An Open Development Platform for auditory real-time signal processing

Seon Man Kim , Stefan Bleeck

PII: S0167-6393(17)30152-8
DOI: 10.1016/j.specom.2017.12.003
Reference: SPECOM 2510

To appear in: *Speech Communication*

Received date: 26 April 2017
Revised date: 1 November 2017
Accepted date: 4 December 2017

An Open Development Platform for auditory real-time signal processing

Seon Man Kim and Stefan Bleeck
Institute of Sound and Vibration Research,
University of Southampton,
Southampton, SO17 1BJ, United Kingdom

Abstract

Algorithms that are used in real-time hearing devices, such as hearing aids or cochlear implants, need to work with a short time latency in the order of 10ms and must be efficient enough to run on battery power. Furthermore, in order to evaluate ecological validity such algorithms often need to be validated in the development stage using real-time processing of environmental sounds. However, researchers often lack the ability to access to the technology and they use offline schemes instead. In order to allow more and less specialised researchers access to the necessary technology and make it as easy and straightforward to develop real-time algorithm, we have developed an auditory real-time processing algorithm platform ("Open Development Platform", ODP). The ODP is designed to help the development from existing offline algorithms to novel real-time algorithms by introducing a clear set of rules for development and evaluation. ODP has a software component that works in MATLAB/Simulink and a hardware component that works on the Speedgoat® Target machine. We also explain several algorithms from the available library that have low latency and low power consumption: filter bank, envelop power estimator, frequency shifter, and adaptive feedback canceller. As example of a complete project, we demonstrate here, how they can be combined to a fully functional hearing aid for research purposes.

Introduction

In auditory signal processing research, we are often interested in algorithms that modify sounds to provide benefits in specific situations, for example to increase speech intelligibility in noise. Often it is interesting and sufficient to process sounds offline and play them back in an experiment to participants at a later stage. This approach has the advantage of full control over all parameters including background noise. However, in order to know if the algorithm works in real-life, we want to know about interactions between algorithm, environment and participant in real-life situations; only then the algorithm can be fully evaluated for its usefulness. In fact, most algorithms that have shown benefit in laboratory situations were either never further evaluated in real-time environments or failed to live up to their promise if they have [1], [2]. Real-time processing of environmental sounds means giving up control of most environment parameters, but gaining highest possible 'ecological validity', that is to test the participant in unpredictable situations as they would face in real life. 'Ecological validity' is defined as the extent to which the results of a study can be generalized to real-world settings. This approach is important for example for hearing aid noise reduction algorithms, where the wearers outside of the lab encounter unpredictable situations. However, in order to develop algorithms that have potential to be implemented in a future generation of auditory devices, they must be tested in many possible situations, but testing real-time algorithms in real-life situations is difficult or impossible for many researchers. The evaluation of real-world benefits of signal processing algorithms has been investigated and described by several research groups [1], [2].

Research and development (RND) in audio processing algorithms commonly consists of three sequential development stages that can be described as: 1) offline development, 2) real-time development, and finally 3) system integration for the fully developed algorithm [3]. Generally, a new algorithm is developed first offline, and functional performance is evaluated by using an offline development database (e.g. speech databases [4], [5]). Typically, at this stage it is sufficient to process experimental sounds offline and it is imperative to have full control of the environment and all parameters. Usually, only when the algorithms' benefit is demonstrated under these offline laboratory situations the next development stage is justifiable. In the second stage, the algorithm is thoroughly evaluated under real-time conditions prior to hardware implementation. This often requires tuneable parameters for controlled manipulations of the considered signal processing. In this stage the algorithm is evaluated and optimised in a closed loop cycle of offline and real-time stages in real-time. Finally, in the third stage, the algorithm is integrated and realized as a prototype. The system integration is usually outside the scope of the researchers in academic institutions and will sometimes happen in industry.

Both offline and real-time stages require specific development platforms. The platform design should specify the hard- and software as well as the rules that describe how they fit together [6]–[8]. In existing systems, a high-level programming languages such as MATLAB, Simulink, and C/C++ is used on a personal computer (PC) platform for the offline development stage. Despite their relative simple configuration, with today's computer power, these platforms are often enough to develop and evaluate even complex algorithms offline.

However, existing *real-time* development platforms are much more complex and require additional training for the researcher. Real-time platforms typically consist of signal input/output (I/O) devices, an analogue-to-digital converter (ADC), a central processing unit (CPU), random-access memory (RAM), a digital-to-analogue converter (DAC) and so on. Furthermore, to deal with this extensive hardware, they usually employ a specialised suitable PC device (e.g., a performance real-time target machine) that is fast enough to do the processing for the real-time calculations [8], [9]. Furthermore, to realise real-time processing on existing platforms, specialized skills are required, such as fixed point implementation, complexity optimization and system embedding – skills that are often outside the scope of academic researchers [10].

In order to enable more researchers to utilise the benefits of real-time algorithm development, it would be beneficial to close the gap between online and offline stage and to make it easier to use offline algorithms in real-time directly.

We have developed such a platform, the 'Open Development Platform (ODP)' and introduce it in this paper.

Our platform is operated and controlled using the high-level programming language MATLAB, which is already often used as the language of choice for offline algorithm development, and implemented for real-time processing with the available tools of Simulink. Algorithms written in MATLAB can be automatically compiled and transferred to a performance real-time target machine (http://www.speedgoat.ch) using Simulink Real-Time Target (formerly known as xPC Target) and Simulink Coder™ (formerly Real-Time Workshop®) from MathWorks. The used target machine in this paper is commercially available ('the Speedgoat® target machine') that we used for the fully tested implementation. However, the ODP is explicitly not restricted to this hardware, all algorithms should work also on a wide variety of target machines supported by MathWorks, (supported targets include for example Android OS, Raspberry Pi (http://www.raspberrypi.org), Arduino (http://www.arduino.cc), National Instruments, Texas Instruments, and so on. For a full updated list, see https://kr.mathworks.com/programs/products/simulink-real-time/supported/hardware-drivers.html).

The ODP is designed to take out most of the pain of development for the researcher by simplifying and standardising the notoriously difficult steps in the design processes by compiling and transferring offline algorithms into usable real-time algorithms – ready to be used in experiments with high ecological validity [9], [11].

The novel approach that we report in this paper simplifies access to complex technology. We combine existing hardware with novel software; we present a simple and useful manual and we offer a freely accessible database of algorithms. All of this together will it make easier for other researchers to perform more powerful experiments.

Limitations

Note that for a true real-world evaluation, the size of the real-time platform must be small and light enough to be portable. Furthermore, it must run on battery for sufficient amount of time. It might thus be beneficial to have additional small scale real-time portable platforms besides laboratory test platforms that are often heavy and large.

Apart from the soft- and hardware issues on the real-time platform, using auditory algorithms in real-time processing is not straightforward due to a number of issues, foremost latency: minimizing processing time on the real-time platform is often crucial for auditory devices. It is important that original surrounding sounds and the sounds produced by the algorithm are entering the ear roughly at the same time; however, processing inevitably causes some time delay, which generates annoying comb filter effects even when the processing time is very short. Thus, algorithm processing time delay should be considered already at the offline RND stage in conjunction with real-time processing. Typically, a 10 ms time latency is allowed for auditory devices such as hearing aids [12]–[15]. A delay > 20 ms is generally considered to be annoying. This time constraint has been a barrier for the development of powerful algorithm especially in cases where the signal is split into several frequency bands, for example in dynamic range compression (DRC) or automatic gain control (AGC) [8], [16], [17]. This has led to development of low delay filter banks [13], [18], but it is still often an issue. Furthermore, algorithms should aim for low power consumption due to limitations of the battery in a portable platform.

For the ODP, we have developed an implementation of a low latency and low power filter bank that is described below and that can be freely used. In addition, especially in the case of hearing aid development, direct acoustic feedback must be considered. In auditory processing systems, such as a hearing aids, the microphones and ‘receivers’ (loudspeakers) are located within millimetres and sometimes within the cavity of the ear canal. This leads to feedback that can range from annoying to catastrophic [19]. In the ODP we have developed a feedback canceller based on frequency shifting that is described below and that can be freely used.

The number of channels in ODP is only restricted by the used hardware, not the software. It is entirely possible to write algorithms that use multiple microphone inputs to generate for example directional microphone patterns or adaptive directionality. Inputs can be monaural or binaural and can be combined in the software for multichannel processing.

Comparison to previous platforms

Several other hard- and software solutions have been presented in the past that allow real-time development and implementation of signal processing algorithms [3], [7], [8], [10], [20]–[24]. However, they are all restricted by the used soft- or hardware: no existing system combines the advantages to provide a freely accessible library of auditory algorithms ready to be used on a commercially available target. With the release of ODP, we aim to facilitate the adoption of real-time experimentation by other researcher by providing a systematic explanation of a systematic and documented set of rules and guidelines about:

- how to implement auditory algorithms step by step from offline stage to real-time stage,
- how to combine the algorithms on the real-time platforms,
- how to compile and transfer its executable file into the hardware device.

The detailed set of rules that we present here should also facilitate networked activities with common standards that ensure compatibility among different algorithms by offering tools that govern algorithm sharing and policies that constrain user implementation style for real-time processing and sharing of network participants.

We aim to open up the field of real-time signal processing to a wider audience who have access to the soft- and hardware, but lack the specific skills that are necessary to program real-time applications. Our principal philosophy is to enable and to share. The complete ODP (including a library of many more algorithms) can be downloaded for free from the repository at www.soundsoftware.ac.uk.

The remainder of this paper is organized as follows. Section 2 reviews some technical background knowledge needed for understanding the operation of the ODP. Section 3 provides the detailed description of some basic auditory algorithms that are part of the ODP library such as filter bank, feedback canceller, frequency shifter and power envelope estimator. Section 4 explain the ODP algorithm library and section 5 then implements a hearing aid system with a noise reduction algorithm as an example of using the ODP library algorithms.

ODP technical background

In this section, we describe the technical concepts necessary to understand how to operate the platform. All described algorithms on the ODP are written in MATLAB and implemented into real-time processing functional blocks in Simulink. The algorithms are automatically compiled and transferred to a target machine using Simulink Real-Time (formerly known as xPC Target). During the operation of the target machine, the transferred algorithms on the target machine are controlled in real-time by a host PC through the Simulink Real-Time kernel.

2.1 Definition of real-time processing

A computer can be very fast and process data as fast or faster as real-time, but that is not the same as real-time processing. Standard PCs today are very fast, but the operating systems (Windows, MacOS X, etc.) are not optimised or even suitable for real-time processing because too many components (e.g. graphical user interfaces, Ethernet or peripherals) require the processors attention via interrupts at unpredictable times and for unpredictable durations. It is thus difficult to use standard PC operating systems for suitable real-time experiments and therefore usually specialised kernels are used for this task.

Real-time processing suitable for auditory tasks is event driven and requires three components: an input buffer, an interrupt service routine (ISR) and an output buffer. The principle that we employ for the ODP is illustrated in Figure 1: first, an input buffer (bottom row) is filled with real-time data. Every ‘sample time’ (not to be confused with the ‘sample rate’) an ISR is called (middle row) and the current data in the buffer is processed. Please note that the phrase “sample time” represents the time interval in which the processing algorithms should finish processing the algorithms and should produce their outputs at the end of the time interval. The ISR must finish within one sample time and in that time, fill the output buffer (top row). A parent control framework process ensures that the input buffer is filled (and the output buffer is played) at every sample time, whatever it contains. This framework illustrates some fundamental restrictions of real-time processing as employed in the ODP, primarily that the overall output delay is fixed to be twice the length of the buffer. The maximum processing time is also determined by the sample time. If the data is not processed in this time, a ‘drop out’ occurs and the output buffer is not filled, resulting in audible distortions. There is a small additionally time necessary for communication and overheads, but this is usually neglectable compared to the processing time.

Note that because of the real-time nature of the processing, every algorithm has to be causal, which means that it can only work on data from the past. This seems obvious, but requires a different programming style from what researchers often do: to read in long audio files and process them in one go. Publicly available popular toolboxes for Auditory Signal Processing (like the ‘auditory toolbox’ or the ‘auditory modelling toolbox’) for example work entirely on the principle of processing whole files and can therefore not be used directly for real-time processing. Programmers therefore have to adjust their programming technique towards working with small chunks of data (‘frame based’) and the use of memory persistent variables.

2.2 The Speedgoat® target machine

The ODP in our lab was implemented on the real-time target machine from Speedgoat® GmbH (http://www.speedgoat.ch) shown in Figure 2. The target machine was used in the H2020 EU ICanHear project (http://www.icanhear.eu) and as a proof of concept, all network partners developed software that runs on this target using the ODP. Transparency and openness was required for compatibility and the rule set defined by the OPD was validated. The advantages of the Speedgoat® target machine include multi-channel I/O with synchronization, short latency, high power CPU, large internal memory and easy-to-monitoring & logging of data. On the other hand, this system is expensive compared to other real-time solutions and it has a high-power consumption making it difficult to run off the grid and is also too large and heavy to be truly portable.

2.3 The Simulink real-time workshop/Simulink coder

MATLAB is a widely-used programing language, Simulink is less widely used, but offers advantage that make it our choice, specifically the Simulink Coder™ (formerly Real-Time Workshop®), that allows compilation of MATLAB functions into standalone executables that run in real-time (processor permitting), and also compilation and deployment of functions onto 'targets' that are specialised computers or processors. At this time, possible targets include also Texas instruments DSP, iPhone and the Raspberry Pi. We chose to develop software on this basis, because MathWorks is offering constant support and updates for hardware and drivers and have a policy of updating drivers periodically.
MATLAB/Simulink take away the necessity to program and even to understand the compilation of code into C and linking it into executable files. The user only needs a knowledge of MATLAB and a basic understanding of Simulink in order to program complex systems on the ODP.

2.4 'Simulink MATLAB Function'

Simulink, developed by MathWorks, is a graphical environment for programming and simulating signal processing algorithms. The programming is conducted by diagramming graphical algorithm object blocks from a set of block libraries. In addition, there are several ways of producing viable Simulink code by custom function blocks such as 'Interpreted MATLAB Functions', 'S-Functions', 'Level-2 MATLAB Functions', 'MATLAB Systems', and 'MATLAB Functions'. Those programming methods will all work within the framework of the ODP and among them we recommend the use of MATLAB functions in order to directly employ MATLAB script codes. This recommendation is based on our experiences that the user is familiar with MATLAB script coding, but not necessarily with Simulink-based graphical object coding, and using graphical blocks can often result in very complex and unruly models that are prone to errors and need a lot of experience and training to use optimally. Additionally, and maybe surprisingly, MATLAB Functions are not necessarily slower than other implementation options after compilation. In fact, many MATLAB function-based algorithms are faster than native Simulink blocks because the inbuilt compiler produces highly optimised code. A complete description of the speed test that led to this conclusion is beyond the scope of this paper and can be downloaded from the repository on soundsoftware.ac.uk.

The other main reason for our recommendation to use MATLAB functions is that because of the available sophisticated tools (e.g. debugging) they allow the fastest overall development time from an original MATLAB file to a running real-time executable. Furthermore, MATLAB function allow first using offline MATLAB script codes for a real-time simulation before the final real-time implementation and finally, they make the program easier to understand, debug and maintain.

2.5 Real-time windows target vs. real-time xPC target

The “Real-Time Windows Target (RTWT)” that is produced by Simulink compilation is needed for the real-time processing of a Simulink model on a Windows operating system (OS)-based computer. It shares hardware resources with Windows. It does not matter how many other applications are running on the computer. Those applications may slow down when the real-time kernel takes large amounts of CPU time, but the performance of the kernel itself is unaffected by the applications. This typically means a lower cost solution than using a specialised Target as no additional computer hardware is necessary. However, the RTWT does have disadvantages compared with a specialised target machine, especially as it is very difficult to achieve low enough latencies with standard PC sound cards.

The Simulink Coder is also needed to achieve real-time processing of a Simulink model on independent “target” computers. These target computers are booted with a dedicated real-time kernel and achieve low latency real-time performance as the computer hardware is completely dedicated to real-time tasking. While RTWT can achieve sample rates up to 5 kHz, Simulink Coder can achieve sample rates approaching 50 KHz, depending on processor performance level, model size, and I/O complexity.

2.6 ODP program development chain

Even with today’s technology, it is not straightforward to produce algorithms that work in real-time. With our set of rules suggested here, we aim to make it as simple as possible for people with no background in real-time programming to achieve success quickly. From our experience, almost all programmers start from analysing offline audio files and almost everyone gets stuck there for the lack of help and the complexity of the task ahead. The full description of the processes and tutorial is beyond the scope of this paper, but a full documentation and extensive tutorial material can be downloaded from the repository. At the moment those documents include:

- An extended guideline (90 pages) about the ODP that explains the background and the principles of programming as well as a tutorial how to get started;
- A technical report how to install the Speedgoat® target machine and get it to work with the ODP;
- A document outlining the background of a comparison for speed test evaluation that is behind the motivation to use MATLAB function block programming rather than native Simulink code.

When programming real-time applications, we recommend using 4 distinct development stages outlined below and in Figure 3. Every stage is useful in itself and development can stop at each stage if the goals have been achieved:

1. (**Offline**) First, the algorithm is shown to work in principle in a non-frame offline mode. Reading audio files, processing them and sending them back to output audio files where it can be analysed objectively or be listened to via headphones or loudspeakers. A successful evaluated algorithm demonstrates a first proof of concept, but it could turn out to be not implementable into real-time (for example because it is not causal).
2. (**Offline: frame-based**) The first novel step for most programmers will be re-programming the algorithm frame based. That means instead of reading the whole audio file, reading small chunks into a buffer, processing the buffer and sending the result to an output buffer. This step holds the greatest conceptual challenge for any programmer, as it requires strict separation of different scopes of variables, strict and watertight definition of global parameters and an understanding of persistent and local variables and finally concepts of overlapping buffering (for overlap and add). So far, speed of processing is not a consideration. This step can be fully implemented in MATLAB Functions without the need of Simulink. A successful evaluated algorithm demonstrates a significant step towards a real-time system. From here, only technical problems (like processing speed) might hamper further progress.
3. (**Online: on host PC**) The third step is conceptually actually a very small step, although it signifies the main step from offline to online processing and is the biggest step in terms of programming behind the scenes done by MATLAB. Luckily the user doesn't notice very much of this. The file input and output is replaced by microphones, and the interpreted MATLAB code is replaced by the Simulink environment. Now speed becomes an issue, as the processing of the buffer must be faster than the sample time.
4. (**Online: on target machine**) This step does not add anything conceptually. In theory, if the algorithm runs in step 3, it should run on the target machine. This is for the Simulink compiler to work out. In practice, however, this step, at least for new programmers, holds considerable challenge, all to do with the (sometimes somewhat fluid) interface between Simulink and the real-time hardware.

3. ODP basic auditory algorithms

In this section, we describe some examples of real-time audio signal processing algorithms in detail that are already implemented in the ODP, are part of the library and can be used and combined freely. They include a low-latency filter bank, a power envelope estimator, a frequency shifter and an adaptive feedback canceller. These algorithms will be combined in chapter 5 for a full implementation of a hearing aid.

3.1 Polyphase DFT filter banks

A digital filter bank is an essential algorithm that is frequently at the beginning of the auditory signal processing system. The specific requirements for filter banks for auditory devices, particularly hearing aids, have been well outlined in [13]: first, it is desirable to have uniformly spaced and narrow frequency bands with little overlap between the bands, with stop band attenuation of at least 60 dB, preferably higher. Secondly, the overall algorithm processing delay should not exceed 10ms to avoid adverse effect on the subjective listening experience. Furthermore, low computation complexity is desirable due to the restricted capacity of processor and battery to be realized to real portable devices. Thus, choice of the filter bank is an important consideration when planning such a system in terms of signal quality, computational complexity, and signal delay. Efficient implementations of filter banks which are feasible can be classified into four categories [13], [18]: cascaded infinite impulse response (IIR) filter banks, short time Fourier transform (STFT)/windowed overlapped fast Fourier transform (FFT), uniform polyphase filter banks, and warped polyphase filter banks. The method of choice is most often the uniform polyphase filter banks. These filter banks are based on the uniform discrete Fourier transform (DFT) with weighted overlap-add STFT.

We suggest to use the uniform polyphase filter banks for the ODP because of its efficiency, expandability into non-uniform filter banks, low latency and perfect reconstruction characteristics. Additionally, it has the advantage of being implemented as an overlap-and-add (OLA) STFT, which make it easy to reuse the already calculated short-term spectral amplitude (STSA) for noise reduction algorithms as most noise reduction algorithms are based on the STFT [25].

We start by describing the filter bank implementation with the prototype low pass filter design. Figure 4 shows for two examples of 96 window sequences by (a) the standard Kaiser Window and (b) the approach [13] with their frequency domain magnitude shape in the upper and lower panels, respectively. As shown in the figure, Kaiser Window provides more attenuation than the approach [13]; however it also has more ripples and wider bandwidth. Even though we suggest using the approach [13] in our ODP, we hereafter described ODP examples by using the standard Kaiser Window, because the Kaiser window is also commonly used for the prototype low pass filter design [18] and can be easily obtained by the the MATLAB built in function “kaiser(128,15)”.

In order to implement an *M*-bands filter bank, a downsample factor *R* (or frame shift length via STSA approach), FFT size *K* and prototype filter length *L* need to be defined. Once the number of filter bank bands is fixed as *M,* the down sample factor *R* is determined by oversampled condition as *R=M/2* or the perfect reconstruction filter bank under spectral weighting conditions. The FFT size *K* is fixed as *K=4M* or *K=8R.* Then, the prototype filter length *L* is determined to be *K*; however, lower length is desirable to reduce the time delay by using the zero-padding method. Note, that the prototype filter length *L* also affects the filtering performances such as stop band attenuation and main lobe widths. The above procedure can be summarized as follows:

- Step1: Set the number of frequency channels M
- Step2: Determine downsample factor $R \leftarrow M/2$
- Step3: set FFT size $K \leftarrow 8R$
- Step4. Choose the prototype filter length $L \leq K$

For instance, if a 32 bands filter bank is implemented, i.e., $M = 32,$ then $R = 16$ and $K = 128$. Also, $L = 96.$ This example filter bank first causes twice the buffering delay of $R = 16$ samples (refer to Figure 1). That is 2 ms with a 16-kHz sampling rate. Then, the OLA STFT also causes the time delay 5ms according to the difference 80 samples. ($L-R$). Thus, the final overall time delay is estimated as 7 ms, which meets the requirements of maximally 10 ms time delay.

In order to generate high quality output sounds, the filter bank must be perfectly reversible. Thus, the filter bank consists of an analysis and a synthesis part. The first is for decomposing an input time domain signal into several frequency band signals and the latter is for restoring the output signal from frequency bands signals. The following sub-sections describe those analysis and synthesis filter banks in more detail.

3.2 Analysis filter bank

For the analysis filter bank, the input time discrete signal $x(n)$ is buffered to form the ℓth frame signal

$$\mathbf{x}_\ell = [x(\ell R), x(\ell R+1), x(\ell R+2), \cdots, x(\ell R+L-1)]^T, \quad (1)$$

where T denotes the transpose operator. $x(n)$ describes the discrete input time signal. The time reversed window signal $\hat{\mathbf{x}}_\ell$ is obtained by applying the window function $\mathbf{w}=[w(0), w(1), \cdots, w(L-1)]^T$ to $\mathbf{x}_\ell$. That is,

$$\hat{\mathbf{x}}_\ell = \begin{bmatrix} w(L-1)x(\ell R) \\ w(L-2)x(\ell R+1) \\ \vdots \\ w(0)x(\ell R+L-1) \end{bmatrix}. \quad (2)$$

The $\hat{\mathbf{x}}_\ell$ is then zero-padded to meet K FFT length as

$$\hat{\mathbf{x}}_\ell^{\vec{\mathbf{0}}} = \begin{bmatrix} \hat{\mathbf{x}}_\ell \\ \vec{\mathbf{0}}_{(K-L)\times 1} \end{bmatrix}, \quad (3)$$

where $\vec{\mathbf{0}}_{(K-L)\times 1}$ is a $(K-L)$ size zero vector. The zero padded version of $\hat{\mathbf{x}}_\ell$, $\hat{\mathbf{x}}_\ell^{\vec{\mathbf{0}}}$, is transformed into the complex valued spectral component $X_k(\ell)$ at the kth frequency bin $(k=0,1,\cdots K-1)$ and the ℓth frame by a FFT. If the conditions of $R=M/2$, $K=8R$, and $L\le K$ are satisfied, then the R down sampled-signal at the mth frequency band $x_m(n_{\downarrow R})$ can be obtained by

$$x_m(n_{\downarrow R}) = X_{2m}(\ell)e^{-j2\pi\frac{2m}{K}(K-L)}, \quad (4)$$

where $n_{\downarrow R}(=0, R, 2R, \cdots, \ell R, \cdots)$ denotes sample index and $e^{-j2\pi(2m/K)(K-L)}$ is for compensating the delay by zero padding (1 in the case of $K=L$). Since $x_m(n_{\downarrow R})$ is a complex value, the real-time domain signal can be obtained by taking real value i.e., $real(x_m(n_{\downarrow R}))$. Example analysis filter bank output signals for the ODP implementation are shown in Figure 5.

3.3 Low latency synthesis filter bank

The signal $x_m(n_{\downarrow R})$ is usually further processed by algorithms such as equalizer, noise reduction, DRC, and so on. After this, the modified version of $x_m(n_{\downarrow R})$, $\hat{x}_m(n_{\downarrow R})$, will be synthesized over all bands $(m=0,1,\cdots,M-1)$ to resynthesize the output signal $y(n)$. To this end, the zero padding delay compensation factor in (4) to $\hat{x}_m(n_{\downarrow R})$ is first reversed by taking

$$Y_m(\ell) = \hat{x}_m(n_{\downarrow R}) e^{j2\pi\frac{2m}{K}(K-L)}, \tag{5}$$

which then creates a $\mathbf{Y}_\ell$ as follow:

$$\begin{aligned}\mathbf{Y}_\ell = [&Y_0(\ell), Y_1(\ell), \cdots, Y_{M-1}(\ell), \\ &0, Y^*_{M-1}(\ell), Y^*_{M-2}(\ell), \cdots, Y^*_2(\ell)]^T,\end{aligned} \tag{6}$$

where $*$ denotes complex conjugate. By using a size $K/2$ (or $2M$) inverse FFT (IFFT), $\mathbf{Y}_\ell$ is transformed into a $2M$ size time domain signal $\mathbf{y}'_\ell$. Note that $\mathbf{y}'_\ell$ is half-size spectral down-sampled value, thus spectral up-sampling is applied to $\mathbf{y}'_\ell$ in the time domain by duplicating it:

$$\mathbf{y}^{\bar{\mathbf{0}}}_\ell = \begin{bmatrix} \mathbf{y}'_\ell \\ \mathbf{y}'_\ell \end{bmatrix}. \tag{7}$$

Because $\mathbf{y}^{\bar{\mathbf{0}}}_\ell$ is the K size zero padded version by (3), only the first L samples are taken from $\mathbf{y}^{\bar{\mathbf{0}}}_\ell$, on which the window $\mathbf{W}$ is applied as

$$\hat{\mathbf{y}}_\ell = \begin{bmatrix} w(L-1)y^{\bar{0}}(\ell R) \\ w(L-2)y^{\bar{0}}(\ell R+1) \\ \vdots \\ w(0)y^{\bar{0}}(\ell R+L-1) \end{bmatrix}. \tag{8}$$

Finally, the synthesized signal $\mathbf{y}_\ell$ is obtained by the OLA of $\hat{\mathbf{y}}_\ell$. Figure 6 shows the impulse response of the transfer function of the 32-channel analysis/synthesis filter bank system based on the described polyphase DFT algorithm.

3.4 Power envelope estimation

In order to provide a smooth signal power level, the attack time (AT) and release time (RT) parameters are used for controlling the gradual change of power. Well-chosen time parameters are crucial for perception since unpleasant audible artefacts are often associated with the wrong choice of these time control parameters. Several approaches have been introduced in [17], and among them the "Smooth Peak detectors" method has been chosen for implementation here. The power envelope $P(n)$ of the time domain signal $x(n)$ is estimated as

$$P(n) = \alpha_n P(n-1) + (1-\alpha_n)\,|\,x(n)\,|, \tag{9}$$

where

$$\alpha_n = \exp(-1/(\tau_n f_s)). \tag{10}$$

The time parameter τ_n is switched between AT and RT parameters of τ^{AT} and τ^{RT}, respectively, according to the following rules;

$$\tau_n = \begin{cases} \tau^{AT} & \text{if} \quad |x(n)| \rangle P(n-1) \\ \tau^{RT} & else \end{cases} \tag{11}$$

The power $P^{dB}(n)$ then is calculated as

$$P^{dB}(n) = 20\log_{10} P(n). \tag{12}$$

The procedure of (9)–(12) can be also applied to estimate the power on the mth frequency band signal $real(x_m(\ell))$ in (4). In other words, the power envelope $P_m(\ell)$ at the mth frequency band is estimated as

$$P_m(\ell) = \alpha_{m,\ell} P_m(\ell-1) + (1-\alpha_{m,\ell}) \,|\, real(x_m(\ell)) \,|, \tag{13}$$

$$\alpha_{m,\ell} = \exp(-R/(\tau_{m,\ell} f_s)), \tag{14}$$

$$\tau_{m,\ell} = \begin{cases} \tau_m^{AT} & \text{if} \quad |real(x_m(\ell))| \rangle P_m(\ell-1) \\ \tau_m^{RT} & else \end{cases}. \tag{15}$$

Note that the time parameters of AT and RT may be different according to the frequency band. Finally, the signal power at the mth frequency band, $P_m^{dB}(\ell)$, is calculated as

$$P_m^{dB}(\ell) = 20\log_{10} P_m(\ell)\,. \tag{16}$$

Figure 7 shows examples of applying power envelope estimation methods (13)–(15) into the 5th, 10th, and 25th channel signals in Figure 5, with AT = 2 ms and RT = 10 ms.

3.6 Feedback cancellation

3.6.1 Frequency shifting

According to the Nyquist stability criterion two conditions can cause a loop gain that leads to feedback: firstly, the magnitude of the signal travelling around the loop does not decrease in each round trip, and secondly the feedback signal adds up in phase with the microphone signal. If the loop gain of a system exceeds 1 for any frequency, the system may become unstable and this leads to acoustic feedback. In order to minimize the effects of acoustic feedback, a variety of solutions have been suggested [26]. For the ODP, we have chosen the approach of phase modification to decrease the possibility of in-phase addition situation of feedback signal to the microphone signal [27] and to reduce acoustic feedback. The frequency shifting method aims to break the acoustic feedback loop by moving the feedback sound to a different frequency; however, this can lead to sound distortions and quality reduction in the resulting sound. Therefore, the amount of the frequency shifting must be chosen carefully to allow sufficient feedback reduction while the distortions remain small. A frequency shift of 15Hz is generally considered to be sufficient for feedback cancellation while being imperceptible or at least leading to sufficiently undistorted sound [27]. However, if an additional acoustic feedback detector detects feedback, the frequency shift can be made larger because a reduction in quality is preferable to feedback.

In this section, we describe the frequency shifter introduced in [27]. The implementation of the frequency shifter is based on a causal Hilbert filter introducing a short additional time delay, which is also implemented by using a complex finite impulse response (FIR) filter. Therefore, in the case of using a filter bank and frequency shifter together, both filter tap numbers are chosen in order for the full-time delay not to exceed 10 ms.

To describe the frequency shifter, we start with designing a low pass filter with cut-off frequency $fs/4$, $\mathbf{h}_{LP}^{(fs/4)} = [h_{LP}(0), h_{LP}(1) \;\cdots, h_{LP}(Q-1)]^T$, where Q is the number of filter taps. If we choose the filter tap number 128, i.e., $Q = 128$, we can get the coefficients of $h_{LP}^{(fs/4)}$ by MATLAB built-in function "fir1(128,1/2, 'low')". Then, the $\mathbf{h}_{LP}^{(fs/4)}$ is multiplied by exponential term vector of $\exp(j(\pi/2)n)$ for the complex FIR filter $\bar{\mathbf{h}}_{LP}^{(fs/4)}$, i.e.,

$$\overline{\mathbf{h}}_{LP}^{(fs/4)} = \begin{bmatrix} h_{LP}(0)e^{j\frac{\pi}{2}(0)} \\ h_{LP}(1)e^{j\frac{\pi}{2}(1)} \\ h_{LP}(2)e^{j\frac{\pi}{2}(2)} \\ \vdots \\ h_{LP}(Q-1)e^{j\frac{\pi}{2}(Q-1)} \end{bmatrix}. \tag{17}$$

The input signal $x(n)$ is then processed by the complex FIR filter $\overline{\mathbf{h}}_{LP}^{(fs/4)}$ to return a complex output signal $y(n)$. The final frequency shifted signal $\bar{x}^{\Delta f}(n)$ is obtained by

$$\begin{aligned} \bar{x}^{\Delta f}(n) = real(y(n))2\cos\left(2\pi\frac{\Delta f}{f_S}n\right) \\ - imag(y(n))2\sin\left(2\pi\frac{\Delta f}{f_S}n\right), \end{aligned} \tag{18}$$

where $imag(\cdot)$ represents the function giving the imaginary part of the complex value and Δf denotes the frequency shift in Hz.

Figure 8 shows example spectrograms of speech (upper panel) and its 1 kHz-frequency shifted version (lower panel). Please note that in this example the amount of shifted frequency is set to an extremely distorting and unrealistic 1 kHz in order to make the frequency shift effect graphically visible. In real auditory applications, a shift of around 15 Hz is applied which would be invisible in the spectrogram.

3.6.2 Adaptive Feedback Cancellation

Unlike the frequency shifter in the previous sub-section, the adaptive feedback canceller (AFC) does not modify the signal artificially but just tries to cancel the additional feedback components, which generally allows for a better sound quality than a frequency shifter [26], [28], [29]. Despite the benefits of the adaptive feedback canceller, it has potential problems of biased estimation on the feedback signal, which is caused by the correlation between microphone input signal and loudspeaker output signal [19], [30], [31]. The biased estimation degrades the feedback cancelling performance and sound quality. Therefore, almost all current approaches to tackle adaptive feedback cancellation are focusing on the methods for unbiased estimation, where the key concept is to make the microphone input signal uncorrelated with the loudspeaker signal [19], [30], [31]. In this section, we describe the basic AFC system that is implemented on the ODP platform so that researchers can apply their own unbiased estimation solution into the AFC algorithm. The basic AFC may not provide good benefits without unbiased estimation, thus we empirically suggest to use the frequency shifter in the previous subsection at the AFC output signals, because the frequency shifter can also be regarded as the one of the uncorrelation methods to overcome the biased estimation.

The block diagram of the basic AFC is shown in Figure 9. Assuming that an external input signal, $x(n)$, is deteriorated by a feedback signal $v(n)$, then the microphone input time discrete signal $y(n)$ is related to $x(n)$ and $v(n)$, by

$$y(n) = x(n) + v(n). \tag{19}$$

The cancellation of the feedback signal $v(n)$ starts with the assumption that $v(n)$ is a convolved version of the loudspeaker output signal $u(n)$ with an acoustic feedback path $\mathbf{h}^{fb}(n)$, which can be described by an adaptive filter, i.e., $\hat{\mathbf{h}}^{fb} = [\hat{h}^{fb}(0),\ \hat{h}^{fb}(1), \cdots, \hat{h}^{fb}(Q^{AFC}-1)]^T$. Here, Q^{AFC} denotes the number of filter taps. In order words, the estimate of feedback signal $v(n)$, $\hat{v}(n)$, is obtained as

$$\hat{v}(n) = \sum_{i=0}^{Q^{AFC}-1} \hat{h}_i^{fb}(n)u(n-i) \tag{20}$$

Based on (19) and (20), an external input signal $u(n)$ is estimated robustly as

$$\begin{aligned}\hat{x}(n) &= y(n) - \hat{v}(n)\\ &= y(n) - \sum_{i=0}^{Q^{AFC}-1} \hat{h}_i^{fb}(n)u(n-i).\end{aligned} \tag{21}$$

Here, the filter coefficients of $\hat{\mathbf{h}}^{fb}$ are updated based on the least mean square (LMS) algorithm, which is based on minimizing the expectation of the mean square value of $\hat{x}(n)(= y(n) - \hat{f}(n))$. That is,

$$\hat{\mathbf{h}}^{fb}(n+1) = \hat{\mathbf{h}}^{fb}(n) + \mu\mathbf{u}(n)\hat{x}(n), \tag{22}$$

where μ is the step size, which may be a time variable parameter for an normalized LMS (NLMS) algorithm, i.e.,

$$\mu(n) \leftarrow \frac{\mu}{||\mathbf{u}(n)||^2}, \tag{23}$$

where $||\cdot||$ denotes the vector norm operator and

$$\mathbf{u}(n) = [u(n), u(n-1), \cdots, u(n-Q^{AFC}+1)]^T. \tag{24}$$

This implementation of the above adaptive filter included in the ODP library is implemented in the frequency domain.

To demonstrate the effectiveness of the AFC, the acoustic feedback event was simulated by using an impulse response shown in the top panel of Figure 10, where the spectrogram of the 2nd row of Figure 8 was used for the input test signal. Initially, the AFC filter coefficients are filled with zeroes and then constantly updated while processing. The resulting spectrogram is shown in the 3rd row of the figure. As shown in the figure, some feedback is still present because the adaptation does not perfectly estimate the feedback path accurately with initial zero values. Thus, the initial values of filter coefficients are important. We therefore estimated the feedback path with a white noise signal and the resulting values were set as AFC filter coefficients. The resultant spectrogram in the bottom panel of Figure 10 demonstrate to be more effective to cancel the feedback components comparing to the zero-value initialized AFC.

4. ODP algorithm library

Our platform aims to enables current users of MATLAB to quickly port their algorithms to real time. To this end, we provide an open library that can be used freely and that will grow in future. When a library of algorithms is readily available for researchers, faster realization of novel ideas and more advanced and complex systems can be explored and built with more ease. Easy comparisons between different algorithms and implementations can be made such that better system design choices can be made. The algorithms can run on standard PC hardware or on special hardware when required. The platform is the result of collaboration within the ICanHear European Training network (http://www.icanhear.eu). The software in the library as well as supporting documentation can be downloaded for free under a fair use licence.

At the time of writing, the library consists of the described algorithms as well as some other modules that are shown in Table I. A real-time cochlear implant vocoder is also included to make it possible to experience the effects of CI speech processor in normal-hearing listeners. Two types of vocoder (noise carrier and sine wave vocoder) are included in the library. To enable compilation of non-power of two FFT MATLAB function, several point FFTs are also implemented and included in the library.

5. Open hearing aid algorithm platform

In our work we are particularly interested in the development of hearing aids. While the ODP has been developed primarily to build hearing aid, it is by no means restricted to this application. However, hearing aids are becoming more and more important in our society. Today in the UK 6 million people could benefit from an aid, but only 2 million use one. The problem is similar in other countries. One of the most cited reasons for this is the aids' inability to provide benefit in noisy situations. Progress in a signal processing algorithm research has been slow and despite the massively increased processing powers of modern hearing aids, the perceived benefit for the wearer has not improved as dramatically. Basically, today's signal processing algorithms in state of the art hearing aids can improve the wearer's comfort, but they are not improving speech intelligibility in noisy situations. Even though there have been many published noise reduction algorithms, it is not straightforward to apply them to hearing aids and evaluate the benefit because hearing aids are typically composed with combinations of several algorithms such as AFC, low latency filter bank, DRC, envelope power estimator, and so on. We show here the implementation of a fully functioning hearing aid on the ODP – including an (industry standard) noise reduction algorithm - with the following characteristics that are all described in more detail in the previous section:

- 32 channel DFT filter bank implementation using Windowing & 128 point FFT/IFFT & OLA
- 16 kHz sample rate
- Latency 8 ms
- 'Wiener filter' noise reduction algorithm
- LMS adaptive algorithm for feedback cancellation

Figure 11 shows the implemented hearing aid model on Simulink. This implemented model can be downloaded and used for free from the repository so that interested researcher can extend and evaluate their own algorithms for hearing aid applications.

Calibration

Since an auditory device mainly aims to generate audio signals entering the human ear, it is crucial to be able to calibrate each frequency channel accordingly. We used the calibration and equalization techniques that have been set out and explained in detail in [32] for the Speedgoat® target machine. The principle of calibration is that the gain between microphone input and headphone output is defined in each frequency band. Details are described in [32] .

6. Discussion

We present here the ‘Open Development Platform’ ODP that was designed to allow researchers a simplified way to translate offline algorithms into real-time systems ready to be used in experiments in realistic environments with high ecologic validity.

We have created a library with a basic set of real-time algorithms that can be downloaded from the repository at http://www.soundsoftware.ac.uk and we plan to extended the library in future.

The concept of the ODP has been validated by demonstrating a fully functioning multi-channel low-latency hearing aid. The main limitation of the ODP, as with every real-time system is speed: the interrupt service routine must be able to calculate the result of a complex computation within one sample time to avoid acoustic artefacts. However, we argue that the benefit from being able to test many algorithms in real world situations often outweighs the drawback of limited computing power and Moore’s law predicts that processing power will keep on increasing. Algorithms that pushed a standard PC to the limit 10 years ago, now run on the Raspberry Pi, a $3 micro-computer, or a cheap smartphone. Today, when considering the effort of development realistically, one has to weigh the time that it takes to optimize an algorithm via just waiting for the next processor generation.

The described algorithms are reasonably optimized for speed, but not at the detriment of code-legibility. We consider this to be an academic project in which it will almost always be more important to produce and share code that can be read and understood by others rather than to tickle out the last few percentages of processing power on a specific target. The published library is not only a collection of algorithms but also serves as a means of documenting algorithmic ideas and design concepts. The exchange of algorithms between labs can therefore facilitate an exchange of ideas that promotes teaching and learning between groups.

7. Acknowledgement

This work was funded by the People Programme (Marie Curie Actions) of the European Union’s Seventh Framework Programme FP7/2007-2013/under REA grant agreement PITN-GA-2012-317521.

8. References

[1] Gnewikow, D., Ricketts,T., Bratt, G. W., M utchler, L. C., 2009. Real-world benefit from directional microphone hearing aids. J. Rehabil. Res. Dev., 46 (5), 603–618.

[2] Jensen, N. S., Neher, T., Laugesen, S., Johannesson, R. B., Kragelund, L., 2013. Laboratory and field study of the potential benefits of pinna cue-preserving hearing aids. Trends Amplif., 17 (3), 171–88.

[3] Krüger, H., Lotter, T., Enzner, G., Vary, P., 2003. A PC based Platform for Multichannel Realtime Audio Processing. In: Proceedings of International Workshop on Acoustic Echo and Noise Control (IWAENC), Kyoto, pp. 195–198.

[4] Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., Zue, V., 1993. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1.

[5] Varga, A., Steeneken, H. J. M., 1993. Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems. Speech Commun., 12 (3), 247–251.

[6] Grimm, G., Herzke, T., Berg, D., Hohmann, V., 2006. The Master Hearing Aid : A PC-Based Platform for Algorithm Development and Evaluation," Acta Acust. united with Acust., 92, 618–628.

[7] Gopalakrishna, V., Kehtarnavaz, N., Loizou, P., 2010. Real-time implementation of wavelet-based advanced combination encoder on PDA platforms for cochlear implant studies. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Dallas, pp. 1670–1673.

[8] Buchholz, J. M., 2013. A real-time hearing-aid research platform (HARP): Realization, calibration, and evaluation. Acta Acust. united with Acust., 99 (3), 477–492.

[9] Hu, H., Krasoulis, A., Lutman, M. E., Bleeck, S., 2013. Development of a real time sparse non-negative matrix factorization module for cochlear implants by using xPC target. Sensors, 13 (10), 13861–13878.

[10] Rass, U., Steeger, G. H., 2001. A High Performance Pocket-Size System for Evaluations in Acoustic Signal Processing. EURASIP Journal on Advances in Signal Processing, 2001 (3), 163–168.

[11] Hu, H., Sang, J., Lutman, M. E., Bleeck, S., 2013. Non-negative matrix factorization on the envelope matrix in cochlear implant. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, pp. 7790–7794.

[12] Löllmann, H. W., Vary, P., 2009. Low Delay Noise Reduction and Dereverberation for Hearing Aids. EURASIP Journal on Advances in Signal Processing, 2009, 1–9.

[13] Bäuml, R. W., Sörgel, W., 2008. Uniform polyphase filter banks for use in hearing aids: Design and constraints. In: Proceedings of European Signal Processing Conference. Lausanne, 1–5.

[14] Stone, M. A., Moore, B. C., 2003. Tolerable hearing aid delays. III. Effects on speech production and perception of across-frequency variation in delay. Ear Hear., 24 (2), 175–183.

[15] Stone, M. A., Moore, B. C., 1999. Tolerable hearing aid delays. I. Estimation of limits imposed by the auditory path alone using simulated hearing losses. Ear Hear.,20 (3),182–192.

[16] McNally, G. W., 1984. Dynamic range control of digital audio signals. J. Audio Eng. Soc, 32 (5), pp. 316–327.

[17] Giannoulis, D., Massberg, M., Reiss, J. D., 2012. Digital dynamic range compressor design - A tutorial and analysis. AES J. Audio Eng. Soc., 60 (6), 399–408.

[18] Löllmann, H. W., Vary, P., 2007. Uniform and warped low delay filter-banks for speech enhancement. Speech Commun., 49 (7–8), 574–587.

[19] Ngo, K., Waterschoot, T. V, Christensen, M. G., Moonen, M., Jensen, S. H., 2013. Improved prediction error filters for adaptive feedback cancellation in hearing aids. Signal Processing, 93 (11), 3062–3075.

[20] Grimm, G., Herzke, T., Berg, D., Hohmann, V., 2006. The master hearing aid: A PC-based platform for algorithm development and evaluation. Acta Acust. united with Acust., 92 (4), 618–628.

[21] Sit, J. J., Simonson, A. M., Oxenham, A. J., Faltys, M. A, Sarpeshkar, R., 2007. A low-power asynchronous interleaved sampling algorithm for cochlear implants that encodes envelope and phase information. IEEE Trans. Biomed. Eng., 54 (1), 138–149.

[22] Peddigari, V., Kehtarnavaz, N., Loizou, P., 2007. Real-time labview implementation of cochlear implant signal processing on PDA platforms. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Honolulu, pp. 357–360.

[23] Ali, H., Lobo, A. P., Loizou, 2011. A PDA platform for offline processing and streaming of stimuli for cochlear implant research. ," In: Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Boston, pp. 1045–1048.

[24] Ali, H., Lobo, A. P., Loizou, P. C., Hansen, J., 2013. Design and evaluation of a personal digital assistant-based research platform for cochlear implants. IEEE Trans. Biomed. Eng., 60 (11), 3060–3073.

[25] Loizou, P. C., 2007. Speech enhancement: theory and practice. CRC Press.

[26] Waterschoot, T. V., Moonen, M., 2011. Fifty Years of Acoustic Feedback Control: State of the Art and Future Challenges. Proceedings of the IEEE, 99 (2), 288–327.

[27] Scheuing, J., Yang, B., 2006. Frequency shifting for acoustic feedback reduction. In: Proceedings of European DSP Education and Research Symposium (EDERS), Munchen, pp. 1–8.

[28] Guo, M., Jensen, S. H., Jensen, J., 2012. Novel acoustic feedback cancellation approaches in hearing aid applications using probe noise and probe noise enhancement. IEEE Trans. Audio, Speech Lang. Process., 20 (9), 2549–2563.

[29] Waterschoot, T. V., Moonen, M., 2009. Adaptive feedback cancellation for audio applications. Signal Processing, 89 (11), 2185–2201.

[30] Boukis, C., Mandic, D. P., Constantinides, A. G., 2007. Toward bias minimization in acoustic feedback cancellation systems. J. Acoust. Soc. Am., 121 (3), 1529–1537.

[31] Ma, G., Gran, F., Jacobsen, F., Agerkvist, F. T., 2011. Adaptive feedback cancellation with band-limited LPC vocoder in digital hearing aids. IEEE Trans. Audio, Speech Lang. Process., 19 (4), 677–687.

[32] Hu, H., Li, G., Lutman, M. E., Bleeck, S., 2011. Enhanced Sparse Speech Processing Strategy for Cochlear Implants. In: Proceedings of European Signal Processing Conference, Barcelona, 491–495.

Category	algorithms
Auditory Modeling	• Cochlea Vocoder (for use in CI simulations) • Gammatone analysis filter-bank
Frequency Decomposition	• DFT filter-bank summation • Polyphase DFT filter-bank • Non-power of two point FFT/IFFT (48, 96, 160, 192, 320, 384, 640, 768, 896, 1280, 1536, 1792)
Dynamic Range Compressor	• Power envelope estimator • Wide dynamic range compressor
Single-mic. Noise Reduction	• Wiener filter (MATLAB source scripts from [27] • Spectral subtraction(MATLAB source scripts from [27]
Feedback Cancellation	• Notch filter • Frequency shifter • Time domain adaptive feedback cancellation • Frequency domain adaptive feedback cancellation
Etc.	• Overlap-and-add

Table I. List of algorithms included in the ODP library at the time of writing the paper.

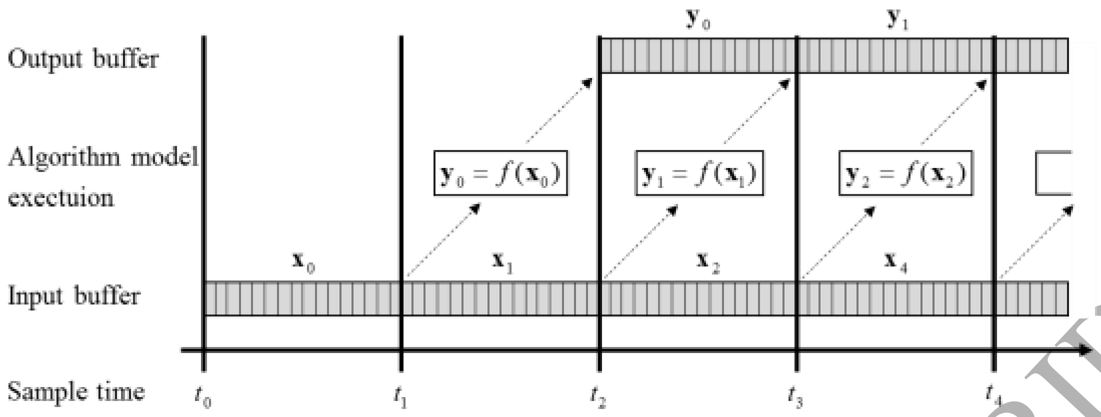


Figure 1. Principle of real-time processing employed by ODP.

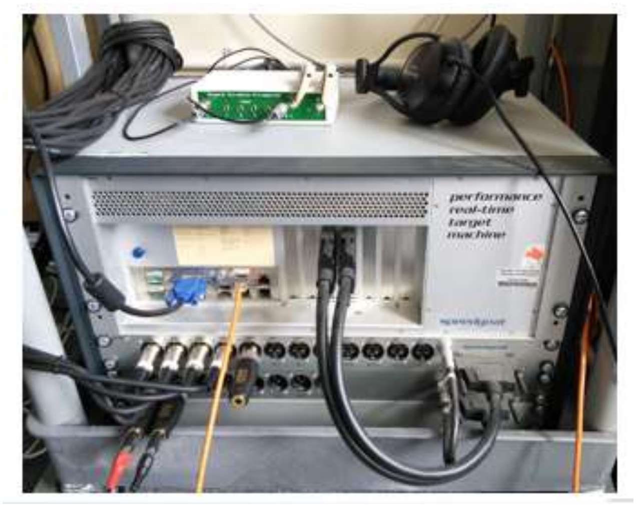


Figure 2. The real-time target machine in our lab is from Speedgoat GmbH running with Simulink Real-Time from MathWorks. The output can be via headphones, loudspeaker or a hearing aid dummy (pictured lying on top of the target machine)

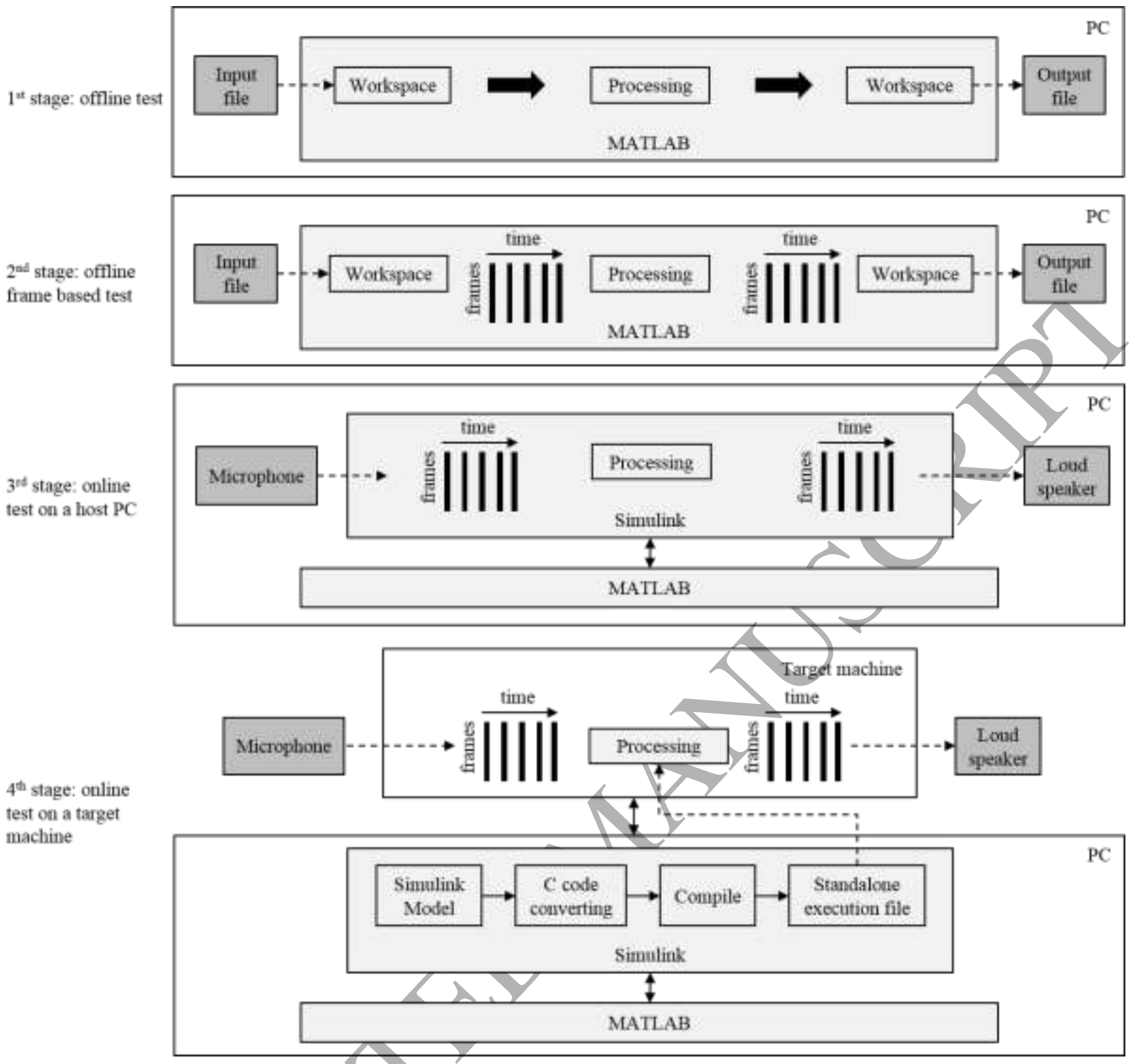
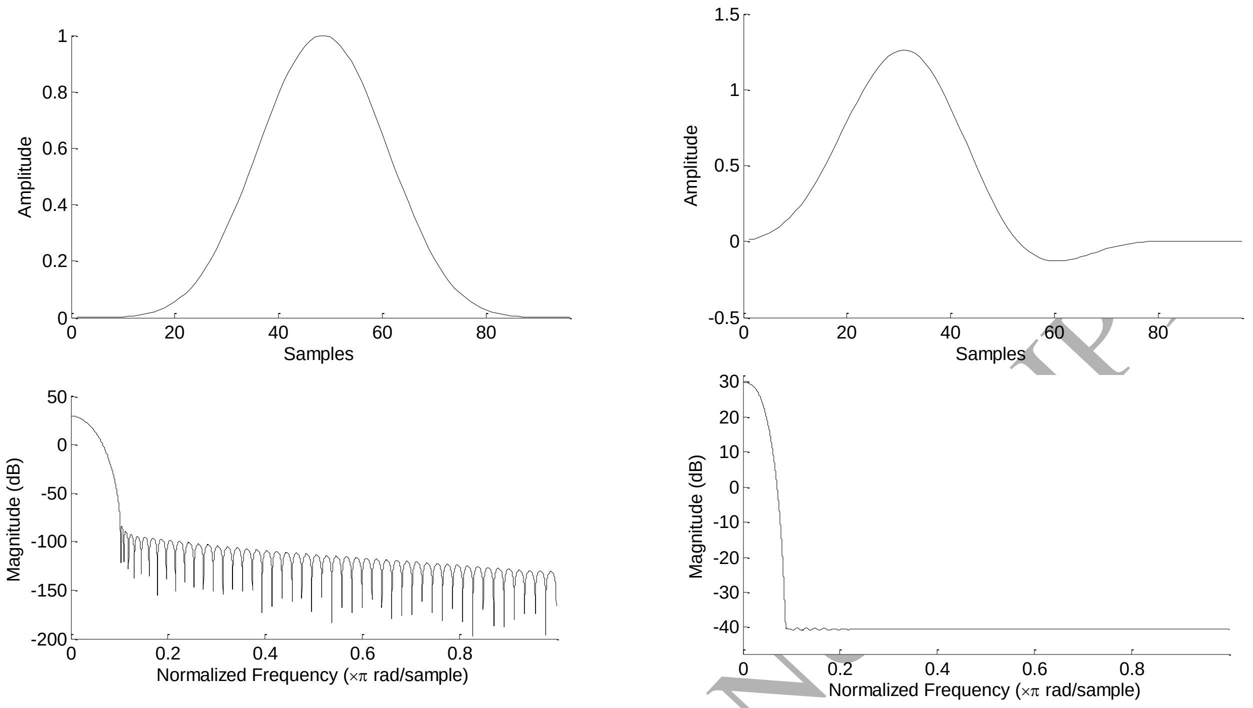


Figure 3. Schematic representation of the 4 development stages of a real-time algorithm.



(a)

(b)

Figure 4. Impulse response of low pass filter prototype with 96 sample point (upper panel) and its magnitude spectra (lower panel) on normalized frequencies; (a) standard Kaiser Window, (b) Lowpass filter in [13].

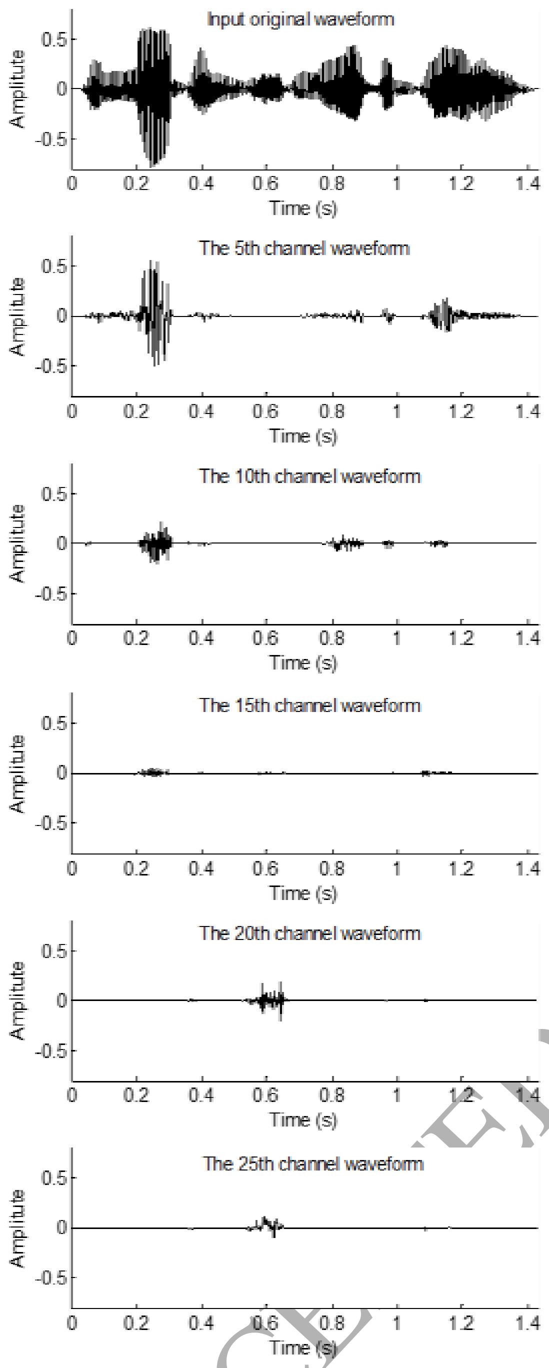


Figure 5. Illustration of the input original speech signal on sampling rate 16kHz (1st row), and 32channel polyphase DFT analysis filter bank output signals at the 5th channel (2nd row), the 10th channel (3rd row), the 15th channel (4th row), the 20th channel (5th row), the 25th channel (6th row). The 5th, 10th, 15th, 20th, and 25th channels are centered at 1125 Hz 2375 Hz, 3625Hz 4875 Hz, and 6125 Hz, respectively.

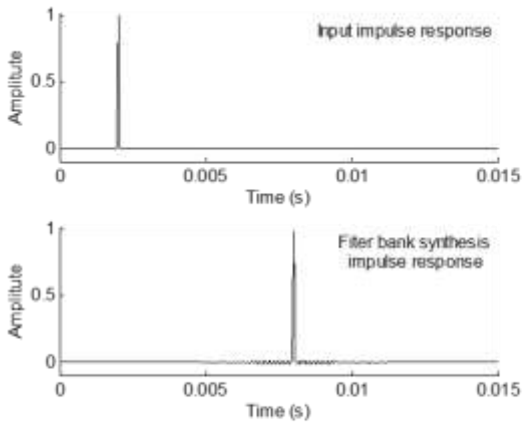


Figure 6. Illustration of the input impulse response (upper panel), and its transfer impulse response of the 32-channel polyphase DFT analysis/synthesis filter bank in section 4.1 (lower panel).

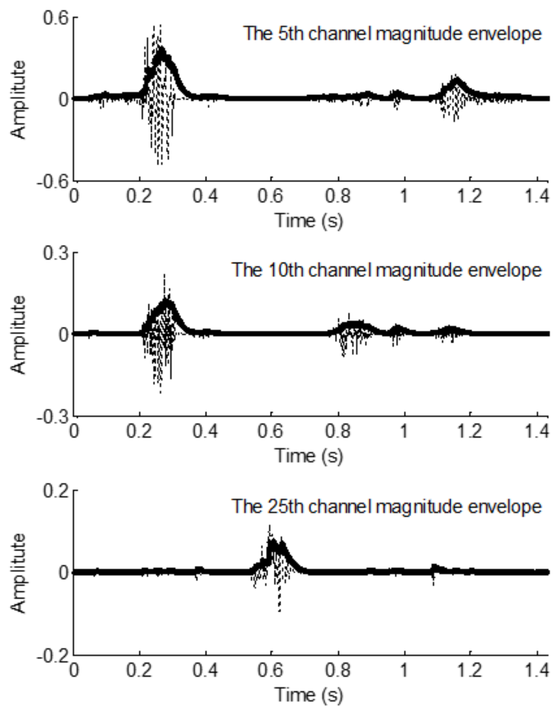


Figure 7. Examples of power envelope estimation. (13) – (15) are applied into the 5th, 10th, and 25th channel signals in Figure 5 where AT 2 ms and RT 10ms are used. The broken line denotes the channel waveform and the bold solid line denotes its magnitude envelope.

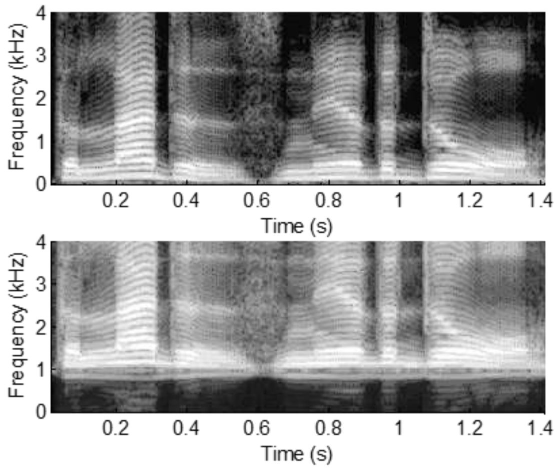


Figure 8. Illustrations of spectrograms of speech (upper panel) and its frequency shifted version (lower panel). In this example the frequency shift is set to an unrealistic 1 kHz in order to make it visible in this figure. In real-world applications, the shift would only be a fraction of this.

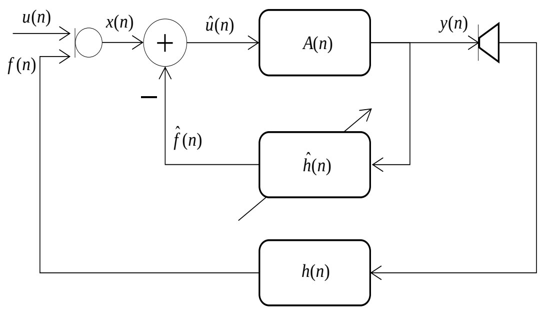
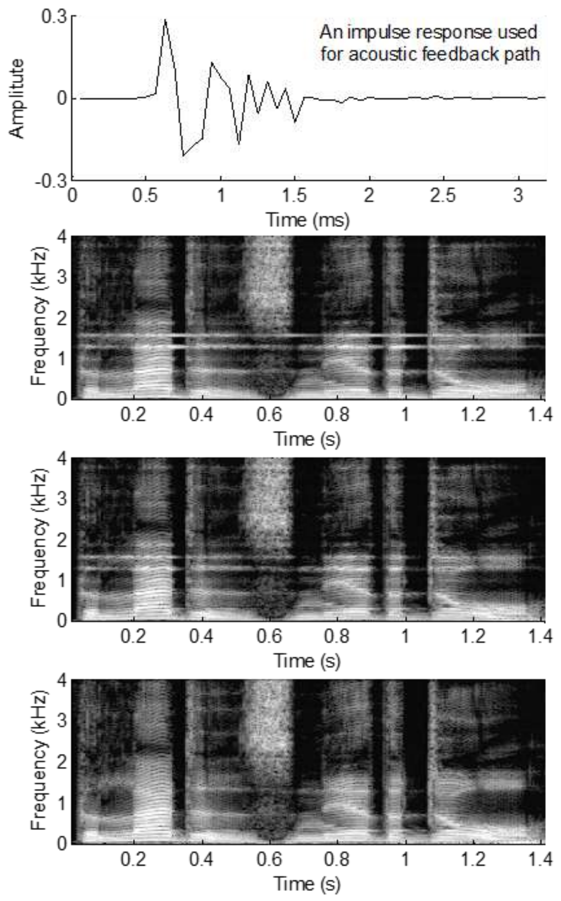


Figure 9. Block diagram of a basic AFC algorithm employed to the ODP.



.

Figure 10. Step-by-step illustrations of adaptive feedback cancellation: the impulse response used for the acoustic feedback path (top row), its simulated resultant spectrogram of the input speech in the upper panel of Figure 8 (2nd row), the AFC processing version without initial feedback path estimation (3rd row), and the AFC processing version with initial feedback path estimation (bottom row).

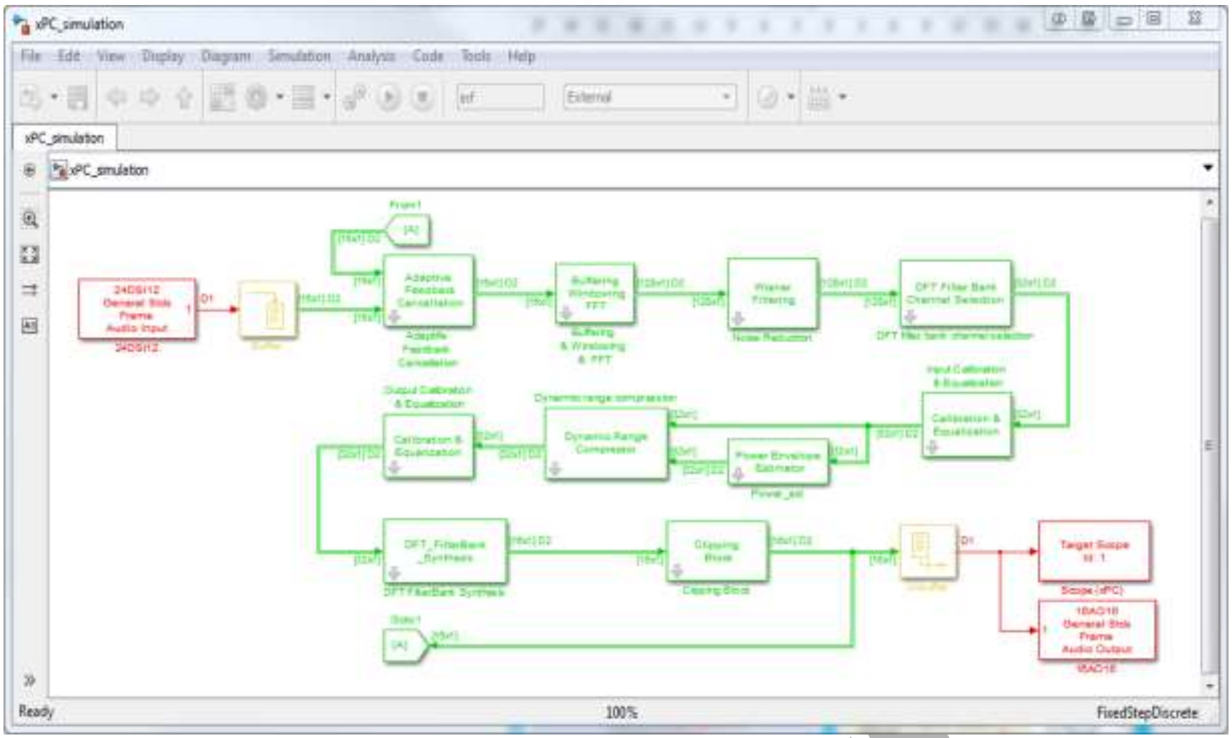

xPC_simulation
File Edit View Display Diagram Simulation Analysis Code Tools Help
inf
External
xPC_simulation
24DS12 General Std Frame Audio Input
24DS12
Buffer
From1
[A]
Adaptive Feedback Cancellation
Buffering Windowing FFT
Buffering & Windowing & FFT
Wiener Filtering
Noise Reduction
DFT Filter Bank Channel Selection
DFT filter bank channel selection
Input Calibration & Equalization
Calibration & Equalization
Dynamic range compression
Dynamic Range Compressor
Power Envelope Estimator
Power_est
Output Calibration & Equalization
DFT_FilterBank_Synthesis
DFT FilterBank Synthesis
Clipping Block
Unbuffer
Target Scope Id: 1
Scope (xPC)
16AO16 General Std Frame Audio Output
16AO16
Goto1
Ready
100%
FixedStepDiscrete

Figure 11. The screen shot of implemented model on Simulink of a multichannel compressive hearing aid with noise cancellation and feedback cancellation. This implemented model is downloaded for free from the ODP library.