isvr Institute of Sound and Vibration Research

An Investigation of the Design and Implementation of Cross Talk Cancellation Filters for Virtual Acoustic Imaging

D. Tavan, P. Mannerheim and P.A. Nelson

ISVR Technical Memorandum No 951

July 2005

University of Southampton

SCIENTIFIC PUBLICATIONS BY THE ISVR

Technical Reports are published to promote timely dissemination of research results by ISVR personnel. This medium permits more detailed presentation than is usually acceptable for scientific journals. Responsibility for both the content and any opinions expressed rests entirely with the author(s).

Technical Memoranda are produced to enable the early or preliminary release of information by ISVR personnel where such release is deemed to the appropriate. Information contained in these memoranda may be incomplete, or form part of a continuing programme; this should be borne in mind when using or quoting from these documents.

Contract Reports are produced to record the results of scientific work carried out for sponsors, under contract. The ISVR treats these reports as confidential to sponsors and does not make them available for general circulation. Individual sponsors may, however, authorize subsequent release of the material.

UNIVERSITY OF SOUTHAMPTON

INSTITUTE OF SOUND AND VIBRATION RESEARCH

FLUID DYNAMICS AND ACOUSTICS GROUP

An Investigation of the Design and Implementation of Cross Talk Cancellation Filters for Virtual Acoustic Imaging

by

D Tavan, P Mannerheim and P A Nelson

ISVR Technical Memorandum No. 951

July 2005

Authorized for issue by
Professor R J Astley, Group Chairman

Contents

FIGURES IV
TABLES IV
ABSTRACT V

1 INTRODUCTION 1

2 THE SYSTEM CURRENTLY IN USE 5

3 HARDWARE SPECIFICATIONS 7

3.1 PROCESSOR (CPU) 7
3.2 MEMORY (RAM) 8
3.3 SOUND CARD 8
3.4 A/D AND D/A CONVERSION 9

4 SOFTWARE SPECIFICATIONS 10

4.1 IDENTIFICATION OF THE SOFTWARE ENVIRONMENT 10
4.2 FUNCTIONS TO BE IMPLEMENTED 10
4.3 CHARACTERIZATION OF THE FUNCTIONS 11
4.4 SUMMARY TABLE 13
4.5 OTHER CONSIDERATION FOR THE SOFTWARE SPECIFICATIONS 14

5 EXISTING SOLUTIONS FOR VIRTUAL IMAGING ON A PC 15

5.1 AURORA PLUG-IN SUITE FOR ADOBE AUDITION 15
5.2 SIR PLUG-IN WITH HOST APPLICATION 17
5.3 REAL-TIME CONVOLUTION PERFORMANCE 19
5.4 FAST DECONVOLUTION ALGORITHM PERFORMANCE 20

6 POTENTIAL SOLUTIONS FOR THE SOFTWARE DESIGN 21

6.1 OPEN SOURCE CLASS LIBRARIES TO CONSIDER FOR THIS PROJECT 23
6.1.1 Real-time Audio Input/Output 23
6.1.2 Real-time Audio Convolution 24
6.1.3 Summary Table 25
6.2 WHICH LIBRARIES FOR OUR SOFTWARE DESIGN ? 26
6.2.1 Which library for Real-time Input/Output ? 26
6.2.2 Which library for Real-time Convolution ? 26

7 HEARING CROSS-TALK CANCELLATION PERFORMANCE 28

8 CONCLUSIONS 29

REFERENCES 30

APPENDIX 33

Figures

FIG. 1: TRANSMISSION PATHS WITHIN THE CONTEXT OF SOUND REPRODUCTION OVER TWO STEREO LOUDSPEAKERS. (*AFTER* [5]) 2

FIG. 2: FILTER NETWORK FOR CROSS-TALK CANCELLATION 3

FIG. 3: *STEREODIPOLE* ARRANGEMENT (AFTER [6]) 3

FIG. 4: OPTIMAL SOURCE DISTRIBUTION ARRANGEMENT (HF = HIGH-FREQUENCIES; MF = MID-FREQUENCIES; LF = LOW-FREQUENCIES) 4

FIG. 5: VIRTUAL SOURCE IMAGING SYSTEM IN USE AT THE AUDIO LABORATORY OF THE ISVR 5

FIG. 6: DIAGRAM OF THE HARDWARE INTERFACE 9

FIG. 7: FUNCTIONAL DIAGRAM FOR THE SOFTWARE DESIGN 10

FIG. 8: VIEW OF THE REAL-TIME CONVOLUTION PLUG-IN FOR ADOBE AUDITION 16

FIG. 9: PLOGUE BIDULE CONFIGURATION PATCH FOR REAL-TIME CROSS-TALK CANCELLATION 17

FIG. 10: VIEW OF THE SIR VST PLUG-IN WITH A STEREO IMPULSE RESPONSE LOADED 18

FIG. 11: PROPOSED SOFTWARE IMPLEMENTATION 21

Tables

TABLE 1: SOFTWARE SPECIFICATIONS 13

TABLE 2: REAL-TIME CONVOLUTION PERFORMANCE LIMIT ON A PENTIUM PRO 200 MHZ (AFTER [16]) 19

TABLE 3: REAL-TIME CONVOLUTION PERFORMANCE LIMIT ON A PENTIUM 4 2.8 GHZ 19

TABLE 4: CLASS LIBRARIES COMPARISON CHART 25

Abstract

This report aims at giving an overview of the design issues to consider for a real-time implementation of the required processing for virtual source imaging, that is to say "cross-talk cancellation". This includes an assessment of the hardware needed, an analysis of currently available software solutions for carrying out this processing on a standard PC, as well as a study of a possible design model. The target platform is a standard MS-Windows PC and the programming language to be used is C++.

1 Introduction

At the ISVR researchers have for the last ten years been working on using digital signal processing to improve the quality of sound reproduction systems. The ultimate goal is to be able to produce the illusion in a listener of being in a virtual acoustic environment which is entirely different from that of the space in which the listener is actually located. Sound systems designed for this purpose are usually referred to as *Surround Sound* systems, or *3D-Audio* systems. At the ISVR, the term *Virtual Imaging* system is preferred.

These systems rely on the *Binaural Technology*, whose principle is that producing the correct ear sound pressures should lead to almost the same sensation as the listener would experience in the real sound field. One first needs to get *Binaural Recordings*, which are obtained by capturing sounds of any given type by employing in the recording setup a realistic representation of the human head, ears and torso, called a *Dummy-head*, with microphones used in place of the ear drums to convert the sound into an electrical signal. Then, these recordings have to be reproduced, bearing in mind that the left-channel signal has to reach the listener's left ear only and the right-channel the right ear only. Of course, there is the possibility to use headphones to solve this problem but for several reasons[1] it is preferred to reproduce binaural recordings over loudspeakers.

The challenge has therefore been taken at the ISVR to make it possible to reproduce accurately *Binaural Recordings* over loudspeakers, which is not an easy task. Actually, to reproduce at the listener's ears signals that are identical to the recorded ones, it is necessary to cancel all the effects applied to the signals to be reproduced by the environment in which are situated both the listener and the loudspeakers. In other words, this means that all effects that can modify the recorded signals prior to their transmission to the loudspeakers (as electrical signals) and until they reach the listener's eardrums (as acoustical waves) have to be cancelled. The most damaging effect is related to the comb filtering resulting from the interference of the direct and cross-talk signals. But the unlinearities in the loudspeakers' response, the room reverberation, the numerous reflections on the listener's body and all the other acoustical parameters that can damage the signals given out by the loudspeakers need not to be forgotten. Hence, given the numerous damaging effects that can't be identified independently from the others, it is better to treat the system as a whole.

[1] There are many drawbacks related to the use of headphones. For instance the fact that the sound often seems being either too close or inside the head (since the transducers are very close to the eardrums), or that headphones may not be comfortable to wear for a long time period. Another drawback of the reproduction through headphones, in comparison with listening to a real sound source, is that the head movement has no effect on localisation of the sound, a situation that does not correspond to actual circumstances. Some recent techniques consist of adding a head-tracking system on the headphones to put this right, but it makes the system a lot more complex.

1. *Identification of the system as a whole*

This consists of measuring by means of a dummy-head the response of the loudspeaker system over which binaural recording are to be reproduced. A recording of a loudspeaker response made with a dummy-head is referred to as a *Head-Related Transfer Function* (HRTF).

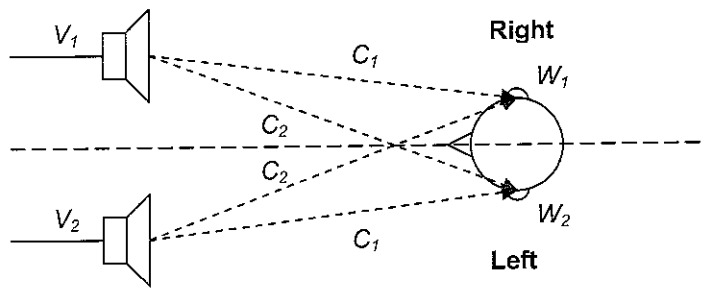

Fig. 1: Transmission paths within the context of sound reproduction over two stereo loudspeakers. (*After* [5])

In the Fig. 1 above each acoustical path corresponds to an HRTF given that:

$$C_1 = \left.\frac{W_1}{V_1}\right|_{V_2=0} = \left.\frac{W_2}{V_2}\right|_{V_1=0} \quad \text{and} \quad C_2 = \left.\frac{W_2}{V_1}\right|_{V_2=0} = \left.\frac{W_1}{V_2}\right|_{V_1=0} \tag{1}$$

Hence, we need a set of four HRTFs in order to identify the system, also referred to as the *"plant"*.

Many HRTF databases are made available for researchers, like that from the MIT for example [1], freely distributed over the Internet and therefore widely used by researchers. More recently, the ISVR has measured its own HRTF database [2], thus obtaining the most extensive HRTF database yet available. However, since the latter was conducted by the *ISVR Samsung Joint Lab*, it is only available to the *Virtual Acoustics Project* at the ISVR.

2. *Inversion of the identified system*

Once the system has been identified and represented as a matrix **C** of four HRTFs, as shown below, the idea is then to invert this matrix.

$$\mathbf{w} = \mathbf{C}\mathbf{v} \quad \text{where} \quad \mathbf{C} = \begin{bmatrix} C_1 & C_2 \\ C_2 & C_1 \end{bmatrix}, \ \mathbf{v} = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}, \ \mathbf{w} = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} \tag{2}$$

Then, by applying the model of the inverted system **H** to the signals we want to reproduce, we ideally obtain at the listener's ears the recorded signals unchanged since multiplying the plant by its inverse gives an identity system between source (binaural recordings) and receiver (listener's eardrums).

Given $\mathbf{H} = \begin{bmatrix} H_1 & H_2 \\ H_2 & H_1 \end{bmatrix}$ and $\mathbf{H} = \mathbf{C}^{-1}$ we have $\mathbf{I} = \mathbf{CH}$ (3)

Considerable work on this inverse filtering method for sound reproduction has been undertaken at the ISVR. It is referred to as *Cross-talk cancellation* since its biggest contribution is to cancel the effect of the cross-talk signals (represented by the C_2 path on Fig.1 above), and the algorithm used to invert the plant system is referred to as the *Fast Deconvolution Method Using Regularization* [3], [4].

The Fig. 2 below shows how the so called *cross-talk cancellation filters* are used to process the source signals before they are reproduced over the loudspeakers. What we ideally obtained is therefore a reproduction without cross-talk signals.

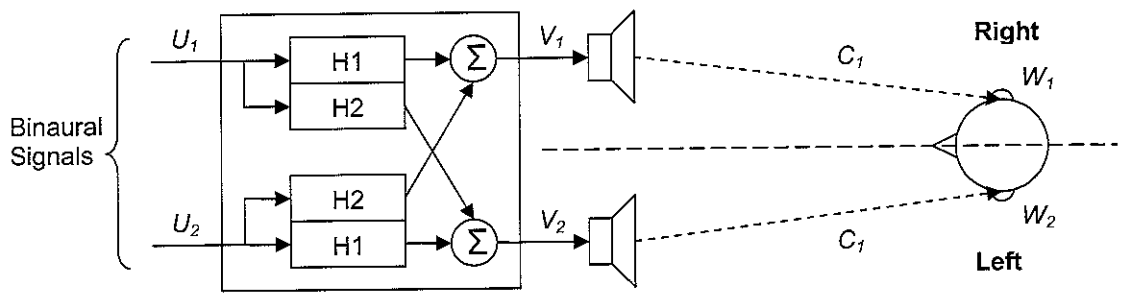

Fig. 2: Filter network for cross-talk cancellation

However, this processing is in itself not sufficient, since it doesn't ensure that the reproduction is robust to the listener's head movements. But it has been discovered that the loudspeaker position has a great influence on the efficiency of the cross-talk signals cancellation. This resulted in the finding that a loudspeaker span angle of 10° (as shown on Fig. 3 below) gives the best results for a two-channel system. This specific arrangement is referred to as the *StereoDipole* or *SD* [5].

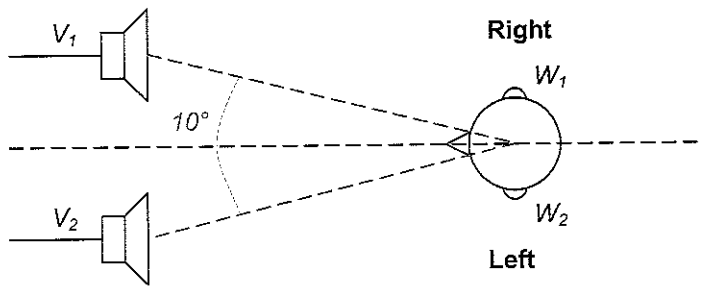

Fig. 3: *StereoDipole* arrangement (After [6])

More recently, difficulties faced with using the *StereoDipole* as well as improvements in the reproduction quality have been provided by a new system design referred to as *Optimal Source Distribution* or *OSD* [7]. Basically, this is based on the observation that by varying the loudspeaker span depending on the frequency range, the sound reproduction becomes much more accurate. The Fig. 4 below shows the typical arrangement for this system.

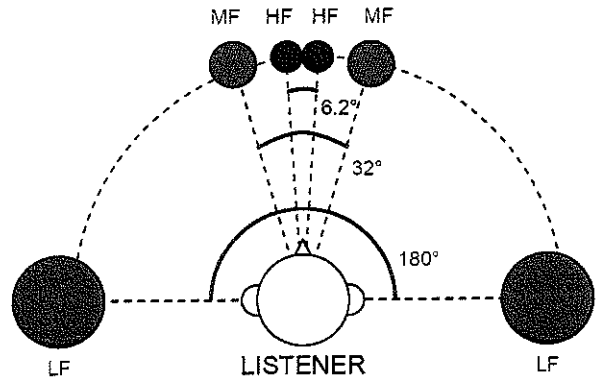

Fig. 4: Optimal Source Distribution arrangement
(HF = High-Frequencies; MF = Mid-Frequencies; LF = Low-Frequencies)

Following the findings made in the course of this research, a system has been implemented on a purpose-built DSP platform. The latter carries out real-time convolution of the binaural recording with pre-computed cross-talk cancellation filters. This system works for a single listener situated at a defined position.

But it turns out that the system is currently evolving to something that can also work for a listener moving in a room. This is referred to as *Visually Adaptive Imaging* and is achieved by using a video camera and image processing techniques to localize the listener in the room. It requires that the cross-talk cancellation filters are re-calculated each time the listener moves, since the filters' coefficients depend on the listener's location. Hence, in this situation it is a problem to have the filters computed off-line. It has been tried to pre-compute several sets of filters for several listener's locations, and then to switch in real-time between the filters as the listener was moving. But, apart from the poor resolution (since the number of calculated positions is inevitably limited), there is an audible noise each time the system switches between the filters. The only solution is therefore to update the cross-talk cancellation filters in real-time.

The purpose of this report is therefore to introduce an analysis of the design of a real-time implementation of the processing described above. The primary requirement is to enable the system to be implemented in a recent PC since the DSP platform features no longer meet the requirements of the ongoing project. Also, given the massive advances in recent years in computer processing technology, the requirements of software based virtual acoustic imaging systems are likely to be met by relatively simple processing schemes.

2 The system currently in use

The Virtual Imaging system currently used at the ISVR is an implementation of the principle of Optimal Source Distribution (OSD), as described on the Fig. 5 below.

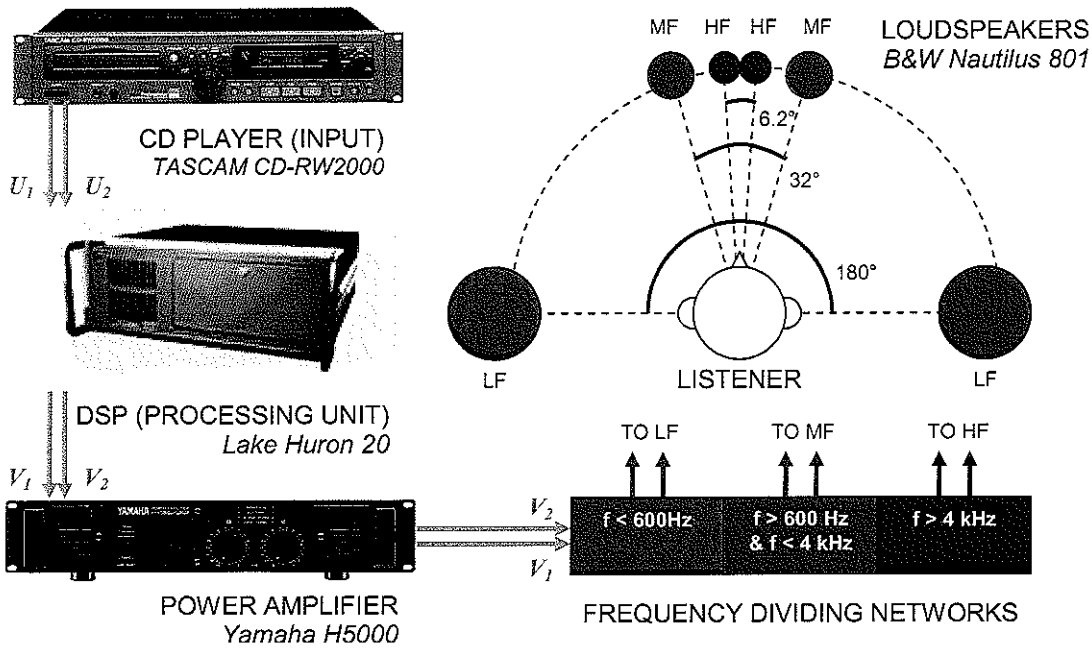

Fig. 5: Virtual source imaging system in use at the Audio Laboratory of the ISVR

The processing is currently undertaken on a purpose-built DSP system that has been designed to enable real-time convolution of the input signals with specified Finite Impulse Response filters. This system, the *Huron 20* made by *Lake Technology Ltd* [8], is referred to as a *Multi-channel Digital Audio Convolution Workstation* and implements the famous hybrid zero-delay convolution algorithm patented by Lake Technology and clearly described by Gardner [9]. The convolution in itself is undertaken by a series of DSP boards mounted in a host PC (*equipped with an Intel CPU - Pentium II MMX 350 Mhz with 128 Mo RAM*) from which the DSP boards can be controlled from an MS-Windows OS.

The cross-talk cancellation filters are computed off-line by means of a Matlab script implementing the *fast deconvolution method using regularisation* [10] on selected HRTFs. The computation is made by using the set of HRTFs corresponding to the loudspeaker arrangement that will be used to reproduce the binaural signals, according to the position of the loudspeakers relative to the listener. It is possible to compute the filters directly onto the Huron20 (off-line), using the host PC, or to compute them on another computer and then import them on the Huron Workstation. In both cases, after the filters are computed in Matlab, they need to be converted to a proprietary format.

The Matlab version shipped with the station contains specific tools to convert and also to control the DSP boards of the Huron Workstation. Finally, once in the good format, the filters are sent to the DSP units to be convolved in real-time with the binaural signals, which are given in input to the processing unit by means of a CD player.

The convolution engine provides good results in real-time. But the characteristics of the host machine are very limited, so that it is not possible to consider using it for the last evolutions of the Virtual Acoustic project (real-time computation of cross-talk cancellation filters and image processing). Moreover, the quality of the embedded A/D and D/A converters is rather poor. Hence, an implementation on a better platform is justified.

Documentation for the *Huron20* and also for the other components of the system (CD player, Amplifier, Loudspeakers) is available on the CD-ROM enclosed with this report (in the *Documentation/Technical* folder).

3 Hardware Specifications

The following assessment is about the equipment we need to design a replacement processing unit for the *Huron 20*. This analysis is widely inspired by the work of Anders Torger on an implementation of a real-time convolution algorithm [11], [12].

Since the system to be designed has to replace the current unit, it will have to acquire the source signal to be processed from the same CD player, and to send the processed signals to the same sound reproduction system. Hence, we need a PC equipped with a sound card whose inputs can be connected to the external CD player and whose outputs can be connected to the external amplifier.

The question is, can we use the soundcard supplied with the computer (a cheap embedded soundcard) to make real-time quality processing ? The answer is, surely not ! Nowadays, all embedded soundcards are of AC'97 compatible type. These soundcards are said to be full-duplex and have a reasonable quality for multimedia applications. But it is far from being sufficient for a professional audio signal processing application, since the latency and throughput are most of the time not sufficient. We therefore considered purchasing a professional sound card. Since our development is in some ways similar to the work of Anders Torger [11], we followed his advice regarding which model of sound card to buy.

Finally, we also have to wonder if the performance of a standard PC will be sufficient for our implementation. But there is nothing to worry about since, as it is shown later in this document, any PC currently available on the market is able to undertake the processing for virtual imaging in real-time.

3.1 Processor (CPU)

For the real-time implementation of cross-talk cancellation systems, Torger recommends at least a 500 MHz processor, just for the real-time convolution of the cross-talk cancellation filters [13]. In our case, we will need to process the inversion of the *Plant Matrix*[2] in real-time, and also the image processing for listener's location tracking. But given the performances of state-of-the-art PC now available on the market, there is no doubt about the capability of these stations to handle such processing. The one we have chosen is an Intel Pentium 4 CPU with Hyper-Threading, working at 2.8 GHz with 512Ko of L1 Cache memory. According to Torger and Farina [12], an Intel Pentium 4 seems to be a good choice in comparison with an AMD Athlon, especially because the FFT computation (used for processing real-time convolution) works better on Intel processor than on AMD ones. However, whatever processor we use, we have to bear in mind that any other runtime performance altering technologies must be avoided,

[2] The matrix of electro-acoustic transfer functions describing the system whose input are the loudspeaker's input signals and whose outputs are the sound pressures at the listener's ears.

since the engine used for real-time crosstalk cancellation will load the processor constantly at high load. Any performance drop may cause buffer underflow and stop the audio processing.

3.2 Memory (RAM)

Torger recommends 256 megabytes of RAM or more for his application [13]. The more RAM, the more filter programs can be kept simultaneously in memory. In our case, we will use less (and shorter) filters than him, so it is essentially the speediness of the memory that is important. The faster the memory, the better, since the engine will be both very processor and memory intensive. For high-performance systems, fast memory is as important as a fast processor. But we neither have to worry about this since the computer we are using is equipped with 1GB of High-speed RAM (533MHz DDR2 ECC SDRAM Memory).

3.3 Sound card

It is strongly recommended to make use of a high quality digital sound card, and employ external A/D and D/A converters, since a computer is not a good environment for analog signals.

An important feature of the sound card is that it must be full duplex, that is to say it must be able to receive input at the same time as it generates output. It is possible to use one sound card for input, and another for output, but it is not recommended, mainly since the I/O-delay[3] cannot be guaranteed with such a configuration, and sample clock synchronisation may be problematic. If the sound card supports synchronous starting of input and output, then I/O-delay will be fixed. It is also good if the sound card can operate with short interrupt cycles, since it allows for shorter I/O-delay. If the input is digital, the sound card must be configured to work as clock slave, that is to say to adapt to the clock signal received on the input.

Currently, the RME Audio HDSP sound card [14] is recommended since it supports all features mentioned above. The RME HDSP9632 is therefore the model to be purchased for the need of the project. Moreover, the RME HDSP9632 sound board also has the following very important features:

- **Supports ASIO 2.0** *(we talk about the importance of this characteristic in the analysis of the software design later in this document)*
- **0% CPU Load** *(the CPU resources available for the software are thus optimized)*
- **All settings can be changed in real-time**

[3] I/O-delay is the time it takes from the first sample is received on the input until its processed version is available on the output.

- **Customizable buffer size** *(control over the latency)*
- **Zero latency monitoring** *(full real-time monitoring of the sound card)*
- ***SyncAlign*** **function that guarantees samples aligned and never swapping channels**
- **Fully compatible with Hyper-threading** *(using this sound card with a last generation Intel® Pentium 4 platform is thus optimized)*
- **ADAT and S/PDIF digital interfaces**

3.4 A/D and D/A conversion

As observed above, it is strongly recommended that the A/D and D/A conversion are not made inside the machine running the software, but instead in external dedicated converters. The analog performance is generally considered to be better in external converters than those that need to cope with the rather hostile environment (for analog technology) existing inside a computer.

Common digital standards for digital transmission are S/PDIF and ADAT. S/PDIF supports stereo 44.1 - 48 kHz, and ADAT supports eight channels in 44.1 – 48 kHz. ADAT is transported over an optical link, S/PDIF through a coaxial cable or an optical link.

The A/D - D/A converter we have chosen is from RME, as for the sound card. Actually, this company also produces high quality A/D and D/A converters with an ADAT interface. We are thus able to get a piece of equipment which is fully compatible with the HDSP9632 sound card. We have chosen the ADI-2, a high-performance 192 kHz/24 Bit 2-channel AD/DA-converter [15].

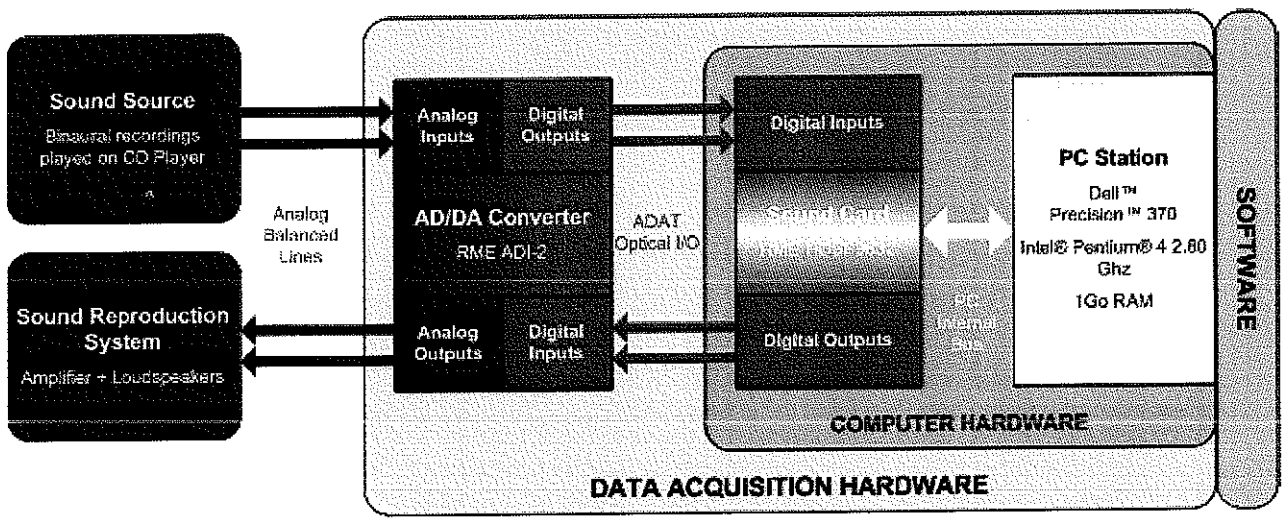

Fig. 6: Diagram of the hardware interface

4 Software Specifications

4.1 Identification of the software environment

The **software** will be designed to work on a standard PC workstation. The **user** will interact with it (to adjust its parameters, offline or in real-time) using the common controls available with any workstation, that is to say a screen, a keyboard and a mouse. The software will also have to interact with an **input system**, a CD player from which the binaural recordings are played, and an **output system**, a sound reproduction system reproducing the processed signals, this by means of a sound card embedded in the workstation. The source signals acquired from the **input system** have to be *convolved in real-time* with a set of *cross-talk cancellation filters*. These filters are computed by the *fast deconvolution algorithm* which undertakes the inversion of *HRTFs* loaded from a **database** and selected according to the **output system** arrangement. The coefficients of the *cross-talk cancellation filters* can be updated as a function of the listener's position given by a **head-tracking device**.

4.2 Functions to be implemented

These functions explain the link between the software that has to be designed and the elements of the software's environment.

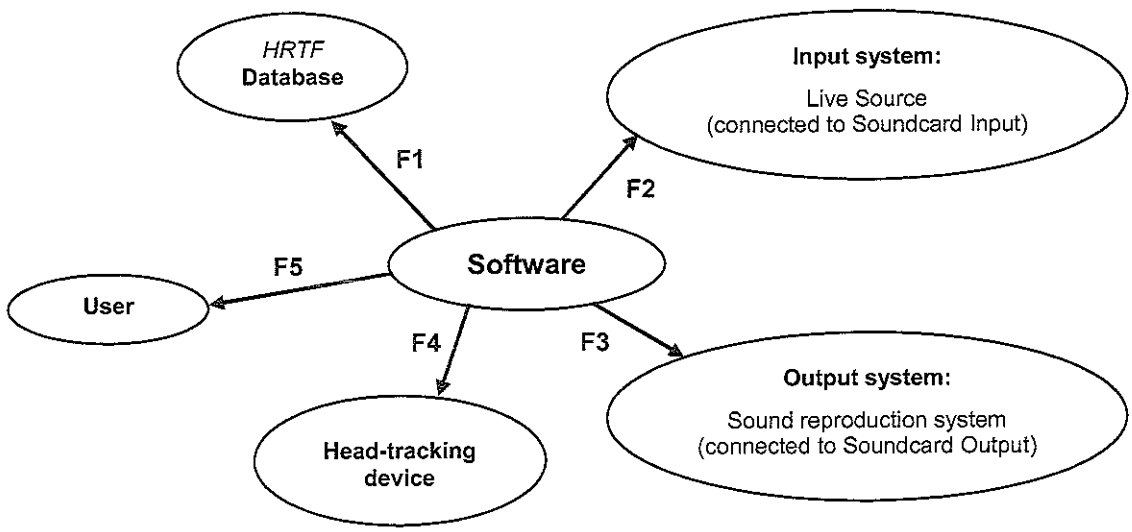

Fig. 7: Functional diagram for the software design

The analysis of the need has lead to define the following functions:

- **F1 :** Compute cross-talk cancellation filters in real-time
- **F2 :** Convolve filters with live source in real-time
- **F3 :** Display processed signals on a sound reproduction system in real-time
- **F4 :** Update filters' coefficients as a function of listener's position in real-time
- **F5 :** Interacts with user in real-time

4.3 Characterization of the functions

Each function is characterized by one or more acceptance criteria (defining the requirements that need to be met in order to validate the function) with a given level (required to validate this function).

- **F1 :** Compute cross-talk cancellation filters in real-time
  - **C1 :** Number of filters computed simultaneously
    - **L1 :** 4 filters
  - **C2 :** Size of computed filters
    - **L2 :** 4096 taps minimum (and up to 32,000 taps)
  - **C3 :** Computation time
    - **L3 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)

- **F2 :** Convolve filters with live source in real-time
  - **C1 :** Number of filters to convolve simultaneously
    - **L1 :** 4 filters
  - **C2 :** Number of input channels to convolve with the filters
    - **L2 :** 2 channels
  - **C3 :** Size of filters to convolve
    - **L3 :** 4096 taps minimum (and up to 32,000 taps)
  - **C4 :** Computation time
    - **L4 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)

- **F3 :** Display processed signals on a sound reproduction system in real-time
  - **C1 :** Number of channels to reproduce
    - **L1 :** 2 channels
  - **C2 :** Latency
    - **L2 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)

- **F4 :** Update filters' coefficients as a function of listener's position in real-time
  - **C1 :** To define
    - **L1 :** To define

- **F5 :** Interacts with user in real-time
  - **C1 :** To define
    - **L1 :** To define

4.4 Summary Table

Table 1: Software specifications

Functions	Criteria	Levels
F1 : Compute cross-talk cancellation filters in real-time	**C1 :** Number of filters computed simultaneously	**L1 :** 4 filters
	C2 : Size of computed filters	**L2 :** 4096 taps minimum (and up to 32,000 taps)
	C3 : Computation time	**L3 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)
F2 : Convolve filters with live source in real-time	**C1 :** Number of filters to convolve simultaneously	**L1 :** 4 filters
	C2 : Number of input channels to convolve with the filters	**L2 :** 2 channels
	C3 : Size of filters to convolve	**L3 :** 4096 taps minimum (and up to 32,000 taps)
	C4 : Computation time	**L4 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)
F3 : Display processed signals on a sound reproduction system in real-time	**C1 :** Number of channels to reproduce	**L1 :** 2 channels
	C2 : Latency	**L2 :** Short enough to give the software true real-time capabilities (ideally, a few milliseconds)
F4 : Update filters' coefficients as a function of listener's position in real-time	**C1 :** To define	**L1 :** To define
F5 : Interacts with user in real-time	**C1 :** To define	**L1 :** To define

4.5 Other consideration for the software specifications

This project also aims to provide the research team with a core software implementation of the system they have developed. At present the only implementation they have is the research script made under Matlab for the fast deconvolution method [10] we have referred to previously in this document.

Since this software is for research use it has to respect the following requirements:

- *Be Fully configurable* (i.e., access to all possible parameters)
- *Be Easily adaptable* (i.e., easily modifiable by other people and connectable with other modules)
- *Be Updatable in real-time* (i.e., the modifications of some parameters has to be effective instantly without producing any noticeable damage on the sound quality).

Hence, we obviously need the following methods and technologies in order to achieve a proper software design:

- *Object-Oriented programming*: It allows any programmer to re-use very easily and efficiently the work of other people and simplifies the re-use of what is to be designed within this project.
- *UML specification*: It makes the design be very clear to other people not deeply involved in the design of the software, or to those who will try to re-use the code later.
- *Open-Source project*: It gives the possibility for other people working on the same kind of system to re-use the code and eventually to improve it or to add other functionalities. For instance, even if this project only aims at getting the software to work on a windows platform, another developer may after want to re-use this work to get it to work under Linux.

This way of developing the virtual acoustic imaging software will guide us through most of the decision we will have to take regarding the design.

5 Existing solutions for Virtual Imaging on a PC

Before going deeply into the software development issues, it is interesting to have a look at existing software packages that make it possible to achieve the necessary processing for virtual imaging on a standard PC with a minimum amount of configuration. Actually, it is nowadays possible to perform the same task as the Huron convolution workstation, that is to convolve in real-time a set of pre-computed filters with a live source. Moreover, the components needed are most of the time freeware or low-cost shareware, and it doesn't take more than 5 minutes to set up and configure all the elements needed for a complete cross-talk cancellation network.

5.1 Aurora plug-in suite for Adobe Audition

This first solution doesn't really fit in the description given above since the processing can only be done from a source file edited in *Adobe Audition*. Actually Audition is primarily for audio editing, so inputs and outputs to the processing performed within this software are always audio files. However, the *Real-time Convolver* plug-in for Adobe Audition, which is part of the *Aurora* plug-in suite designed by Angelo Farina, has a preview function that allows the user to listen to the result of the convolution in real-time without writing any file. It is then possible to perform the convolution on the entire file, and thus obtain a pre-convolved waveform.

Aurora is shareware. It is possible to download it from Angelo Farina's website [16], try it, and then buy a license if you plan to use the plug-in suite.

Adobe Audition is a £250 software, but it is possible to download it from Adobe's website [17] and try it for free during 30 days.

The procedure to achieve real-time cross-talk cancellation using Aurora for Adobe Audition is as follows:

1) Edit in Adobe Audition the waveform of the binaural recording you want to reproduce. Some binaural waveforms are available on the CD-ROM enclosed with this report in the *Data\Waveforms\Binaural Recordings* folder.

2) Launch Aurora's *Real-time Convolver* plug-in.

3) Load a set of cross-talk cancellation filters (*L_XtalkFilter.wav* in *IR #1* and *R_XtalkFilter.wav* in *IR #2*). You can either compute the filters using Matlab [10] or use the pre-computed ones located in the *Data\Impulse Responses\Pre-computed cross-talk cancellation filters* folder on the CD-ROM. Have a look at the Appendix for a description of the files containing the filters. The pre-computed filters are for the *StereoDipole* arrangement. To get advice on how to set up correctly a *StereoDipole*, you can read the *readme.txt* file located in the

Data\Waveforms\Pre-convolved Binaural Recordings folder on the CD-ROM. This folder also contains some pre-convolved waveforms that can be used to check if the positioning of the loudspeakers and the listener are good for cross-talk cancellation. Refer the section of this document called *"Hearing cross-talk cancellation performance"* in order to know how to assess the system's efficiency.

4) Set the parameters as on the Fig. 8 below.

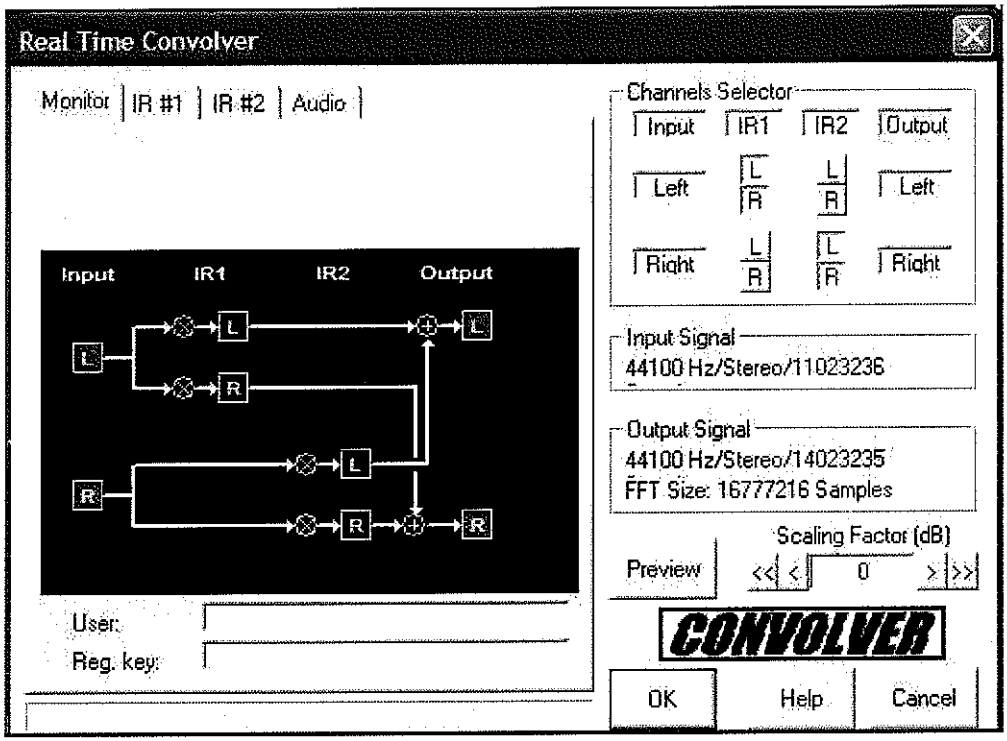

Fig. 8: View of the Real-time convolution plug-in for Adobe Audition

5) Click on the *Audio* tab and select the audio device you want to use.

6) Click on the *Preview* button and listen to the result !

7) Clicking on the *OK* button will process the waveform edited in Adobe Audition. This can be used to pre-process binaural waveforms with a given cross-talk cancellation network in order to obtain pre-convolved binaural waveforms like those available on the CD-ROM in the *Data\Waveforms\Pre-convolved Binaural Recordings* folder.

However, this solution is not the best to perform real-time cross-talk cancellation since it is not possible to convolve the filters with a live source.

5.2 SIR plug-in with host application

Here is a solution that can replace the Huron convolution workstation. Actually, using *SIR*, a VST plug-in implementing a real-time stereo frequency domain convolver, and a host application like *Plogue Bidule*, to connect the convolver with the soundcard's inputs and outputs, one can set up a real-time cross-talk cancellation in about 5 min !

SIR is freeware, downloadable from the Christian Knufinke's website [18].

Plogue Bidule is shareware. It can be downloaded from *Plogue Art et Technologie*'s website [19] and freely used for about 2 months. The registration fees to use the application beyond this limit are moderate.

The procedure to achieve real-time cross-talk cancellation using the SIR VST plug-in within the Plogue Bidule host application is as follows:

1) Open the "patch" file *xtalkcancel.bidule* for Plogue Bidule contained in the archive *SIR_Plogue_Bidule.zip* located in the *Source\Plogue Bidule* folder of the CD-ROM. You should obtain a patch similar the one shown on Fig. 9 below.

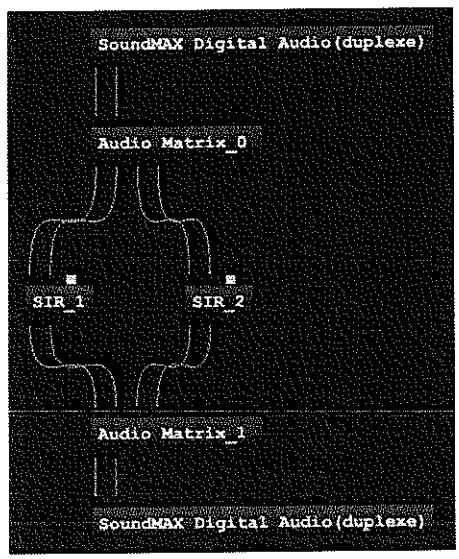

Fig. 9: Plogue Bidule configuration patch for real-time cross-talk cancellation

However, check that the audio device loaded by Plogue Bidule is the one you want to use, as this device may differ from the one originally used when this patch was created.

2) Load a set of cross-talk cancellation filters in the SIR plug-in (*L_XtalkFilter.wav* in *SIR_1* and *R_XtalkFilter.wav* in *SIR_2*). You can either compute the filters using Matlab [10] or use the pre-computed ones located in the *Data\Impulse Responses\Pre-computed cross-talk cancellation filters* folder on the CD-ROM. Have a look at the Appendix for a description of the files

containing the filters. The pre-computed filters are for the *StereoDipole* arrangement. To get advice on how to set up correctly a *StereoDipole*, you can read the *readme.txt* file located in the *Data\Waveforms\Pre-convolved Binaural Recordings* folder on the CD-ROM. This folder also contains some pre-convolved waveforms that can be used to check if the positioning of the loudspeakers and the listener are good for cross-talk cancellation. Refer the section of this document called *"Hearing cross-talk cancellation performance"* in order to know how to assess the system's efficiency.

3) Load the configuration files for each Impulse response processor (*SIR_1.fxb* for *SIR_1* and *SIR_2.fxb* for *SIR_2*). These files can also be found in the archive located in the *Source\Plogue Bidule* folder of the CD-ROM. The Fig. 10 below shows SIR.

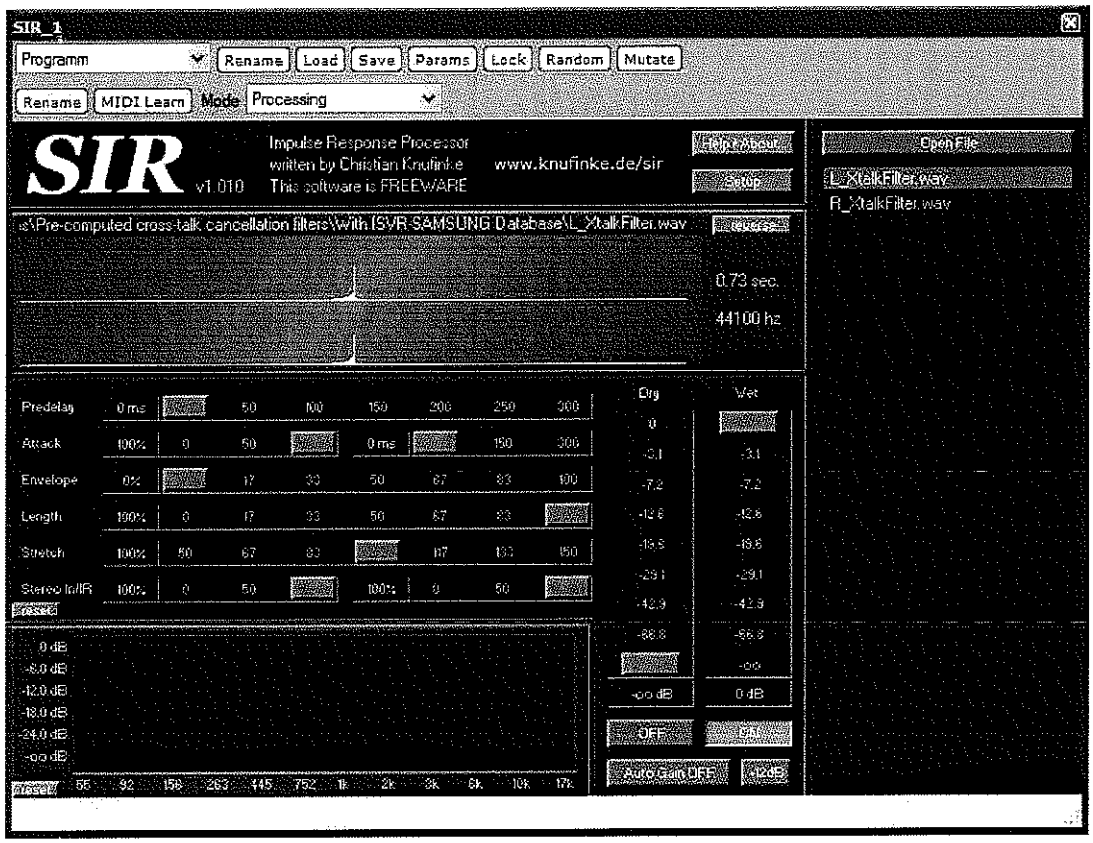

Fig. 10: View of the SIR VST plug-in with a stereo impulse response loaded

4) Play a binaural recording from the external CD player connected to your soundcard's input and listen to the result on your loudspeakers arranged in *StereoDipole* configuration.

This is probably the easiest to use and most effective way to perform real-time cross-talk cancellation on a PC, provided that the station is equipped with a decent soundcard.

5.3 Real-time convolution performance

By looking for existing solutions for processing cross-talk cancellation on a PC, we were also looking for a way to assess quickly the capabilities of this new architecture. Basically this assessment consists in evaluating the maximum filter size that can be convolved in real-time by the computer.

This study has been done for some real-time convolvers (like the Aurora plug-in described above) on pretty old stations, and gives results which already outperform DSP-based real-time convolver like the Huron 20 ! The Table 2 below shows the real-time performance limits of Aurora's convolution plug-in on a PC with a Pentium Pro 200 Mhz , 256 k cache CPU, working with 44.1 kHz waveforms:

Table 2: Real-time convolution performance limit on a Pentium Pro 200 Mhz (After [16])

Description	*N. of channels of the source signal*	*N. of channels of the Impulse Response(s)*	*Impulse Response Length (taps)*
mono input with mono IR	*1*	*1*	*1,000,000 (probably even more, there were no larger IR for testing...)*
mono input with stereo IR	*1*	*2*	*512,000*
stereo input with stereo IR	*2*	*2*	*300,000*
stereo input with 2 stereo IRs	*2*	*4*	*14,000*

The Table 3 below shows the result of the same assessment done on a state-of-the-art PC with a Pentium 4 2,8 Ghz, 512k cache CPU :

Table 3: Real-time convolution performance limit on a Pentium 4 2.8 Ghz

Description	*N. of channels of the source signal*	*N. of channels of the Impulse Response(s)*	*Impulse Response Length (taps)*
stereo input with 2 stereo IRs	*2*	*4*	*3,000,000 (probably even more, there were no larger IR for testing...)*

In other words, there is not any real limit on the filter size with last generation state-of-the-art stations since their calculation power is really huge. However, it has to be noticed that convolving such large filters drastically increases the latency.

Another assessment done with BruteFIR [11], a real-time convolution engine for Linux, shows that it is possible to run a cross-talk cancellation network using 8192 taps filters with a delay as low as 3 millisecond, which is the limit of the RME Hammerfall soundcard we are using (look at section 3.3 of this document). Even if it can't be really expected to get such good results under Windows, since BruteFIR is highly optimized for Linux and the latter is less resource-consuming than Windows, these results are very promising.

The first tests undertaken with the solutions described above have shown that cross-talk cancellation can be efficiently undertaken under Windows with a latency of 12ms for 32000 taps filters, which is enough for our purpose.

5.4 Fast deconvolution algorithm performance

Then, about the computation of the crosstalk cancellation filters, the reference [20] gives us an indication on the computational efficiency of the fast deconvolution algorithm. Actually, in 2001 (4 years ago, which corresponds to an eternity when it is about computer performance!) it was possible to compute these filters using the fast deconvolution algorithm in less than 100 ms.

In order to evaluate the performance of the fast deconvolution algorithm on last generation PC, we only have Matlab at present, which is not the best reference to take into account for the computation efficiency since it is a very high level programming language, consequently not at all as optimized as a C/C++ implementation can be. However, the results obtained are already promising. A set of 4 filters of 4096 taps each is computed in less than 200 ms and the same set with 32000 taps filters is computed in about 1.5 second. Since the filter size for cross-talk cancellation will usually be contained between these two limits, and since a C++ implementation of the fast deconvolution algorithm will drastically decrease the computation time, we can say without taking too many risks that a full real-time implementation of the processing for cross-talk cancellation will work on most of the last generation PCs.

6 Potential solutions for the software design

According to the software specifications stated in section 4 of this document, a possible implementation of the software has been defined. Below is a diagram showing the structure of the proposed implementation.

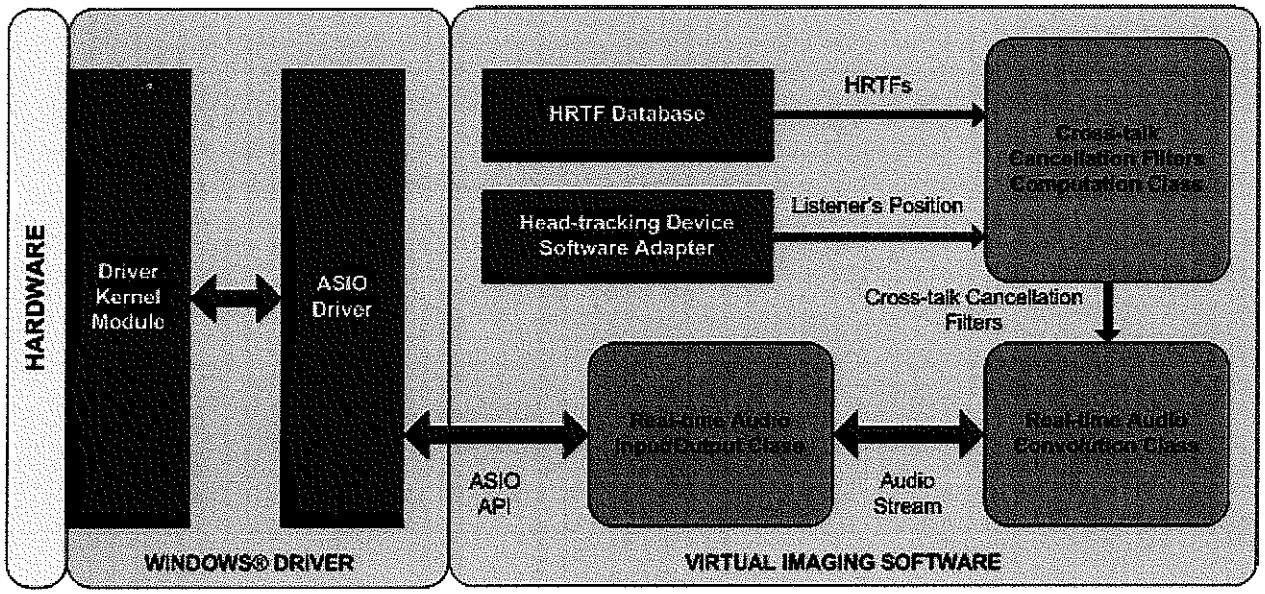

Fig. 11: Proposed software implementation

The software is to be written in C++ since it is the most popular object-oriented programming language. It is therefore the best choice in order to find ready-to-use class libraries.

The target platform is a PC with a MS-Windows operating system since it is the most popular platform.

As remarked previously, a great advantage of object-oriented programming is the possibility of re-using easily other programmers' work. This let us focus only on the design of the project-specific components without having to re-write the additional components we need, which are not the real purpose of our project.

Thanks to open source projects, it is possible to freely re-use a huge number of class libraries available all over the Internet. And after having looked at a great quantity of audio programming related resources over the web, it turns out that a great part of the basic functions we need for this software have already been implemented as C++ class libraries.

Among all the components showed on the diagram above, only three are specific to the project and therefore not yet available as C++ class libraries. These are:

- **The implementation of the fast deconvolution method**, used to compute the cross-talk cancellation filters. The only existing implementation from which we can base our work is a Matlab script, recently re-written by the author [10]. Have a look at the Appendix for a description of the filter files generated with this script.

  Writing the C++ implementation from this Matlab code should be easy since the code re-writing has been done thinking about general implementation issues.

- **The management of an HRTF database**, whose elements are processed by the fast deconvolution algorithm to give the cross-talk cancellation filters. The Matlab script referred to above also includes functions to manage the communication with two existing HRTF databases:

  - A sample HRTF database[4]
  - The ISVR/SAMSUNG HRTF database

  Within the C++ implementation, the management of the HRTF database might be part of the class implementing the fast deconvolution method. The main issue will certainly be the definition of a format for the database. Do we have to keep it in a Matlab-like format or to convert it to a software specific format (that we previously have to define)? This question is still to be answered.

  The work on this implementation can be inspired by the work of J. Bescos on a C++ class library for auralization [21] which contains a class for HRTF database management (*hrtf.h/cpp*) and the work of B. Lehman on the same sort of class (*spatializer.h/cpp*) which is part of the CREATE Signal Library [22].

- **The communication with the head-tracking device software adapter**, whose design work is undertaken by Pål Mannerheim, PhD student at the ISVR. Here also, the communication should be made using a method within the class implementing the fast deconvolution algorithm.

Then, the other components we need for this software design have already been implemented in C++ by other teams. These are:

- **The real-time convolution engine**, used to convolve the cross-talk cancellation filters with a live audio stream in real-time.

- **The real-time audio input/output adapter,** necessary to interface the software (mainly, the convolution engine) with (almost) any kind of audio driver on different platforms with simple access methods. This allows a faster design since one doesn't have to learn how to handle a particular kind of audio driver on a particular platform. It also gives the possibility to make a platform and hardware independent software. Nonetheless, in our case we will focus on a MS-Windows implementation using ASIO drivers (for low latency).

[4] This database is enclosed with the Matlab script. It was made up after the MIT HRTF database.

Below are the descriptions of the packages we can use for this project. We will consider the functionalities as much as the performances of these libraries, to check if they meet the requirements stated in the software specifications.

6.1 Open source class libraries to consider for this project

6.1.1 Real-time Audio Input/Output

There are mainly two libraries, quite similar to each other, which are used by most of the Real-time Audio related software project one can find over the Internet.

6.1.1.1 PortAudio

PortAudio [23] is a free, cross platform, open source, audio I/O library. It lets you write simple audio programs in 'C' that will compile and run on many platforms including Windows, Macintosh (8,9,X), Unix (OSS), SGI, and BeOS. *PortAudio* is intended to promote the exchange of audio synthesis software between developers on different platforms.

Source version: V18.1
Last Update: 06/2003

Since this library was originally written in C, a C++ wrapper is needed in order to include properly this library within a full C++ implementation. Fortunately, this wrapper is available on the website dedicated to the library. MS-Visual C++ project files and a tutorial are also provided for a quick assessment of the library.

6.1.1.2 RtAudio

RtAudio [24] is a set of C++ classes which provide a common API (Application Programming Interface) for real-time audio input/output across Linux (native ALSA, JACK, and OSS), Macintosh OS X, SGI, and Windows (DirectSound and ASIO) operating systems. This is also an open source library, and has been inspired by the *PortAudio* library described above. It is therefore very similar to *PortAudio*, but with the advantages of a full Object-Oriented implementation and real cross-platform capabilities which allow the programmer to build applications that are at the same time Platform and Audio Hardware independent.

Source version: 3.0.1
Last Update: 03/2004

MS-Visual C++ project files and tutorial are provided for a quick assessment of the library.

6.1.2 Real-time Audio Convolution

Here also, there are mainly two libraries containing the functions we are looking for. It can be remarked that the *CSL* library relies on the *PortAudio* Library described above where as *SndObj* has its own Audio Input/Output Class.

6.1.2.1 SndObj

The Sound Object (*SndObj*) Library [25] is an open source, object-oriented audio processing library. It provides objects for synthesis and processing of sound that can be used to build applications for computer-generated music. The core code, including sound file and text input/output, is fully portable across several platforms. Platform-specific code includes real-time audio IO and MIDI input support for Linux (OSS, ALSA and Jack), Windows (MME and ASIO), MacOS X (CoreAudio, but no MIDI at moment), Silicon Graphics (Irix) machines and any Open Sound System-supported UNIX.

Source version: 2.6.1a
Last Update: 06/2005

Binaries for Windows (compiled with cygwin g++ and MS-Visual C++ 6.0) are available. There is no available tutorial, but the MS-Visual C++ project is full of demo applications and a comprehensive user manual is provided (available on the CD-ROM enclosed with this report in *Documentation\Class Libraries\SndObj*).

6.1.2.2 CREATE Signal Library (CSL)

The CREATE Signal Library (*CSL*) [22] is a portable general-purpose software framework for sound synthesis and digital audio signal processing. It is implemented as an open source C++ class library to be used as a stand-alone synthesis server, or embedded as a library into other programs.

Source version: 3.2
Last Update: 08/2004

MS-Visual C++ project files are provided for a quick assessment of the library. See the *README* file in the directory where you have uncompressed the *CSL* source archive for an introduction to writing program with *CSL*, and follow the path to MS Visual Studio project files given above for examples.

6.1.3 Summary Table

Table 4: Class libraries comparison chart

Library Name	Version	Language	Real-time Audio Input/Output			Real-time Audio Convolution		
			Capability	Class Name	Supported Platforms and Audio Drivers	Capability	Class Name	Method
PortAudio	V18.1	C (but a C++ wrapper is available)	Yes	See wrapper documentation	Windows (DirectSound, MME, ASIO), Linux (OSS), MacOS (Sound Mgr for OS 7-9, CARBON, Core Audio for OS X), SGI (Irix), BeOS. Jack and ALSA support for Linux are under development for V19.	No		
RtAudio	3.0.1	C++	Yes	RtAudio	Windows (DirectSound, ASIO), Linux (ALSA, Jack, OSS), MacOS X (Core Audio), SGI (Irix)	No		
SndObj	2.6.1a	C++	Yes	SndIO	Windows (MME, ASIO), Linux (ALSA, Jack, OSS), MacOS X (CoreAudio), SGI (Irix)	Yes	Convol	FFT-based fast convolution
CSL	3.2	C++	Yes (Using PortAudio)	PAIO	See Above	Yes	Convolver	FFT-based fast convolution using W Gardner's low-latency scheduling technique. Rely on the FFTW3 library.

6.2 Which libraries for our software design ?

6.2.1 Which library for Real-time Input/Output ?

For this functionality the *RtAudio* library is preferred, essentially according to its ease of use since it is fully implemented in C++ and well commented. A custom-made MS-Visual C++ project is available on the CD-ROM enclosed with this report (in *Source\C++\RtAudio*). The application computed from this project establishes a Real-time connection with the computer's soundcard. Although this script is not doing anything functionally speaking (the input stream is just copied to the output), it allows the user to test the full-duplex capabilities of the soundcard with both ASIO and DirectX drivers. The application uses a configuration file (*"testrtaudio.ini"* located in *Source\C++\RtAudio\Debug*) which contains the parameters for the connection with the audio device. This configuration file has to be in the same directory as the application when the latter is launched.

The *RtAudio* library is a general purpose Audio Input/Output library. Therefore, it doesn't do anything else. However, this library is also part of the *STK* library [26], which has more classes for real-time audio processing. Unfortunately, the *STK* library doesn't contain any class for performing real-time convolution.

6.2.2 Which library for Real-time Convolution ?

Once we have a way to acquire and output audio data in real-time, the next step consists in implementing the processing for real-time convolution. As stated above, two libraries can provide us with an implementation of a real-time audio convolver. After a quick analysis of the two libraries according to the clarity of the documentation, the test project files provided and the general ease of use, it has been decided to try implementing the real-time audio convolver by using the *SndObj* library. This library has the particularity to have its own Real-time Input/Output classes. Hence, it covers the need for both the Real-time Input/Output and Real-time Convolution functionalities we need. In this situation, the *RtAudio* library is no more needed. A custom made MS-Visual C++ project is available on the CD-ROM (in *Source\C++*) with several demo applications:

- **InfoASIO** to get a list of available ASIO devices[5]
- **Duplex** to establish a full-duplex connection with the default ASIO device
- **ConvLoPass** to convolve a Low-Pass filter with a live source
- **ConvImpulse** to convolve any Impulse Response with a live source
- **ConvXtalkFilters** to convolve Cross-talk Cancellation filters with a live source

[5] All the demo applications for SndObj have been designed to work for ASIO devices only.

These demo applications are very simple to understand and to use. To try them, just extract to your hard-drive the *SndObj.zip* archive contained in the *Source\C++* folder of the CD-ROM. Then you can either use the pre-compiled binaries for Windows available in the *bin* folder, or compile the applications from MS-Visual C++. Finally, you must launch the applications from a console window. To get usage guidance, launch any of the applications without arguments. It is recommended to use **32000** for the filter size parameter. For applications using external files (like *ConvImpulse* and *ConvXtalkFilters*) the files containing the impulse responses have to be in the same folder as the application when the latter is launched.

IMPORTANT COMMENT:

Some classes of the *SndObj* library had to be modified in order to get the applications to work. So be careful if you try working from a version of the library freshly downloaded from the internet.

1) In *SndASIO.cpp*

The last lines of the *DriverName* method have been modified. You should have:

```
if(num < 10)
strcpy(name, drivernames[num]);
return name;
```

2) In *SndTable.cpp*

The *MakeTable* method has been modified so that the values loaded in the table are not normalized. Actually, the normalisation doesn't suit the cross-talk cancellation filters, so this functionality has been cancelled by commenting the related code lines in the *MakeTable* method code.

Moreover, since the values contained in the tables are not anymore normalised, one has to be careful with the scaling factor used by the *Convol* class for real-time convolution. Actually, this parameter has been set to **0.2f** and shouldn't be changed in order to avoid saturation.

7 Hearing cross-talk cancellation performance

Below is guidance about how to assess cross-talk cancellation performance, extracted from Torger's user's guide for AlmusVCU [13]:

"The cross-talk cancellation performance is simply decided by how wide the sound stage seem to be. If there is no cross-talk cancellation at all, as for regular stereo, the sound-stage is not wider than the speakers are spaced. If the performance is good, sound sources can be heard far to the sides. Exactly how far to the side a sound can be heard is dependent on the source material, and to some extent the listener. This means that during tuning the same source material should be used, and the same listener should do the listening.

A good source material is a left/right channel pink noise. The farther out to the sides the pink noise is heard, the better is the cancellation performance. The problem with pink noise is that compared to natural recordings it shows great differences between individuals (one listener may hear the noise at +/- 45 degrees, another at +/- 60), and that it may have some unpredictable effects (hearing the sound from within the head for example). However, if it is shown to work for a specific setup, it is probably the most efficient source material to use. Instead of pink noise, and ordinary recording containing anything could be used, with only one channel connected either to the left or right input channel.

An alternative is to use a two-channel natural recording, preferably made with a microphone which preserves natural binaural cues (a sphere microphone for example), containing sounds at the sides. These are often a bit more cumbersome to use for tuning than the pink noise, but is a good alternative."

8 Conclusions

The results obtained at the end of this project are promising. Actually, if we refer to the software specifications stated in section 4 of this document, two out of the five functions are already implemented:

- ***F2 :*** *Convolve filters with live source in real-time*
- ***F3 :*** *Display processed signals on a sound reproduction system in real-time*

The work for the function ***F1:*** *Compute cross-talk cancellation filters in real-time* is already partly done with the new Matlab script for the computation of cross-talk cancellation filters. A C++ implementation should be based on the model of this Matlab script. Adding the functionality ***F4 :*** *Update filters' coefficients as a function of listener's position in real-time* shouldn't be too difficult either once ***F1*** is done, as long as the way the application will communicate with the head-tracking listener is clearly defined.

Finally, a big part of the work still to be done is about building an easy-to-use graphical user interface that makes it easy and friendly to use the application.

References

[1] B. Gardner and K. Martin, **HRTF measurements of a KEMAR dummy-head microphone**, *MIT Media Lab Perceptual Computing*, Technical Report No. 280, 1994.

Link: *http://sound.media.mit.edu/KEMAR.html*

[2] P. Mannerheim, M. Park, T. Papadopoulos and P.A. Nelson, **The measurement of a database of head related transfer functions**, *ISVR*, Contract Report No. 04/07, November 2004.

[3] O. Kirkeby, P.A. Nelson, H. Hamada, and F. Orduna-Bustamante, **Fast deconvolution of multichannel systems using regularization**, *IEEE Trans. Speech Audio Processing*, Vol. 6, pp. 189-194, 1998. Or *ISVR* Technical Report No. 255, April 1996.

[4] O. Kirkeby, P.A. Nelson, P. Rubak and A. Farina, **Design of cross-talk cancellation networks by using fast deconvolution**, *presented at the 106th AES Convention*, Munich, 8-11 may 1999.

Link: *http://pcfarina.eng.unipr.it/public/papers/128-AES99.PDF*

[5] O. Kirkeby, P.A. Nelson, H. Hamada, **The "stereo dipole" - a virtual source imaging system using two closely spaced loudspeakers**, *J. Audio Eng. Soc.*, v 46, n 5, p 387-95, May 1998.

[6] **Website of the Virtual Acoustics Project**, *ISVR*.

Link: *http://www.isvr.soton.ac.uk/FDAG/VAP/index.htm*

[7] T. Takeuchi and P.A. Nelson, **Optimal source distribution for virtual acoustic imaging**, *presented at the 110th AES Convention*, Amsterdam, The Netherlands, May 12-15, 2001. Or *ISVR* Technical Report No. 288, February 2000.

Or

T. Takeuchi and P.A. Nelson, **Optimal source distribution for binaural synthesis over loudspeakers**, *J. Acoust. Soc. Am.* 112, 2786-2797, 2002.

[8] Lake Technology Limited. **Huron20 Digital Audio Convolution Workstation**. Documentation can be found on the CD-ROM in *Documentation\Technical*.

[9] W. G. Gardner, **Efficient convolution without input-output delay**, *J.AES* vol. 43, n. 3, 1995 March, pp. 127-136.

[10] O. Kirkeby, D. Tavan, **Matlab script for the fast deconvolution method using regularization**, *ISVR*, 2005.

Script and Documentation can be found on the CD-ROM respectively in *Source\Matlab* and *Documentation\Matlab*.

[11] A. Torger. **BruteFIR. A real-time convolver working under Linux.**
Link: *http://www.ludd.luth.se/~torger/brutefir.html*

[12] A. Torger and A. Farina, **Real-time partitioned convolution for ambiophonics surround sound,** *proceedings of the 2001 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk Mountain House New Paltz, New York October 21-24, 2001.
Link: *http://www.acoustics.net/objects/pdf/article_farina04.pdf*

[13] A. Torger. **AlmusVCU.**
Link: *http://www.ludd.luth.se/~torger/almusvcu.html*

[14] RME. **HSDP9632 Soundcard.**
Link: *http://www.rme-audio.com/english/hdsp/hdsp9632.htm*
Documentation can be found on the CD-ROM in *Documentation\Technical.*

[15] RME. **ADI-2 Converter.**
Link: *http://www.rme-audio.com/english/adi/adi2.htm*
Documentation can be found on the CD-ROM in *Documentation\Technical.*

[16] A. Farina. **AURORA plug-ins for Adobe Audition.**
Link to the main page: *http://pcfarina.eng.unipr.it/aurora/home.htm*
Link to the page related to the Real-time convolution plug-in:
http://pcfarina.eng.unipr.it/aurora/waveconv.htm

[17] **Adobe's website.**
Link: *http://www.adobe.com/*

[18] C. Knufinke. **SIR VST plug-in.**
Link: *http://www.knufinke.de/sir/*

[19] Plogue Art et Technologie, Inc. **Plogue Bidule.**
Link: *http://www.plogue.com/*

[20] A. Farina, R. Glasgal, E. Armelloni and A. Torger, **Ambiophonic principles for the recording and reproduction of surround sound for music,** *presented at the 19th AES Conference on Surround Sound, Techniques, Technology and Perception*, Schloss Elmau, Germany, 21-24 June 2001.
http://www.ambiophonics.org/Ambiophonic_Principles/Farina_et_al_1.htm

[21] I. Sánchez and J. Bescós, **Software modules for HRTF based dynamic spatialisation,** *Grupo de Tratamiento de imágenes, Universidad Politécnica de Madrid.*
Script and Documentation can be found on the CD-ROM respectively in *Source\C++\Bescos* and *Documentation\Class Libraries\Bescos.*

[22] Stephen Pope. **CREATE Signal Library.** *Centre for Research in Electronic Art Technology (CREATE) - University of California, Santa Barbara (UCSB).*
Link: *http://www.create.ucsb.edu/CSL/*

[23] R. Bencina and P. Burk. **PortAudio Library.**
Link: *http://www.portaudio.com/*

[24] G. P. Scavone. **RtAudio Library.** *Centre for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University.*
Link: *http://www-ccrma.stanford.edu/~gary/rtaudio/*

[25] Victor Lazzarini. **SndObj Library.** *Music Technology Laboratory - National University of Ireland, Maynooth (NUIM).*
Link: *http://www.nuim.ie/academic/music/musictec/SndObj/main.html*

[26] P. R. Cook and G. P. Scavone. **STK Library.** *Centre for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University.*
Link: *http://ccrma.stanford.edu/software/stk/*

Appendix

Here are described the characteristics of the files generated by the Matlab script for cross-talk filters computation. These files contain cross-talk cancellation filters designed for *StereoDipole* playback. The results shown below are for filters generated with the ISVR/SAMSUNG database, using the *no_open* data source.

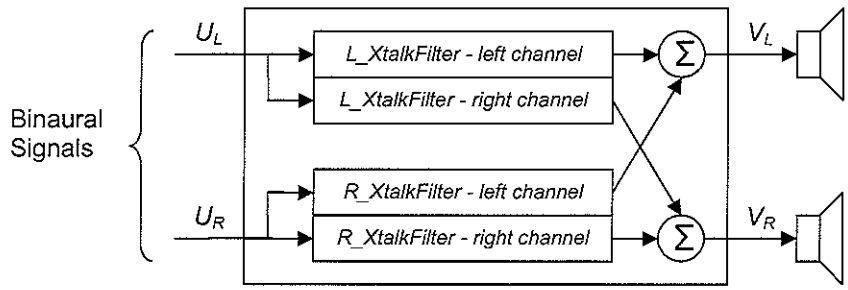

Figure 1: Network of cross-talk cancellation filters

1) Sources and receivers positions

SOURCE_ARRAY = **[355 5]**; %- The positions of the sources in degrees

RECEIVER_ARRAY = **[270 90]**; %- The positions of the ears/microphones in deg.

ELEVATION = **0**; %- The elevation is set at 0 deg.

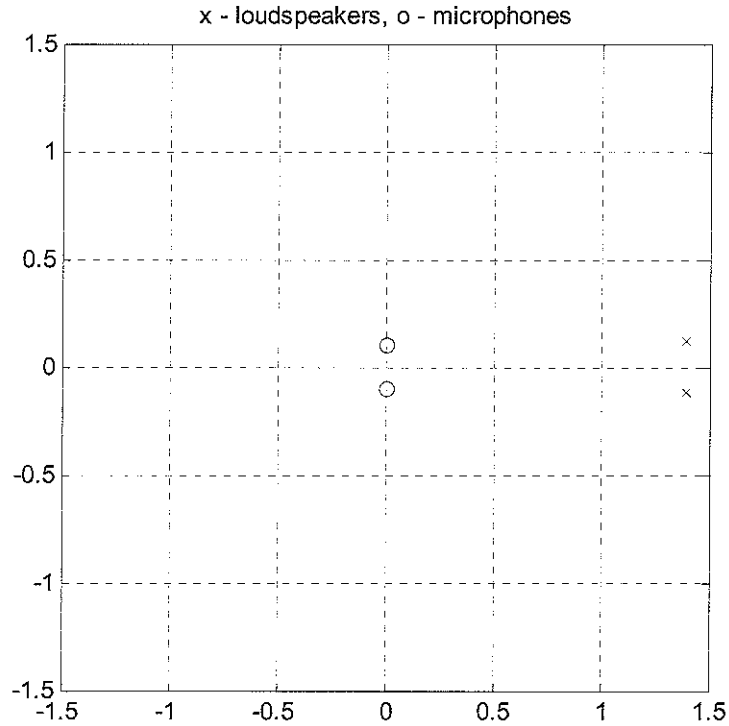

Figure 2: Source and receiver positions

2) Parameters for cross-talk cancellation filters computation

LEAK = **0.0001**; %- The regularisation parameter

N_COEF = **32000**; %- The number of coefficients in each filter

3) Parameters for data display

SAMPLE_RATE = **48000**; %- The sample rate at which HRIRs have been recorded

N_FREQ = **256**; %- The number of frequency values used for logarithmic plot

4) Parameters for output WAV files

S_RATE = **44100**; %- Sample rate

N_BIT = **16**; %- Bit depth

5) Filter performance plot

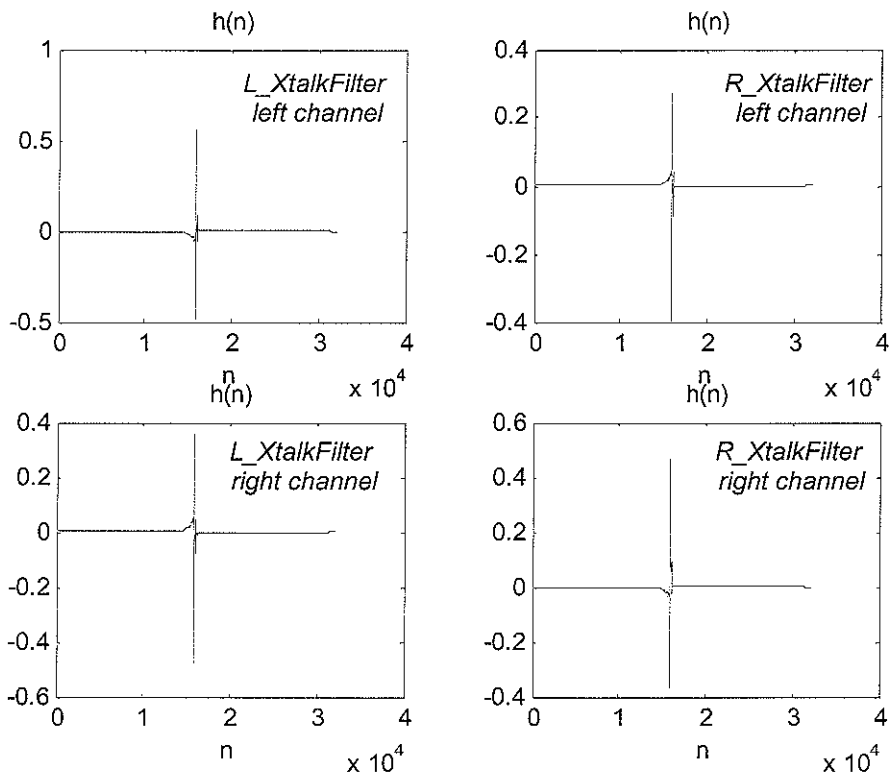

Figure 3: Impulse response

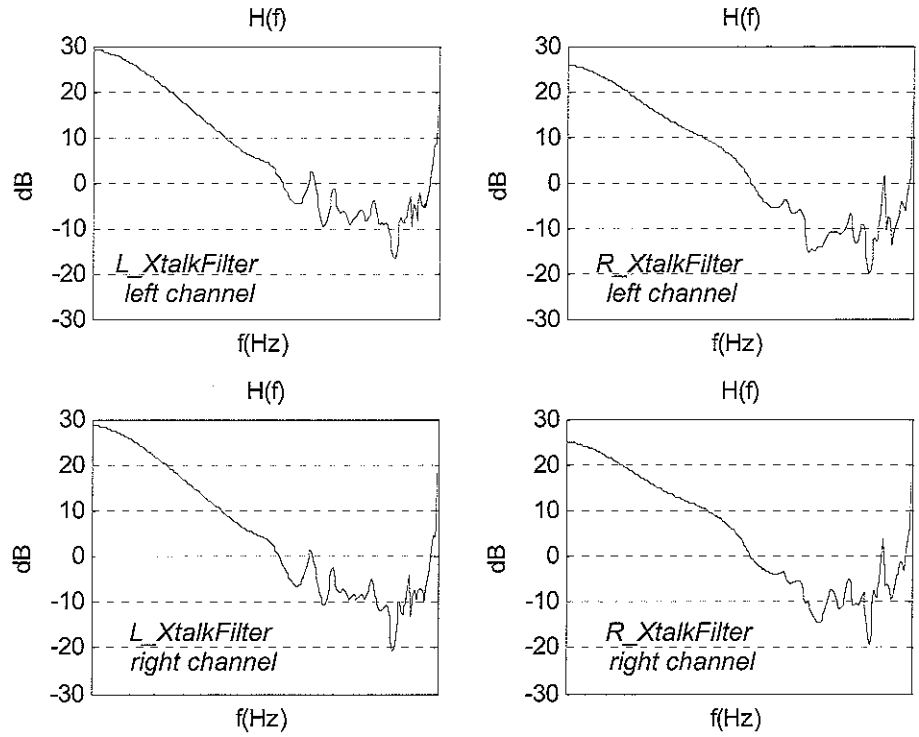

Figure 4: Frequency response

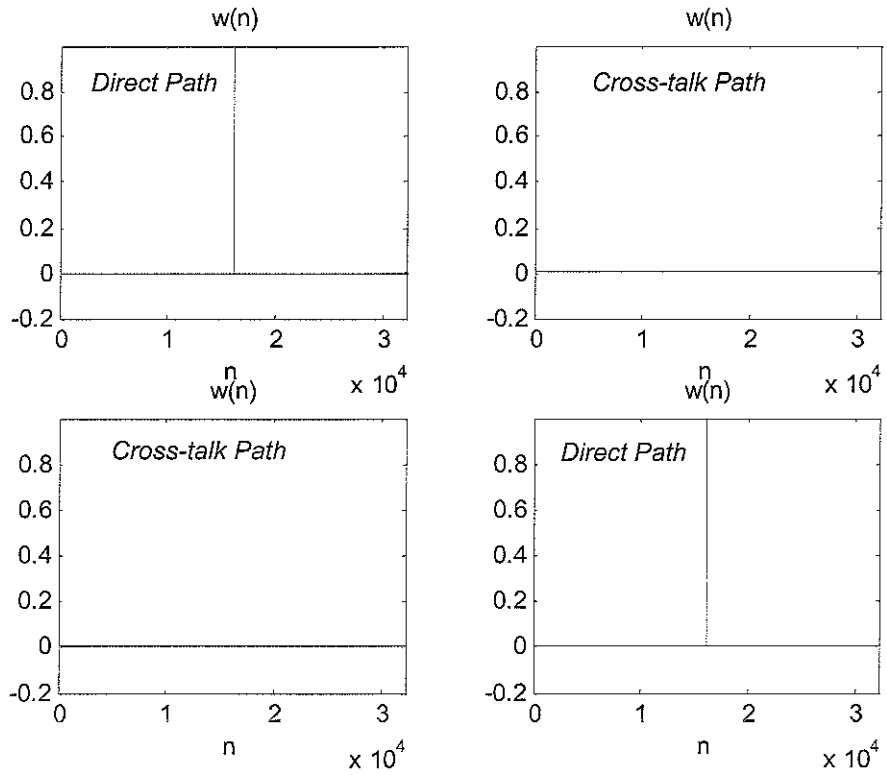

Figure 5: Cross-talk cancellation performance (time domain)

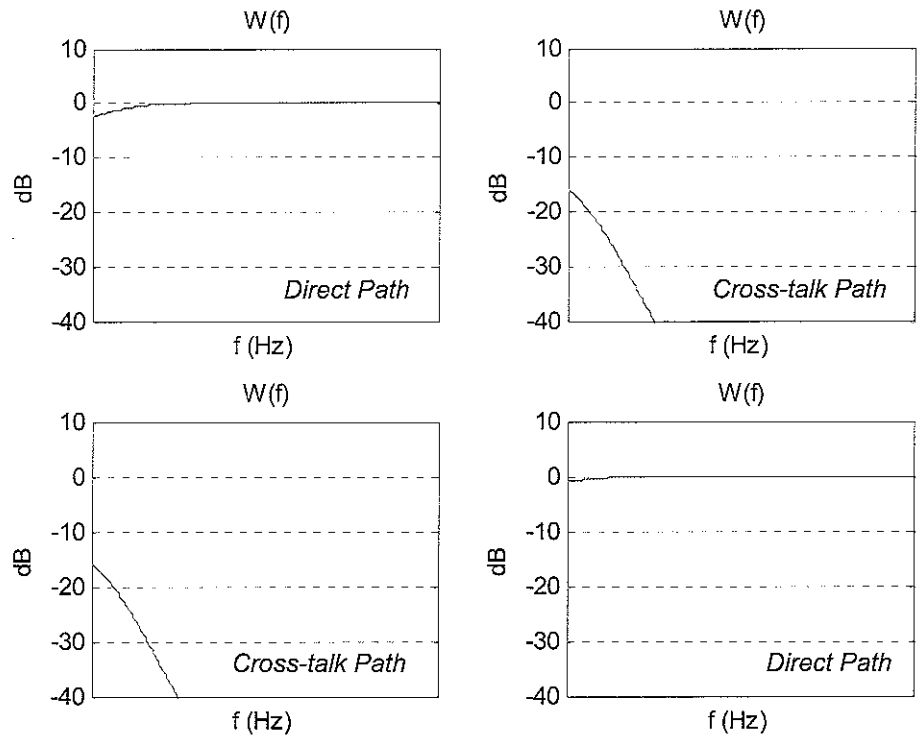

Figure 6: Cross-talk cancellation performance (frequency domain)

Fig.5 and Fig.6 show that cross-talk cancellation with these filters is efficient since the cross-talk path is almost completely cancelled whilst the direct path remains almost unaltered.