LW-SHIFT); i++)
        {
            NewSamples[i] = 0;
        }
        for (i=0; i<NSamples; i++)
        {
            NewSamples[i+LW-SHIFT] = (Samples[i]);
        }
    }
    else
    {
```

```
74          for (i=0; i<(LW-SHIFT); i++)
            {
                NewSamples[i] = (ReserS[i]);
            }
            for (i=0; i<NSamples; i++)
79          {
                NewSamples[i+LW-SHIFT] = (Samples[i]);
            }
        }
            //PROCESS
84      STFFT (NSamples, Win);
        PhaseModule ();
        Estimation (DELAY);
        Signal ();
        ISTFFT (NSamples, Win);
89          //SAVE THE ESTIMATED SIGNAL
        for (i=0; i<(LW-SHIFT); i++)
        {
            ReserS[i] = Samples[NSamples-LW+SHIFT+i];
        }
94      for (i=0; i<NSamples; i++)
        {
            Samples[i] = (NewSamples[i]);
        }
    }
99
    void STFFT (int NSamples, float Win[])
    {
        int i,j,k;
        float u[TF];
104
        k = 0;

        for (i=0; i<NSamples; i=i+SHIFT)
        {
109         for (j=LW; j<TF; j++)
            {
                u[j] = 0;
            }
            for (j=0; j<LW; j++)
114         {
                u[j] = NewSamples [i+j]*Win[j]; //samples windowed
            }
            SPfRFFT (u, TF, 1);       //Fast Fourier Transform
            for (j=0; j<TF; j++)
119         {
                A[k][j] = u[j];       //Matrix with the FT. One column =
                    frame
            }
            k++;
        }
```

```
124 }

    void ISTFFT (int NSamples, float Win[])
    {
        int i,j,k;
129     float x[LW];
        float u[TF];

        k = 0;
        VRfZero (x, LW);
134
        for (i=0; i<NSamples; i=i+SHIFT)
        {
            for (j=0; j<TF; j++)
            {
139             u[j] = A[k][j];
            }
                //Inverse Fast Fourier Transform
            SPfRFFT (u, TF, -1);
            for (j=0; j<(LW-SHIFT); j++)
144         {
                x[j] = Reserv[j] + u[j]*Win[j];
                    //Reconstruction of the signal with the window and
                        the result of the last three IFT
            }
            for (j=(LW-SHIFT); j<LW; j++)
149         {
                x[j] = u[j]*Win[j];
                    //Reconstruction of the signal with the window
            }
            for (j=0; j<(LW-SHIFT); j++)
154         {
                Reserv[j] = x[j+SHIFT];
                    //Save the result of the last three IFT for the
                        next
            }
            for (j=0; j<SHIFT; j++)
159         {
                NewSamples[i+j] = x[j]/1.6;
                    //Amplitude added by the reconstruction of the
                        window
            }
            k++;
164     }
    }

    void PhaseModule ()
    {
169     float a,b,m,f;
        unsigned int i,j;
            // MODULE AND PHASE OF THE FT
```

```
        for (i=0; i<Nrow; i++)
        {
174         j = 0;
            m = A[i][j];
            M[i][j] = sqrtf(m*m);
            P[i][j] = 0;
            for (j=1; j<(TF/2); j++)
179         {
                a = A[i][j];
                b = A[i][j+(TF/2)];
                m = sqrtf(a*a+b*b);
                f = atan2 (b,a);
184             M[i][j] = m;
                P[i][j] = f;
            }
            j = (TF/2);
            m = A[i][j];
189         M[i][j] = sqrtf(m*m);
            P[i][j] = 0;
        }
    }

194 void Estimation (const int DELAY)
    {
        int i,j,t;
        float c,d;

199     t = DELAY;
        c = (3*logf(10))/m_RT;
        d = 1/expf(2*c*m_T);
            //INITIALIZATIONS
        if (value == 0)
204     {
            for (i=0; i<Nrow; i++)
            {
                for(j=0; j<((TF/2)+1); j++)
                {
209                 e[i][j] = 0;
                    Resere[i][j] = 0;
                }
            }
            value = 1;
214     }
            //ESTIMATION OF THE ACTUAL BLOCK, FOR ALL THE FRAMES
        for (i=0; i<Nrow; i++)
        {
            if (i == 0)
219         {
                for (j=0; j<((TF/2)+1); j++)
                {
```

```
                    e[i][j] = BETA*e[Nrow-1][j] + (1-BETA)*(M[i][j]*M[i
                        ][j]);
                }
            }
            else
            {
                for (j=0; j<((TF/2)+1); j++)
                {
                    e[i][j] = BETA*e[i-1][j] + (1-BETA)*(M[i][j]*M[i][j
                        ]);
                }
            }
        }
            //INITIALIZATIONS OF THE ESTIMATIONS
        for (i=0; i<Nrow; i++)
        {
            for(j=0; j<((TF/2)+1); j++)
            {
                E[i][j] = 0;
            }
        }
            //THE FIRST PART OF THE ESTIMATION IS FROM THE LAST BLOCK
        for (i=0; i<t; i++)
        {
            for(j=0; j<((TF/2)+1); j++)
            {
                E[i][j] = (Resere[i][j])*(d);
            }
        }
            //THE SECOND PART IS FROM THE ACTUAL PROCESS
        for (i=t; i<Nrow; i++)
        {
            for(j=0; j<((TF/2)+1); j++)
            {
                E[i][j] = (e[i-t][j])*(d);
            }
        }
            //WE SAVE THE REST OF THE ESTIMATION FOR THE NEXT BLOCK
        for (i=0; i<t; i++)
        {
            for(j=0; j<((TF/2)+1); j++)
            {
                Resere[i][j] = e[Nrow-t+i-1][j];
            }
        }
    }

    void Signal ()
    {
        unsigned int i,j;
```

```
        for (i=0; i<Nrow; i++)
        {
            for(j=0; j<((TF/2)+1); j++)
274         {
                GAIN[i][j] = 1-((sqrtf(E[i][j]))/M[i][j]);
                if (M[i][j] == 0 || E[i][j] == 0)
                {
                    GAIN[i][j] = 1;
279             }
                S[i][j] = GAIN[i][j]*M[i][j];    //Estimated signal
                if (S[i][j] < LANDA*M[i][j])     //Correction of the
                    signal and the gain
                {
                    S[i][j] = LANDA*M[i][j];
284                 GAIN[i][j] = LANDA;
                }
                    //RECONSTRUCCION WITH THE PHASE
                A[i][j] = S[i][j]*cosf(P[i][j]);
                if(j!=0 || j!=(TF/2))
289             {
                    A[i][j+(TF/2)] = S[i][j]*sinf(P[i][j]);
                }
            }
        }
294 }
```

Appendix D

Matlab files

The constants explained below are used in the Matlab functions:

```
1
2       sec = 4;                    %seconds of the speech signal
3       fs = 8000;                  %sample frequency
4       w = 'hamming';              %type of window
5       olap = 75;                  %overlap expressed in
6       lw = 128;                   %length of Hamming window
7       beta = 0.9;                 %beta for the estimation function
8       landa = 0.1;                %used for the spectral subtraction
9       T = 40E-3;                  %also has the value 60E-3
10      fixedRT = 0.448236;         %reverberation time for the room
```

D.1 STFFT

```
1
2  function X = stfft(x,w,shift,lw,l)
3
4
5       win = window(w,lw)';
6       c = 1;
7       for b = 0:shift:(l-lw)
8           u = win.*x((b+1):(b+lw));
9           u = [u zeros(1,128)];
10          t = fft(u);
11          X(:,c) = t';
12          c = c+1;
13      end
```

D.2 ESTIMATION

```
1
2  function [E] = estimation(Y, fs ,T, beta ,fixedRT ,step)
3
4      [m,n] = size(Y);          %where Y = abs(X); (the module)
5      e = zeros(m,n);
6      for b = 1:n
7          if(b==1)
8              e(:,1) = (1-beta)*((Y(:,b)).^2);
9          else
10             e(:,b) = beta*e(:,b-1)+(1-beta)*((Y(:,b)).^2);
11         end
12     end
13
14     STE = 1/(10^(step/10));
15
16     t = fix(T*fs/32);
17     c = ((3*log(10))/(fixedRT));
18     E = zeros(m,n);
19     E1 = zeros(m,n);
20     E(:,t:n) = ((exp(-2*c*T))/STE)*e(:,1:n-t+1);
```

D.3 SIGNAL

```
1
2  function [S,gain] = signal2(Y,E,landa)
3
4      gain = (1.-(sqrt(E)./Y));
5      nan = isnan(gain);
6      [i,j] = find(nan==1);
7      n = length(i);
8      for b = 1:n
9          gain(i(b),j(b)) = 1;
10     end
11
12     S = gain.*Y;
13
14     [i,j] = find(S<landa*Y);
15     n = length(i);
16     for b = 1:n
17         S(i(b),j(b)) = landa*Y(i(b),j(b));
18         gain(i(b),j(b)) = landa;
19     end
```

D.4 ISTFFT

```
1
2  function s = istfft(S,P,shift,lw,l,w)
3
4      Sc = S.*exp(i*P);          %where P = angle(X); (the phase)
5      win = window(w,lw)';
6
7      [m,n] = size(Sc);
8      for c = 1:n
9          sc = real(ifft(Sc(:,c)'));
10         y(:,c) = sc';
11     end
12
13     s = zeros(1,m);
14     e = 1;
15     for b = 0:shift:(l-lw)
16         s((b+1):((m/2+b))) = [s(b+1:b+(m/2)-shift)+((y(1:((m/2)-
               shift),e)').*win(1:(m/2)-shift)) (y(((m/2)-shift+1):(m
               /2),e)').*win((m/2)-shift+1:(m/2))];
17         e = e+1;
18     end
19
20     s = s./1.6;
```

Bibliography

[1] O'Shaughnessy, *Speech Communication.* Addison-Wesley, 1987.

[2] R. Ratman, D. L. Jones, B. C. Wheeler, W. D. O. Jr., C. R. Lansing, and A. S. Feng, "Blind estimation of reverberation time," *Journal of the Acoustical Society of America*, vol. 114, pp. 2877–2892, 2004.

[3] E. A. P. Habets, "Multi-channel speech dereverberation based on a statistical model of late reverberation," *ICASSP 2005.* accepted for publication.

[4] E. D. Geest and R. Garcea, "Simulation of room transmission functions using a triangular beam tracing computer model," *Applications of Signal Processing to Audio and Acoustics, 1995. IEEE ASSP Workshop on*, pp. 253–256, Oct 1995.

[5] F. A. Everest, *The Master Handbook of Acoustics.* Mc Graw Hill, 4th ed., 2001.

[6] J. R. Hopgood, *Nonstationary Signal Processing with Application to Reverberation Cancellation in Acoustic Environments.* PhD thesis, University of Cambridge, Nov. 2000.

[7] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders, *Fundamentals of Acoustics.* John Wiley and Sons, Inc., 4th ed., 2000.

[8] M. Tohyama and T. Koike, *Fundamentals of Acoustic Signal Processing.* Academic Press, 1998.

[9] Y. Haneda and Y. Kaneda, "Interpolation and extrapolation of room transfer functions based on common acoustical poles and their residues," *Applications of Signal Processing to Audio and Acoustics, 1997. 1997 IEEE ASSP Workshop on*, Oct. 1997.

[10] L. Savioja, *Modeling Techniques for Virtual Acoustics.* PhD thesis, Helsinki University of Technology, Aug. 2000.

[11] M. Kahrs and K. Brandenburg, *Applications of Digital Signal Processing to Audio and Acoustics.* Kluwer Academic Publishers, 1998.

[12] M. R. Schroeder., "A new method of measuring reverberation time," *Journal of the Acoustical Society of America*, vol. 37, pp. 409–412, 1965.

[13] W. T. Chu., "Comparison of reverberation measurements using schroeders impulse," *Journal of the Acoustical Society of America*, vol. 63, pp. 1444–1450, 1978.

[14] D. C. Bees, "Enhancement of acoustically reverberant speech using cepstral methods," Master's thesis, Montreal, Canada, July 1990.

[15] K. Lebart and J. M. Boucher, "A new method based on spectral subtraction for speech dereverberation," *Acta Acoustica*, vol. 87, pp. 359–366, 2001.

[16] M. Berouti, R. Schwartz, and J. Makhoul, "Enhancement of speech corrupted by acoustic noise," *IEEE ICASSP'79*, vol. 4, pp. 208–211, 1979.

[17] J. Allen and L. Radiner, "A unified approach to short-time fourier analysis and synthesis," *IEEE Proceedings*, vol. 65, 1977.

[18] LIBTSP, *The package libtsp homepage.*
URL: http://www.tsp.ece.mcgill.ca/MMSP/Documents/Software/Packages/
/libtsp/libtsp.html,
June 2005.

[19] PortAudio, *The PortAudio homepage.*
URL: http://www.portaudio.com/,
June 2005.