Lorente Giner, J.; Ferrer Contreras, M.; Diego Antón, MD.; Gonzalez, A. (2015). The frequency partitioned block modified filtered-x NLMS with orthogonal correction factors for multichannel Active Noise Control. Digital Signal Processing. 43:47-58. doi:10.1016/j.dsp.2015.05.003.

Available online at www.sciencedirect.com

ScienceDirect

Digital Signal Processing 00 (2015) 1–24

Digital Signal Processing

The Frequency Partitioned Block Modified Filtered-X NLMS with Orthogonal Correction Factors for Multichannel Active Noise Control

Jorge Lorente*, Miguel Ferrer, Maria de Diego, Alberto Gonzalez

Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM), Universitat Politècnica de València , Camino de Vera s/n Edificio 8G, Valencia 46022, Spain

Abstract

The Normalized Least Mean Square (NLMS) algorithm with a filtered-x structure (FxNLMS) is a widely used adaptive algorithm for Active Noise Control (ANC) due to its simplicity and ease of implementation. One of the major drawbacks is its slow convergence. A modified filtered-x structure (MFxNLMS) can be used to moderately improve the speed of convergence, but it does not offer a huge improvement. A greater increase in the speed of convergence can be obtained by using the MFxNLMS algorithm with orthogonal correction factors (M-OCF), but the usage of orthogonal correction factors also increases the computational complexity and limits the usage of the M-OCF in massive real-time applications. However, Graphics Processing Units (GPUs) are well known for their potential for highly parallel data processing. Therefore, GPUs seem to be a suitable platform to ameliorate this computational drawback. In this paper, we propose to derive the M-OCF algorithm to a partitioned block-based version in the frequency domain (FPM-OCF) for multichannel ANC systems in order to better exploit the parallel capabilities of the GPUs. The results show improvements in the convergence rate of the FPM-OCF algorithm in comparison to other NLMS-type algorithms and the usefulness of GPU devices for developing versatile, scalable, and low-cost multichannel ANC systems.

Keywords: Active Noise Control, Adaptive Filters, Graphics Processing Unit, Normalized Least Mean Square with Orthogonal Correction Factors

1. Introduction

Active Noise Control (ANC) [1, 2, 3] is a field that combines digital signal processing techniques with traditional acoustics. ANC systems are based on the principle of destructive interference between a disturbance sound field called primary noise and a secondary sound field generated by controlled secondary sources called actuators. The target is to cancel, or at least minimize, the primary noise signal. To cancel the primary noise, the ANC system commonly uses

*Corresponding author. Tel: 34-96-3877007, ext. 73008
Email address: jorlogi@iteam.upv.es (Jorge Lorente)

adaptive algorithms [4] to generate the secondary sound field from a reference signal that is correlated with the primary noise. For this purpose, the noise or undesired signal is monitored at a specific spatial point by a sensor that is called error sensor. Therefore, cancellation is only carried out at that specific spatial point and also at close points around the error sensor (approximately $\lambda/10$, where λ is the wavelength of the highest frequency of the undesired noise). ANC systems can be extended to multichannel ANC systems by overlapping different controlled areas and setting multiple secondary sources [5]. These multichannel systems require a high computational capacity. An even greater capacity is required when massive control systems are considered (a high number of channels, with one channel being defined for each error sensor - secondary source pair), and, in practice, the computational cost is one of the main bottlenecks of multichannel ANC systems.

The filtered-x Normalized Least Mean Squares (FxNLMS) algorithm [6] and its multichannel version (see for example [5]), are the most widely used adaptive filtering techniques applied to single or multiple channel adaptive noise cancelers for ANC applications. This is due to its computational simplicity and ease of implementation. However, one of the major drawbacks of the FxNLMS algorithm is the speed of convergence. The Affine Projection Algorithm (APA)[7], which was developed to address this problem, uses affine subspace projections to speed the convergence of the LMS-type algorithms. From [7], some variants of the APA algorithm have been proposed for different authors, such as the Regularized APA (R-APA)[8], the Partial Rank Algorithm (PRA)[9], the Decorrelating Algorithm (DA)[10] or the NLMS with orthogonal Correction Factors (NLMS-OCF)[11][12]. The basic idea of these algorithms, which are commonly reffered to in the literature as the APA family [13], is to update the weights on the basis of multiple input signal vectors. This is in contrast to the FxNLMS which uses a single input signal vector to update the coefficient weights. In the existing literature, the algorithms of the APA family have traditionally been implemented in the time domain for single-channel system identification or echo cancellation problems. In this paper, we will focus on the NLMS-OCF algorithm because we find its practical implementation easier than the APA implementation, and, as considered in [12], the NLMS-OCF algorithm, which is a generalization of the APA, allows other than unit delay between input vectors. Here, we use the NLMS-OCF algorithm for a multichannel ANC system, but the unavailability of the undesired signal in the ANC system justifies the use of the modified filtered-x scheme. Therefore, the modified filtered-x NLMS-OCF (M-OCF) [1] algorithm is derived from the NLMS-OCF algorithm and is used in a multichannel ANC system. Since the M-OCF uses multiple input vector signals to update the weights, the algorithm is highly demanding from a computational point of view, particularly in multichannel scenarios. This computational drawback limits its implementation in multichannel real-time systems.

Ideally, it would be desirable to have a computationally-simple adaptive algorithm for an ANC system with high performance in terms of convergence rate or disturbance attenuation that can be easily implemented on a hardware device. However, in practice, these characteristics are incompatible in most cases and some kind of trade-off between the

[1] For the sake of simplicity, we will refer throughout paper to the filtered-x NLMS-OCF algorithm as the OCF algorithm and to the modified filtered-x NLMS-OCF algorithm as the M-OCF algorithm.

algorithm performance and the computational cost is needed. Graphics Processing Units (GPUs) are highly parallel programmable co-processors that provide massive computation when the needed operations are properly parallelized. In the last few years, GPUs have been being employed in most of the engineering fields that require intensive computation, and audio signal processing is not an exception. For example, a general overview of audio signal processing on the GPU is given in [14] and [15]. Moreover, there are many recent contributions that leverage GPUs to accelerate acoustic simulations or real-time applications such as: room acoustics [16], acoustics likelihood computation [17], room impulse response reshaping [18], sound localization [19] or wave-field synthesis [20]. There are also GPU contributions in adaptive filtering like: echo cancellation [21] and noise cancellation [22], [23]. Hence, GPUs seem to be a suitable option for multichannel ANC applications where the processing of each channel could be parallelized. Therefore, in contrast to traditional implementations that are based on conventional hardware devices such as Digital Signal Processors (DSPs) [24] and dedicated hardware, this paper presents a scalable multichannel ANC prototype over a GPU platform. For this purpose, the M-OCF has been adjusted to meet the hardware requirements, and, therefore, a frequency-domain partitioned block-based M-OCF algorithm has been proposed. Throughout the paper, it will be called the Frequency Partitioned Block Modified Filtered-X Normalized Least Mean Square algorithm with Orthogonal Correction Factors (FPM-OCF). As we mentioned above, the use of the NLMS-OCF algorithms instead of the APA was motivated in part by the ease of implementation since it can easily be changed to the frequency domain while the implementation of APA in the frequency domain (which is required in most of the practical implementations using non dedicated hardware), is not straightforward. The FPM-OCF algorithm is motivated by the following reasons: the frequency domain allows a fast implementation; the parallel resources of a GPU are better exploited working with blocks of data than by working sample-by-sample, and most of the common audio cards work with block-data buffers. Also, since the adaptive filters could be larger than the block size, the adaptive filters are partitioned [25, 26], and, therefore, the delay is reduced and the parallelization is improved by performing the adaptation of each partition of the filters at the same time.

The main motivation of this work is twofold: on the one hand, to find an algorithm that improves the convergence rate of the NLMS-type algorithms, and on the other, to obtain an efficient version of that algorithm to reduce the computational requirements. For these reasons, we have considered the NLMS-OCF algorithm, in which both the convergence rate and the computational complexity depend on the number of correction factors (R). Therefore, this leads to an adjustable and configurable algorithm with regard to the complexity and the convergence rate. Accordingly, the main contributions of this paper can be summarized in the following points: first, we present the NLMS-OCF algorithm with a modified filtered-x structure embedded (M-OCF) in order to make it suitable for multichannel ANC systems. Second, the M-OCF has been adjusted to its frequency partitioned block-based version (FPM-OCF) in order to meet the hardware requirements of a real-time implementation. Finally, a parallel implementation using a GPU hardware platform is presented to increase the applicability and versatility of the multichannel system.

This paper is organized as follows: section 2 presents the FPM-OCF algorithm derived from the NLMS-OCF. The ANC prototype is described in section 3 and the GPU implementation is explained in section 4. Finally, sections 5

Table 1. Notation for the multichannel ANC algorithms considered

I	Number of reference signals (reference sensors)	J	Number of secondary sources (actuators)
K	Number of error signals (error sensors)	B	Block size
L	Length of the adaptive filters	F	L/B, number of partitions of the adaptive filters
M	Length of the FIR filters that model the estimated secondary paths	P	M/B, number of partitions of the estimated secondary paths
R	Number of successive input vectors used in the weight update of the NLMS-OCF algorithm	D	Number of iterations between the successive input vectors used in the weight update of the NLMS-OCF algorithm
$x_{[i]}(q)$	ith reference signal at sample q	$\mathbf{x}_{[i]Bn}$	$[x_{[i]}(Bn) \quad x_{[i]}(Bn-1) \quad \dots \quad x_{[i]}(Bn-B+1)]^T$
$y_{[j]}(q)$	jth actuator signal at sample q	$\mathbf{y}_{[j]Bn}$	$[y_{[j]}(Bn) \quad y_{[j]}(Bn-1) \quad \dots \quad y_{[j]}(Bn-B+1)]^T$
$e_{[k]}(q)$	kth microphone signal at sample q	$\mathbf{e}_{[k]Bn}$	$[e_{[k]}(Bn) \quad e_{[k]}(Bn-1) \quad \dots \quad e_{[k]}(Bn-B+1)]^T$
$\mathbf{s}_{[jk]}$	M-length estimation of the secondary path that links the jth secondary source with the kth error sensor		
$\mathbf{S}^p_{[jk]}$	FFT of size $2B$ of the pth partition of the acoustic path $\mathbf{s}_{[jk]}$		
$\mathbf{w}_{[ij]q}$	Coefficients of the adaptive filter of length L that links the ith reference signal with the jth secondary source at sample q		
$\mathbf{W}^f_{[ij]n}$	FFT of size $2B$ of the fth partition of the coeffitients of the adaptive filter that links the ith reference signal with the jth secondary source at the nth block iteration		

and 6 respectively present the experimental results and conclusions.

The notation used throughout this paper is the following: boldface symbols are used for vectors, lowercase letters are used for time domain signals, and uppercase letters are used for frequency domain signals; $\|\cdot\|^2$ denotes the 2-norm, $(\cdot)^*$ denotes the complex conjugate, and $\circ$ denotes the element-wise product of two vectors.

2. Description of the algorithms

This section focuses on describing the M-OCF algorithm for a multichannel ANC system that is derived from the NLMS-OCF algorithm introduced in [11] for a system identification application. In [11], the NLMS-OCF algorithm is derived from the NLMS algorithm. The NLMS algorithm adapts the filter weights so that the error signal $e(q)$ is minimized in the mean-square sense. When both the input $x(q)$ and desired output $d(q)$ signals are stationary, the algorithm converges to a filter weights which, on average, are equal to the Wiener-Hopf solution. In other words, the NLMS is an iterative algorithm to implement Wiener filters, without explicitly solve the Wiener-Hopf equation. Finally, the NLMS algorithm is a stochastic implementation of the steepest-descent algorithm, and the adaptation of the filters is done replacing the cost function $\xi = E[e^2(q)]$ by its instantaneous error [27].

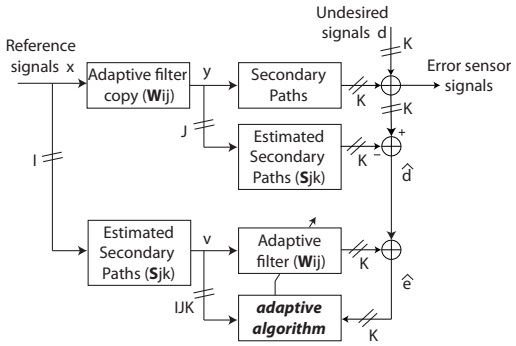

Figure 1. Scheme of the multichannel ANC system.

On the other hand, the NLMS-OCF algorithm solves a constrained minimization problem. As the NLMS-OCF algorithm uses R input vectors for the filter weights update, it iteratively solves the constrained minimization problem until each of the R a posteriori error is forced to zero. In summary, it finds the filter weights $\mathbf{w}_q^r$ such that $\|\mathbf{w}_q^r - \mathbf{w}_q\|$ is minimized subject to $d_{q-r} - \mathbf{x}_{q-r}^T \mathbf{w}_q^r = 0$ for $r = 0, 1, ..., R$ (see appendix of [12]).

In a system identification application, where the NLMS-OCF was introduced, the error signal is the error in estimating the so-called desired signal that is picked up by the microphones. In contrast, in an ANC system, the microphones pick up a mixture of the disturbance or undesired signals and the signals that are used to cancel the noises. Therefore, since the desired signals are not available and they are needed to calculate the error signals [28], we have to use a modified filtering scheme to estimate them. Finally, the FPM-OCF algorithm has been derived from the M-OCF algorithm in order to adjust the algorithm to meet the hardware requirements and to seek the best parallel implementation.

For this purpose, a generic multichannel ANC system with I reference signals, J secondary sources, and K error sensors (I:J:K) has been considered. Moreover, we define L as the length of the adaptive filters and M as the length of the FIR filters that model the estimated secondary paths. The notation in Table 1 will be used to describe the algorithms. For the estimation of the secondary path, we have used an off-line modeling which is obtained by using FIR filters and considering perfect estimation.

2.1. The Modified Filtered-x NLMS algorithm (MFxNLMS)

There are two filtered-x schemes that have been traditionally used in the ANC systems, the conventional filtered-x scheme (which is the most widely used), and the modified filtered-x scheme. The modified filtered-x scheme is computationally more expensive, but it provides faster convergence. However, the reason for our use of the modified filtered-x scheme is not based on the performance or the computational load. As mentioned in the introduction, the use of the modified filtering scheme is mainly based on the need to estimate the undesired signals. The block diagram of a multichannel ANC system based on the MFxNLMS [29] algorithm is depicted in Fig. 1. The following steps constitute the multichannel version of the MFxNLMS algorithm:

1. Calculation of the output signals.

   The jth output signal at the qth instant is calculated as

$$y_{[j]}(q) = \sum_{i=1}^{I} \mathbf{w}_{[ij]q}^{T} \mathbf{x}_{[i]Lq}, \tag{1}$$

   where $\mathbf{x}_{[i]Lq} = [x_{[i]}(q) \quad x_{[i]}(q-1) \quad x_{[i]}(q-2) \quad \dots \quad x_{[i]}(q-L+1)]^T$ is the ith input signal vector with the last L samples from sample q.

2. Calculation of the estimated undesired signals:

$$\hat{d}_{[k]}(q) = e_{[k]}(q) - yf_{[k]}(q), \tag{2}$$

   where $e_{[k]}(q)$ is the qth sample of the kth microphone signal, and sample $yf_{[k]}(q)$ is defined as

$$yf_{[k]}(q) = \sum_{j=1}^{J} \mathbf{s}_{[jk]}^{T} \mathbf{y}_{[j]Mq}, \tag{3}$$

   where $\mathbf{y}_{[j]Mq} = [y_{[j]}(q) \quad y_{[j]}(q-1) \quad y_{[j]}(q-2) \quad \dots \quad y_{[j]}(q-M+1)]^T$ is the jth output signal vector with the last M samples from sample q.

3. Filter updates:

$$\mathbf{w}_{[ij]q+1} = \mathbf{w}_{[ij]q} - \sum_{k=1}^{K} \hat{\mu}_{[ijk]} \mathbf{v}_{[ijk]Lq}, \tag{4}$$

   where the step-size $\hat{\mu}_{[ijk]}$ is calculated as

$$\hat{\mu}_{[ijk]} = \mu \frac{\hat{e}_{[k]}(q)}{\|\mathbf{v}_{[ijk]Lq}\|^2}, \tag{5}$$

   with μ being the step size parameter and $\hat{e}_{[k]}(q)$ being the error signal when estimating the qth sample of the kth undesired signal, $\hat{d}_{[k]}(q)$, given by

$$\hat{e}_{[k]}(q) = \hat{d}_{[k]}(q) + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{[ij]q}^{T} \mathbf{v}_{[ijk]Lq}, \tag{6}$$

   where $\mathbf{v}_{[ijk]Lq} = [v_{[ijk]}(q) \quad v_{[ijk]}(q-1) \quad v_{[ijk]}(q-2) \quad \dots \quad v_{[ijk]}(q-L+1)]^T$ is a vector with the last L samples of the ith input signal filtered through the jkth secondary path, which is calculated as

$$v_{[ijk]}(q) = \mathbf{s}_{[jk]}^{T} \mathbf{x}_{[i]Mq}, \tag{7}$$

   where $\mathbf{x}_{[i]Mq} = [x_{[i]}(q) \quad x_{[i]}(q-1) \quad x_{[i]}(q-2) \quad \dots \quad x_{[i]}(q-M+1)]^T$.

2.2. The Modified Filtered-x NLMS algorithm with Orthogonal Correction Factors (M-OCF)

The NMLS-OCF was introduced in [11] for a system identification mode. This algorithm updates the weights on the basis of R multiple input vectors, while the NLMS algorithm updates the adaptive filter weights on the basis of a single vector. The parameter D is defined as the number of iterations between the successive input vectors used in the weight update. The best improvement in convergence rate occurs if the successive input vectors (corrections) are orthogonal. When the successive input vectors are not orthogonal, the authors in [11] proposed generating orthogonal directions and moving along those directions.

Here we introduce the M-OCF for an ANC system derived from the MFxNLMS similarly to the way the NLMS-OCF is derived from the NLMS in [11] in a system identification application. The equations for calculating the output signals and the estimated undesired signals are the same as in the MFxNLMS algorithm. However, the weight update equation in (4) is modified as [12]

$$\mathbf{w}_{[ij]q+1} = \mathbf{w}_{[ij]q} - \sum_{k=1}^{K} \hat{\mu}^{0}_{[ijk]} \mathbf{v}^{0}_{[ijk]Lq} - \sum_{k=1}^{K} \hat{\mu}^{1}_{[ijk]} \mathbf{v}^{1}_{[ijk]Lq} - \ldots - \sum_{k=1}^{K} \hat{\mu}^{R}_{[ijk]} \mathbf{v}^{R}_{[ijk]Lq} = \mathbf{w}_{[ij]q} - \sum_{r=0}^{R} \sum_{k=1}^{K} \hat{\mu}^{r}_{[ijk]} \mathbf{v}^{r}_{[ijk]Lq}, \tag{8}$$

where R is the number of input vectors that are used in the tap weight adaptation, and $\mathbf{v}^{0}_{[ijk]Lq}$, $\mathbf{v}^{1}_{[ijk]Lq}$,..., $\mathbf{v}^{R}_{[ijk]Lq}$ are orthogonal to each other. As before, $\mathbf{v}^{0}_{[ijk]Lq}$ is the filtered input vector at the qth instant, and $\hat{\mu}^{0}_{[ijk]}$ is chosen as in the FxNLMS

$$\hat{\mu}^{0}_{[ijk]} = \mu \frac{\hat{e}_{[k]}(q)}{\|\mathbf{v}^{0}_{[ijk]Lq}\|^2}. \tag{9}$$

Let us define the vector $\mathbf{v}^{r}_{[ijk]Lq}$, which is obtained from $\mathbf{v}_{[ijk]L_{q-rD}}$ but constrained to be orthogonal to $\mathbf{v}_{[ijk]L_q}$, $\mathbf{v}_{[ijk]L_{q-D}}$, $\mathbf{v}_{[ijk]L_{q-2D}}$, ..., $\mathbf{v}_{[ijk]L_{q-(r-1)D}}$, where D is the distance between the input vectors. It can be calculated using the Gram-Schmidt procedure [30]. The corresponding step-size $\hat{\mu}^{r}_{[ijk]}$ is calculated as

$$\hat{\mu}^{r}_{[ijk]} = \begin{cases} \mu \dfrac{\hat{e}^{r}_{[k]}(q)}{\|\mathbf{v}^{r}_{[ijk]Lq}\|^2}, & \text{if } \|\mathbf{v}^{r}_{[ijk]Lq}\|^2 \neq 0 \\ 0, & otherwise \end{cases} \tag{10}$$

where

$$\hat{e}^{r}_{[k]}(q) = \hat{d}_{[k](q-rD)} + \sum_{i=1}^{I} \sum_{j=1}^{J} {\mathbf{w}^{r}_{[ij]q+1}}^{T} \mathbf{v}_{[ijk]L_{q-rD}}, \tag{11}$$

and

$$\mathbf{w}^{r}_{[ij]q+1} = \mathbf{w}_{[ij]q} - \sum_{k=1}^{K} \hat{\mu}^{0}_{[ijk]} \mathbf{v}^{0}_{[ijk]Lq} - \sum_{k=1}^{K} \hat{\mu}^{1}_{[ijk]} \mathbf{v}^{1}_{[ijk]Lq} - \ldots - \sum_{k=1}^{K} \hat{\mu}^{r-1}_{[ijk]} \mathbf{v}^{r-1}_{[ijk]Lq}. \tag{12}$$

The R parameter must be chosen depending on the computational resources available, but when R increases, the convergence is faster. However, there exists a value of R where the convergence rate saturates. With regard to D, larger values of D provide better performance, especially for low values of R. This might be explained by the fact that

when D increases, the successive input signals used in the weight update are less correlated. Table 2 summarizes the M-OCF algorithm.

2.3. The Frequency-domain Partitioned Block Modified Filtered-x NLMS with Orthogonal Correction Factors algorithm (FPM-OCF)

Finally, the FPM-OCF algorithm has been derived from the M-OCF algorithm. In this algorithm, samples are processed by blocks of size B. L is the length of the adaptive filters, and M is the length of the FIR filters that model the estimated secondary paths. If L and M are higher than B, we have to split up both the adaptive filter length and the estimated secondary path length into F and P partitions, respectively [4]. Thus, the algorithm works simultaneously with all the partitions of size B. Furthermore, the sub-index n and the super-indexes f and p of the following notation denote block iteration and the number of each partition, respectively.

According to the notation, the following steps describe the FPM-OCF algorithm:

1. Calculation of the output signals,

$$\mathbf{Y}_{[j]n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}_{[ij]n}^{f} \circ \mathbf{X}_{[i]n-f+1}, \tag{13}$$

   where $\mathbf{X}_{[i]n} = \mathrm{FFT}\{[\mathbf{x}_{[i]B(n-1)} \quad \mathbf{x}_{[i]Bn}]\}$, $\mathbf{W}_{[ij]n}^{f}$ is the FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_{[ij]}$ at the nth block iteration. The output signals $\mathbf{y}_{[j]Bn}$ are the last B samples of $\mathrm{IFFT}\{\mathbf{Y}_{[j]n}\}$.

2. Calculation of the estimated undesired signals,

$$\hat{\mathbf{D}}_{[k]n} = \mathbf{E}_{[k]n} - \mathbf{YF}_{[k]n}, \tag{14}$$

   where vector $\mathbf{E}_{[k]n}$ is the kth microphone signal in the frequency domain, $\mathbf{E}_{[k]n} = \mathrm{FFT}\{[\mathbf{0}_B \quad \mathbf{e}_{[k]Bn}]\}$, and vector $\mathbf{YF}_{[k]n}$ is defined as

$$\mathbf{YF}_{[k]n} = \sum_{j=1}^{J} \sum_{p=1}^{P} \mathbf{Y}_{[j]n-p+1} \circ \mathbf{S}_{[jk]}^{p}. \tag{15}$$

3. Filter updates.

   The update of the coefficients of each partition of the ijth adaptive filter is calculated in the frequency domain taking into account the R input vectors and is given by

$$\mathbf{W}_{[ij]n+1}{}^{f} = \mathbf{W}_{[ij]n}{}^{f} - \sum_{k=1}^{K} \mathrm{FFT}\{[\boldsymbol{\phi}_{[ijk]0}^{f} \quad \mathbf{0}_B]\} - \sum_{k=1}^{K} \mathrm{FFT}\{[\boldsymbol{\phi}_{[ijk]1}^{f} \quad \mathbf{0}_B]\} - \ldots - \sum_{k=1}^{K} \mathrm{FFT}\{[\boldsymbol{\phi}_{[ijk]R}^{f} \quad \mathbf{0}_B]\}, \tag{16}$$

   where R is the number of input vectors that are used in the tap weight adaptation. These R input vectors are the input signals that are filtered through the corresponding secondary path, $\mathbf{V}_{[ijk]n}$ of the last $R \cdot D$ iterations, where D is the distance in iterations between the input vectors. Vector $\mathbf{V}_{[ijk]n}$ is defined as

$$\mathbf{V}_{[ijk]n} = \sum_{p=1}^{P} \mathbf{S}^{p}_{[jk]} \circ \mathbf{X}_{[i]n-p+1}. \tag{17}$$

Generalizing for an input vector r, the vector $\boldsymbol{\phi}^{f}_{[ijk]r}$ corresponds to the first B samples of the 2B-IFFT of the corresponding partition $\widetilde{\boldsymbol{\mu}}^{f}_{[ijk]r}$

$$[\boldsymbol{\phi}^{f}_{[ijk]r} \quad \bar{\boldsymbol{\phi}}^{f}_{[ijk]r}] = \text{IFFT}\{\widetilde{\boldsymbol{\mu}}^{f}_{[ijk]r}\}, \tag{18}$$
$$f = 1, 2, \ldots, F, \textit{and} \quad r = 1, 2, \ldots, R.$$

Let us define the vector $\mathbf{V}^{r}_{[ijk]n-f+1}$ which is obtained from $\mathbf{V}_{[ijk]_{n-f+1-(rF)D}}$ but constrained to be orthogonal to $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-(F)D}}$, $\mathbf{V}_{[ijk]_{n-f+1-(2F)D}}, \ldots, \mathbf{V}_{[ijk]_{n-f+1-((r-1)F)D}}$, where D is the distance between the input vectors. The corresponding step-size vector $\widetilde{\boldsymbol{\mu}}^{f}_{[ijk]r}$ is calculated as

$$\widetilde{\boldsymbol{\mu}}^{f}_{[ijk]r} = \begin{cases} \mu \dfrac{\hat{\mathbf{E}}^{r}_{[k]n} \circ (\mathbf{V}^{r}_{[ijk]n-f+1})^{*}}{\|\mathbf{V}^{r}_{[ijk]n-f+1}\|^{2}}, & \text{if } \|\mathbf{V}^{r}_{[ijk]n-f+1}\|^{2} \neq 0 \\ \mathbf{0}_{2B}, & \textit{otherwise}. \end{cases} \tag{19}$$

The estimated error signals $\hat{\mathbf{E}}^{r}_{[k]n}$ are calculated as

$$\hat{\mathbf{E}}^{r}_{[k]n} = \hat{\mathbf{D}}_{[k]n-(r)D} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]n-f+1-(rF)D} \circ {\mathbf{W}^{r}_{[ij]n}}^{f}. \tag{20}$$

With regard to the R and D parameters, since this algorithm works by blocks of samples, even when $D = 1$, the successive input signals used in the weight update are separated by at least B samples. Moreover, when the reference signal $\mathbf{x}_{[i]}$ is a white random noise, each block of samples $\mathbf{x}_{[i]Bn}$ is orthogonal to the previous ones. This means that $\mathbf{V}_{[ijk]_{n-f+1-rF}}$ is orthogonal to $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-F}}$, $\mathbf{V}_{[ijk]_{n-f+1-2F}} \cdots$, $\mathbf{V}_{[ijk]_{n-f+1-(r-1)F}}$, and, therefore, the orthogonalization procedure is already done by the nature of the input signal, so we can take $\mathbf{V}^{r}_{[ijk]n-f+1} = \mathbf{V}_{[ijk]_{n-f+1-rF}}$. This fact makes the NLMS-OCF algorithm especially suited to cancel broadband white noise signals, and for this reason, it will be the case studied in the results section.

Note that the FPM-OCF algorithm in the particular case when $R = 0$ and $D = 1$ is similar to the one introduced in [31] named FPBMFxLMS, but adding a power normalization in the weight update. In this paper we will refer to the FPM-OCF algorithm with no orthogonal correction factors ($R = 0$) as the FPM algorithm. A summary of the algorithm instructions executed at each iteration is given in Table 2 for the M-OCF algorithm and the FPM-OCF algorithm.

3. Prototype description

The multichannel ANC prototype is depicted in Fig. 2. With regard to the computing platform, the algorithm has been implemented in two ways: using a CPU with a sequential execution and using a GPU with a parallel execution.

Table 2. Summary of the M-OCF and the FPM-OCF algorithms.

M-OCF algorithm	**FPM-OCF algorithm**
Choose an arbitrary $\mathbf{w}_{[ij]0}$ and the number of Orthogonal Correction Factors $R < L$. Repeat the following steps at each new iteration.	Choose an arbitrary $\mathbf{W}_{[ij]0}$ and the number of Orthogonal Correction Factors $R < L$. Repeat the following steps at each new iteration.
(1) $y_{[j]}(q) = \sum_{i=1}^{I} \mathbf{w}_{[ij]q}^T \mathbf{x}_{[i]Lq}$, $\mathbf{y}_{[j]Mq} = [y_{[j]}(q) \quad y_{[j]}(q-1) \quad \dots \quad y_{[j]}(q-M+1)]^T$	(1) $\mathbf{Y}_{[j]n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}_{[ij]n}{}^f \circ \mathbf{X}_{[i]n-f+1}$
(2) $\hat{d}_{[k]}(q) = e_{[k]}(q) - \sum_{j=1}^{J} \mathbf{s}_{[jk]}^T \mathbf{y}_{[j]Mq}$	(2) $\hat{\mathbf{D}}_{[k]n} = \mathbf{E}_{[k]n} - \mathbf{YF}_{[k]n}$
(3) $v_{[ijk]}(q) = \mathbf{s}_{[jk]}^T \mathbf{x}_{[i]Mq}$, $\mathbf{v}_{[ijk]Lq} = [v_{[ijk]}(q) \quad v_{[ijk]}(q-1) \quad \dots \quad v_{[ijk]}(q-L+1)]^T$	(3) $\mathbf{YF}_{[k]n} = \sum_{j=1}^{J} \sum_{p=1}^{P} \mathbf{Y}_{[j]n-p+1} \circ \mathbf{S}_{[jk]}^p$
(4) $\hat{e}_{[k]}(q) = \hat{d}_{[k]}(q) + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{[ij]q}^T \mathbf{v}_{[ijk]Lq}$	(4) $\mathbf{V}_{[ijk]n} = \sum_{p=1}^{P} \mathbf{S}_{[jk]}^p \circ \mathbf{X}_{[i]n-p+1}$
(5) $\hat{\mu}_{[ijk]} = \mu \frac{\hat{e}_{[k]}(q)}{\|\mathbf{v}_{[ijk]Lq}\|^2}$	(5) $\hat{\mathbf{E}}_{[k]n} = \hat{\mathbf{D}}_{[k]n} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]n-f+1} \circ \mathbf{W}_{[ij]n}{}^f$
(6) $\mathbf{w}_{[ij]q+1}^1 = \mathbf{w}_{[ij]q} - \sum_{k=1}^{K} \hat{\mu}_{[ijk]} \mathbf{v}_{[ijk]Lq}$	(6) $\widetilde{\boldsymbol{\mu}}_{[ijk]}^f = \mu \frac{\hat{\mathbf{E}}_{[k]n} \circ (\mathbf{V}_{[ijk]n-f+1})^*}{\|\mathbf{V}_{[ijk]n-f+1}\|^2}$
For all $1 \le r \le R$, do	(7) $[\boldsymbol{\phi}_{[ijk]}^f \quad \bar{\boldsymbol{\phi}}_{[ijk]}^f] = \text{IFFT}\{\widetilde{\boldsymbol{\mu}}_{[ijk]}^f\}$, $f = n, n-1, \dots, n-F+1$
(7) Compute $\mathbf{v}_{[ijk]Lq}^r$ as the vector obtained from $\mathbf{v}_{[ijk]Lq-rD}$ that is orthogonal to $\mathbf{v}_{[ijk]Lq}$, $\mathbf{v}_{[ijk]Lq-D}$, $\mathbf{v}_{[ijk]Lq-2D}$, …, $\mathbf{v}_{[ijk]Lq-(r-1)D}$ using the Gram-Schmidt procedure [30].	(8) $\mathbf{W}_{[ij]n+1}^1{}^f = \mathbf{W}_{[ij]n}{}^f - \sum_{k=1}^{K} \text{FFT}\{[\boldsymbol{\phi}_{[ijk]}^f \quad \mathbf{0}_B]\}$
(8) $\hat{e}_{[k]}^r(q) = \hat{d}_{[k](q-rD)} + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{[ij]q+1}^r{}^T \mathbf{v}_{[ijk]Lq-rD}$	For all $1 \le r \le R$, do
(9) $\hat{\mu}_{[ijk]}^r = \begin{cases} \mu \frac{\hat{e}_{[k]}^r(q)}{\|\mathbf{v}_{[ijk]Lq}^r\|^2}, & \text{if } \|\mathbf{v}_{[ijk]Lq}^r\|^2 \neq 0 \\ 0, & otherwise \end{cases}$	(10) Compute $\mathbf{V}_{[ijk]n-f+1}^r$ as the vector obtained from $\mathbf{V}_{[ijk]_{n-f+1-(rF)D}}$ that is orthogonal to $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-(F)D}}$, $\mathbf{V}_{[ijk]_{n-f+1-(2F)D}}$, …, $\mathbf{V}_{[ijk]_{n-f+1-((r-1)F)D}}$.
(10) $\mathbf{w}_{[ij]q+1}^{r+1} = \mathbf{w}_{[ij]q+1}^r - \sum_{k=1}^{K} \hat{\mu}_{[ijk]}^r \mathbf{v}_{[ijk]Lq}^r$	(11) $\hat{\mathbf{E}}_{[k]n}^r = \hat{\mathbf{D}}_{[k]n-(r)D} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]n-f+1-(rF)D} \circ \mathbf{W}_{[ij]n}^r{}^f$
end for	(12) $\widetilde{\boldsymbol{\mu}}_{[ijk]r}^f = \begin{cases} \mu \frac{\hat{\mathbf{E}}_{[k]n}^r \circ (\mathbf{V}_{[ijk]n-f+1}^r)^*}{\|\mathbf{V}_{[ijk]n-f+1}^r\|^2}, & \text{if } \|\mathbf{V}_{[ijk]n-f+1}^r\|^2 \neq 0 \\ \mathbf{0}_{2B}, & otherwise. \end{cases}$
(11) $\mathbf{w}_{[ij]q+1} = \mathbf{w}_{[ij]q+1}^R$	(13) $[\boldsymbol{\phi}_{[ijk]r}^f \quad \bar{\boldsymbol{\phi}}_{[ijk]r}^f] = \text{IFFT}\{\widetilde{\boldsymbol{\mu}}_{[ijk]r}^f\}$,
	(14) $\mathbf{W}_{[ij]n+1}^{r+1}{}^f = \mathbf{W}_{[ij]n+1}^r{}^f - \sum_{k=1}^{K} \text{FFT}\{[\boldsymbol{\phi}_{[ijk]r}^f \quad \mathbf{0}_B]\}$
	end for
	(15) $\mathbf{W}_{[ij]n+1}{}^f = \mathbf{W}_{[ij]n+1}^R{}^f$

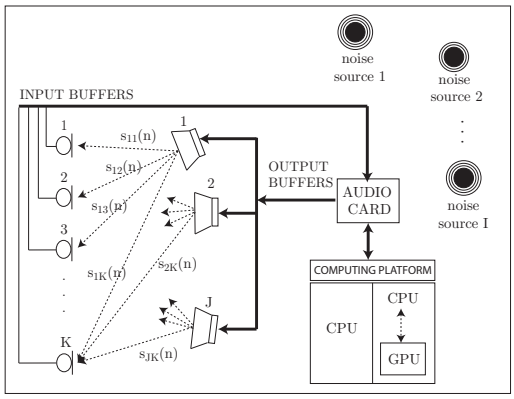

Figure 2. Scheme of the multichannel ANC system.

The CPU is an Intel Core i7 (3.07 GHz), and the GPU is a GeForce GTX 580 with Fermi architecture. The audio card is a MOTU 24I/O. The MOTU audio card uses the ASIO (Audio Stream Input/Output) driver to communicate with the CPU. The ASIO driver provides input/output buffers that are used to collect/send the current microphone and loudspeaker signals. The input buffers are linked to the microphones and the output buffers are linked to the loudspeakers.

The results will show that GPU parallelization is useful in multichannel scenarios. The operation of the GPU-based prototype consists of the following three tasks steps are executed in each iteration:

1. Collect the K input-data buffers of size B from the sensors and transfer them through the PCI-Express bus to the GPU.
2. Carry out the corresponding algorithm on the GPU.
3. Save the output audio samples in the J output-data buffers and send them back to the CPU to be reproduced by the loudspeakers.

The sampling rate (f_s) and the block size (B) are two important parameters of the audio card. The block size describes the number of transferred discrete-time samples per iteration and thereby determines the latency of the algorithm. The latency is the time that is spent from when the input-data buffer is filled up until this data buffer is processed and sent back to the output-data buffer. The time that is spent to fill up the input-data buffers is defined as B/f_s, and we will refer to it throughout the paper as the buffering time (t_{buff}). The choice of the parameters B and f_s is critical for the performance of the system because there are two conditions that must be satisfied:

- **The real-time condition**. The application can work in real time if $t_{proc} < t_{buff}$, where t_{proc} is the execution delay measured from the moment the input-data buffer is sent to the GPU until the output-data buffer comes back to the CPU. This includes transfer delays between the CPU and the GPU and the data processing delay of the GPU.

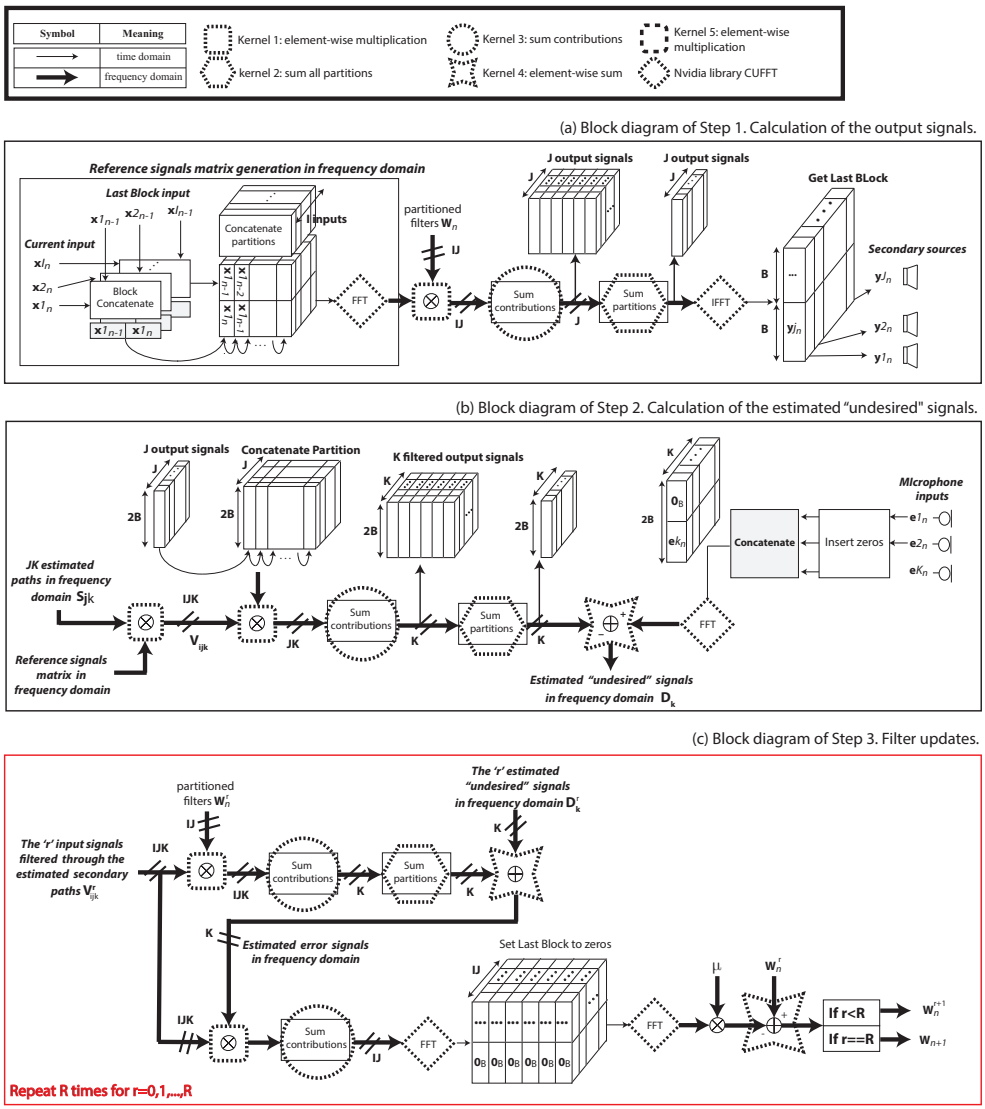

Figure 3. Block diagram of the GPU implementation of the FPBFxLMS and the FPBMFxLMS algorithms.

- **The causality condition**. The algorithm needs to satisfy the condition $t_{buff} + \tau_s < \tau_n$ in order to perform properly [32]. τ_s is the maximum delay of the secondary paths that join the actuators with the error sensors, and τ_n is the minimum delay of the paths that join the noise source with the error sensors. This condition guarantees the causality of the system.

Causality is not a constraint when a sinusoidal excitation is considered because of the deterministic nature of the signal; however, it is important in broadband noise control. In this paper, the multichannel ANC prototype is mounted inside of a listening room where both real-time and causality conditions are fulfilled by choosing the correct distances and parameters. The listening room and some impulse responses in time and frequency domain can be seen in [33]. If the location of the noise source and the environment where the noise has to be canceled do not offer the possibility

to fulfill the causality conditions, the algorithm has to work with a lower block size, and, consequently, with a lower t_{buff} in order to satisfy the causality condition. The audio card offers different values of B between $B = 16$ and 2048. In the experiments, we have used $B = 2048$. Moreover, the audio card offers three different sampling rates: 44.1, 44.8, and 96 kHz. We have chosen $f_s = 44.1$ kHz, which is the lowest rate, but it is a fairly high rate for the common sounds involved in ANC systems. This value of sampling frequency offers us the possibility of controlling sounds up to 22.05 kHz, but at the same, as we have to satisfy the real time condition, it limits the processing time of the algorithm to a few milliseconds.

It should be noted that the signal processing task is carried out by the GPU, while the CPU controls the data transfer between the input/output buffers and the GPU.

4. GPU implementation

This section describes the main issues involved in the GPU implementation of the real-time multichannel ANC prototype based on the FPM-OCF algorithm. The implementation is comprised of three steps that are depicted in Fig. 3: the output signal generation, the estimation of the "undesired" signals, and the update of the adaptive filters. These three steps are implemented as follows:

S1 *Calculation of the output signals*. This step aims to calculate the ANC output signals $y_{[j]}(t)$. The operations of this step correspond to (13). The implementation and the CUDA kernels involved in it are shown in Fig. 3(a).

S2 *Calculation of the estimated "undesired" signals*. This step calculates an estimate of the undesired signals. The corresponding description is shown in (14) and (15). The implementation and the CUDA kernels involved in it are shown in Fig. 3(b).

S3 *Filter updates*. The update of the adaptive filter coefficients involve the implementation of (16)-(20). The details regarding the different steps and kernels are illustrated in Fig. 3(c).

Since when $R = 0$, the FPM-OCF algorithm becomes the FPM algorithm (which is the algorithm presented in [31] with a power normalization), the CUDA Kernels used in this implementation are equivalent to the ones used in [31] but with different input vectors. Therefore, details of the implementation of the Kernels depicted in Fig. 3 can be found in [31].

5. Results

Several experiments were performed to validate the ANC system based on the FPM-OCF algorithm. The experiments were carried out using the prototype described in section 3. Different I:J:K configurations of the ANC system were considered with one reference signal ($I = 1$).

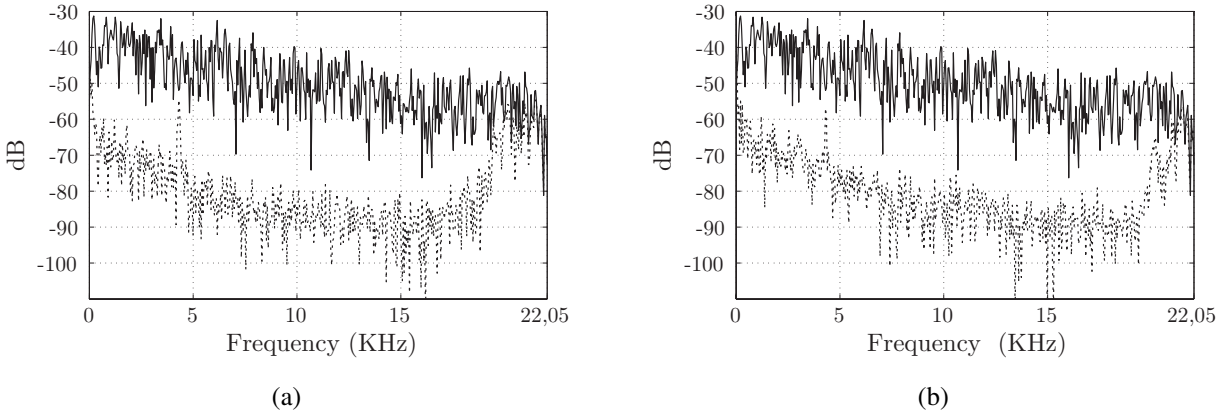

Figure 4. Power spectral density of the average of the signals measured at both error sensors before (solid line) and after (dashed line) the ANC system operation by using the FPM-OCF algorithm in two cases: (a) $R = 0$ (FPM algorithm), and (b) $R = 4$.

The ANC system has been evaluated from different points of view. First, section 5.1 is devoted to validating the ANC performance. For this purpose, the attenuation levels achieved by the ANC system have been obtained from the noise levels measured at the error sensors. Section 5.2 shows an experimental analysis of the convergence speed of the FPM-OCF algorithm varying the value of R. It is important to note that for the case when $R = 0$, the FPM-OCF algorithm becomes the FPM algorithm. Moreover, the computational complexity of the GPU implementation of the algorithm for different R values is also analyzed in section 5.3. Finally, section 5.4 presents the computing results of the GPU implementation of the FPM-OCF algorithm.

With regard to the computing results, a previous analysis of the distribution of threads in a block and blocks in a grid is necessary for each specific case in order to achieve good performance [23]. This previous analysis consists of testing the processing delay of the algorithm by changing both the dimensions of the blocks of threads and the grid of blocks to find the fastest configuration for each different case.

5.1. Residual Noise levels

A multichannel ANC system with dimensions $I = 1$, $J = 2$ and $K = 2$ (1:2:2 configuration) has been considered in this experiment. The following parameters were chosen: $B = 2048$, $L = M = 4096$, $R = 0$, and $R = 4$, with broadband white random noise as reference signal.

Fig. 4 shows the power spectral density of the average signals measured at both error sensors by using the proposed algorithm when $R = 0$ (FPM algorithm) and $R = 4$. The noise reductions showed in Fig. 4 are obtained at the specific points where the error sensors are placed. A similar noise reduction performance is achieved in both cases. It can be easily observed that a reduction of around 25-30 dB can be achieved up to about 20 kHz. We can conclude that the noise reduction in the steady state does not depend on the R value.

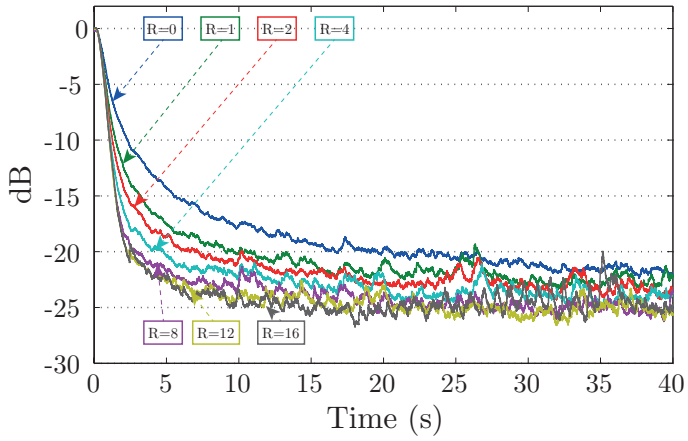

Figure 5. The learning curves of the FPM-OCF algorithm for different values of R and using white noise as the reference signal.

5.2. Convergence performance

The convergence performance of the NLMS-OCF has been theoretically analyzed for system identification configuration in [12]. The theoretical analysis is also validated with simulation results. Specifically, in [12], it has been shown how the NLMS-OCF algorithm outperforms the NLMS algorithm in terms of convergence speed. Moreover, the benefits in terms of convergence speed of the NLMS-OCF algorithm have been also shown when compared with the APA family algorithms in [34]. In this paper, the convergence performance of the FPM-OCF algorithm is evaluated for an ANC application when varying the value of R. As it has been said, when $R = 0$, the FPM-OCF algorithm becomes the FPM algorithm. The FPM solves a mean square problem using a gradient descent algorithm while the FPM-OCF solves a constrained minimization problem. To reach the solution, both algorithms use an iterative procedure. However, the FPM-OCF algorithm uses R input vectors instead of only one. For this reason, the FPM-OCF algorithm converges faster than the FPM, but exhibits a higher computational cost. In order to evaluate the algorithm performance, the learning curves were obtained for the experiments as:

$$A[n] = 10\log_{10}\left(\frac{P_e[n]}{P_d[n]}\right), \tag{21}$$

with

$$\begin{aligned} P_d[n] &= \alpha P_d[n-1] + (1-\alpha)p_d[n], \\ P_e[n] &= \alpha P_e[n-1] + (1-\alpha)p_e[n], \end{aligned} \tag{22}$$

and

$$\begin{aligned} p_d[n] &= \sum_{k=1}^{K} d^2_{[k]}[n], \\ p_e[n] &= \sum_{k=1}^{K} e^2_{[k]}[n]. \end{aligned} \tag{23}$$

Table 3. Processing time and total number of multiplications, additions and FFTs per iteration of the GPU implementation of the FPBMFxLMS-OCF algorithm for different ANC configurations and varying the value of R when $L = M$.

	I:J:K configuration								
	1:1:1			1:2:2			1:4:4		
	R=0	R=4	R=16	R=0	R=4	R=16	R=0	R=4	R=16
Multiplications	$13L$	$33L$	$93L$	$44L$	$112L$	$316L$	$160L$	$420L$	$1200L$
Additions	$10.5L$	$34.5L$	$106.5L$	$37L$	$125L$	$389L$	$140L$	$476L$	$1484L$
FFTs	5	13	37	9	25	73	17	49	145
Time (ms)	0.73	1.91	5.46	1.16	3.22	9.29	2.25	6.64	19.46
M_0/M_N	1	2.5	7.2	1	2.5	7.2	1	2.5	7.5
A_0/A_N	1	3.3	10.1	1	3.4	10.5	1	3.4	10.6
$\mathrm{FFT_0/FFT_N}$	1	2.6	7.4	1	2.8	8.1	1	2.9	8.5
t_0/t_N	1	2.6	7.5	1	2.8	8.0	1	2.9	8.6

with K being the number of error sensors.

Fig. 5 illustrates the simulations results with the configuration used in subsection 5.1. In this figure it is shown that when the value of R grows, the FPM-OCF provides a faster convergence rate. However, there is a value of R where the increase in the speed of convergence saturates. Specifically, the fastest convergence rate is obtained when 12 input vectors are used to update the weights (R=12), and, consequently, a higher value of R does not result in a faster convergence rate. On the other hand, the learning curves show a similar steady-state behavior, which confirms the results of the subsection 5.1. Focusing on the transitory, the algorithm reaches 20 dB of attenuation in approximately 2 seconds when $R = 12$, while the algorithm takes around 17 seconds when $R = 0$, which means that the convergence speed is accelerated 8.5 times. Another important aspect is that a good convergence performance is obtained even with low R values. For example, when the value of R is increased from $R = 0$ to $R = 1$, the convergence time is more or less halved.

5.3. Computational complexity

The results of the subsection 5.2 demonstrate that the FPM-OCF algorithm outperforms the convergence rate of the FPM when $R > 0$. This section is devoted to analyzing the major drawback of the algorithm: its computational

complexity. Table 3 compares the computing time and the computational complexity in terms of multiplications, additions, and FFTs per iteration of the GPU implementation of the FPM-OCF algorithm for different I:J:K configurations and different R values. As before, note that the FPM-OCF algorithm becomes the FPM algorithm when $R = 0$. Since we use a value of $M = L$ and B=$L/2$ (filters are split up into 2 partitions), the computational complexity only depends on L. It is important to note that the computational complexity can be analyzed by changing the I:J:K configuration (and therefore the number of channels) and also by changing the number of input vectors (R) used in the filter coefficients update.

First, Table 3 shows that, for a given value of R, the computational complexity of the algorithm increases significantly with the increase in the number of channels. As an example, when $R = 0$ (FPM algorithm), if the ANC configuration changes from 1:1:1 to 1:4:4 (16 secondary paths), the number of multiplications increases by a factor of 12, the additions by a factor of 13, and the FFTs by a factor of 3.4 while the time delay increases only by a factor of 3. Approximately the same occurs if $R > 0$. For example, for a value of $R = 16$ and changing the arrangement configuration from 1:1:1 to 1:4:4, the number of multiplications increases by a factor of 13, the additions by a factor of 14, and the FFTs by a factor of 3.9 while the time delay increases only by a factor of 3.5. Since the increase in the number of channels greatly increases the complexity, we can conclude that the computational complexity is a bottleneck of massive multichannel ANC systems regardless of the value of R. However, if the operations of each channel are properly parallelized, an implementation over GPU would reduce this computational drawback and could be a viable and meaningful solution.

Table 3 also shows that when orthogonal correction factors ($R > 0$) are used, the FPM algorithm exhibits higher computational complexity than without using them ($R = 0$). This is due to the usage of the R input signal vectors in the tap weight update (see Fig. 3, box in red). As Table 3 shows, the number of multiplications, additions and FFTs significantly increases. Let us define the ratio M_R/M_0 as the number of multiplications of the FPM-OCF algorithm when $R > 0$, divided by the number of multiplications of the FPBM algorithm ($R = 0$). We define the same ratio for additions (A_R/A_0), FFTs ($\mathrm{FFT_R}/\mathrm{FFT_0}$), and time delay ($t_R/t_0$). It is shown that the ratios of multiplications, additions and FFTs are similar to the ratio of delay. This can be explained by the fact that the processing of step 3 in Fig. 3 must be executed in a sequential mode because the coefficients of the adaptive filters $\mathbf{W}ij_{n+1}^{r}$ are needed for the calculation of the filters $\mathbf{W}ij_{n+1}^{r+1}$, and, therefore, it can not be parallelized. However, even though the calculation of $\mathbf{W}ij_{n+1}^{r}$, $\mathbf{W}ij_{n+1}^{r+1}$, ..., $\mathbf{W}ij_{n+1}^{R}$ can not be implemented in parallel, the calculation of each one of the R times that the coefficients are updated at each iteration could be accelerated by being implemented on a parallel computing device such as a GPU.

Finally, it is important to emphasize that, as showed in sections 5.2 and 5.3, both the improvement in convergence rate and the increase in complexity are linked to the number of correction factors. Therefore, this leads to an adjustable and configurable implementation of the FPM-OCF algorithm with regards to the complexity and convergence rate.

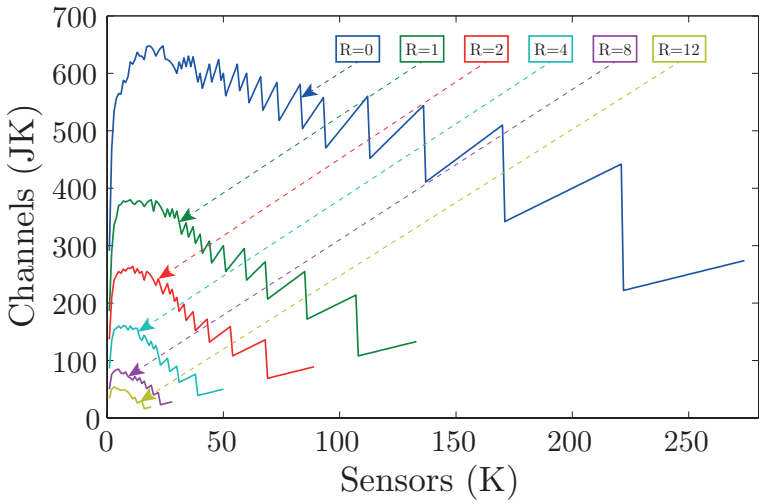

Figure 6. Maximum number of channels (JK) for each K value performing in real-time when I=1 for different values of R, $B = 2048$, and L=M=4096.

5.4. Prototype computing performance

As is well known, the ANC prototype can extend the zone of cancellation by properly adding more sensors and transducers. However, as we have seen in the subsection 5.3, the computational cost can become extremely large, especially when $R > 0$.

This section is devoted to studying the computational constraints of the multichannel ANC prototype based on the FPM -OCF algorithm. For this purpose, Fig. 6 shows the maximum number of channels that the GPU-based ANC system can handle without violating the real-time condition for L=M=4096, $B = 2048$, for one reference signal ($I = 1$) and different values of R. This maximum number of channels is calculated by fixing the number of error sensors and finding the maximum number of actuators (J) that the system can feed without violating the real-time condition. When the maximum J value for each value of K is found, the maximum number of physical channels that are processed for each value of K is $J \cdot K$. For example, for the curve labeled as $R = 0$ in Fig. 6, when $K = 170$, the maximum number of actuators that can be used without violating the real-time condition is $J = 3$; therefore, the maximum number of processed channels is 510. However, for $K = 171$, the maximum number of actuators is $J = 2$ because the real-time condition is violated with $J = 3$. Thus, the maximum number of processed channels is 342 ($J = 2$) when $K = 171$. For this reason, the shapes of the curves jumps with the increase in error sensors.

The following considerations are highlighted in the simulation results depicted in Fig. 6:

- The maximum number of actuators for each value of K is calculated by taking into account that it has to satisfy the real-time condition: $t_{proc} < B/f_s$. Therefore, if R increases, there are more operations to perform in the same time period and thus less channels can be handled.

- Two systems with different I:J:K configurations could have the same number of physical channels but different

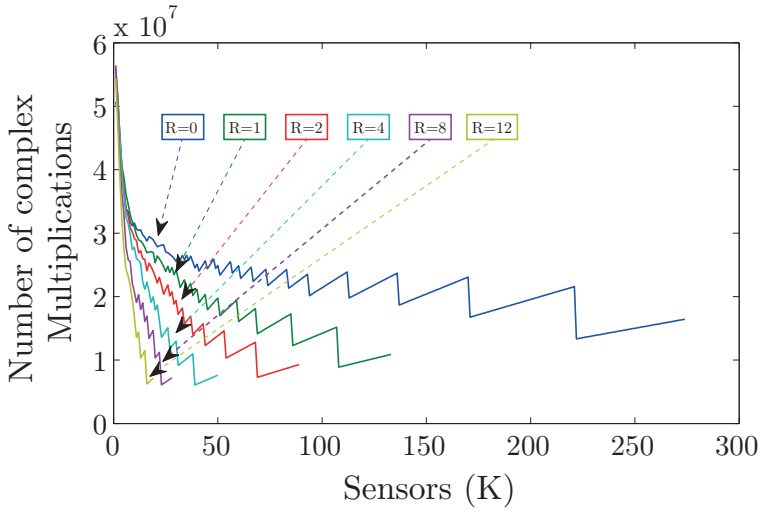

Figure 7. Number of complex multiplications (CM) performed for the maximum JK configuration when I=1 for different values of R, B = 2048, and L=M=4096.

computational costs. For example, both the 1:1:2 and 1:2:1 configurations have 2 physical channels, but the second configuration exhibits a higher computational cost because it has to update two adaptive filters instead of one. Therefore, an increase in the number of adaptive filters involves many more operations than an increase in the number of error sensor signals, with IJ being the number of adaptive filters. Furthermore, when K is low and J is high, the maximum number of channels is limited by the delay of processing the adaptive filters (see Fig. 6 when K is low).

- When K increases, J has to decrease in order to satisfy the real-time condition, and, consequently, the number of adaptive filters decreases and the curves of the number of processed channels grow quickly reaching the maximums. The maximum of the curves is reached when neither K nor J is much bigger than the other.

- On the other hand, when J is small and K is large, even though the number of adaptive filters is low and therefore less operations have to be carried out, the system operation is limited by the handling of the estimated "undesired" signals and the estimated error signals, which involve memory transactions and flow control instructions (see Fig. 3). Therefore, the maximum number of channels decreases with the increase of K. Moreover, since the calculation of the estimated error signal is repeated R times (one for each of the R input vectors used in the tap weight update), when the value of R increases, the decrease in the number of channels with the increase of K is bigger than when $R = 0$.

Once the maximum number of processed channels has been analyzed for each value of K, the number of complex multiplications (CM) involved in both the products and the FFTs for the JK configurations derived in Fig. 6 is depicted in Fig. 7. A CM involves 4 floating point multiplications and 2 floating point additions. It can be observed that for a given value of R, the curve of the number of CM performed is not constant for the different JK configurations. As

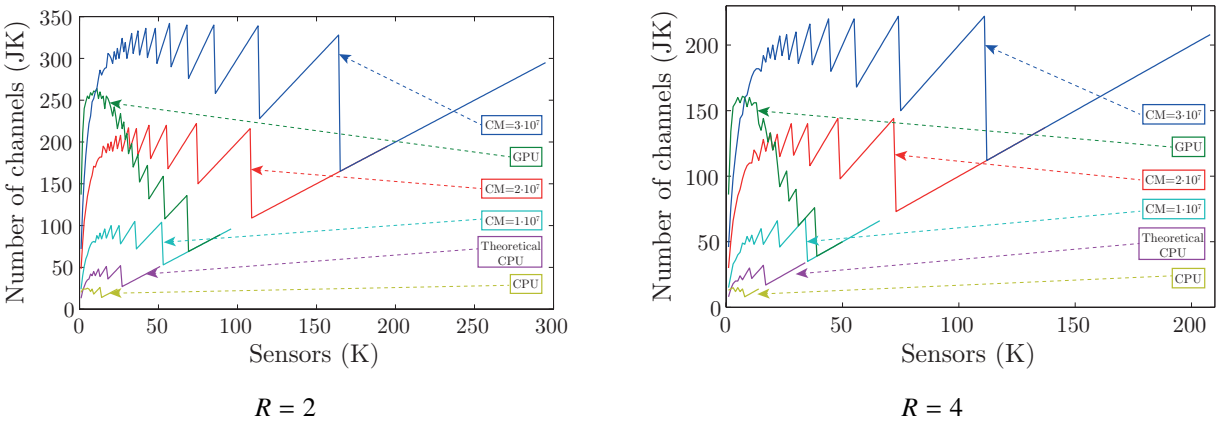

Figure 8. A comparison of the maximum number of processed channels for the GPU implementation, the CPU implementation, and a theoretical processing machine limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ complex multiplications (CM) per block of samples for $B = 2048$ and $I = 1$.

commented above, this is because the GPU implementation is affected by the JK configuration. The most remarkable aspect of Fig. 7 is that the JK configurations with more CM are those with low values of K and high values of J. As explained above, the number of adaptive filters depends on both the I and J variables; therefore, when J grows, more CM are performed because there are more adaptive filters to cope with. Moreover, as the value of K grows, the handling of the estimated "undesired" signals and the estimated error signals limits the processing and therefore less CM are performed. Finally, if we compare the curves with different values of R, it is shown that for low values of K, all of the curves perform a similar number of complex multiplications, but as K increases, the decrease in the number of performed complex multiplications is accentuated with the value of R. This is because the algorithm performs the handling of the K error signals R times.

The maximum number of channels that can be handled in real time by the GPU is shown in Fig. 6 for different values of R. In order to compare the computational capabilities of the GPU with other hardware platforms, an ANC system based on a CPU i7 was also implemented using one core and a sequential execution. The maximum number of channels allowed by the CPU implementation was theoretically and practically studied. Moreover, the GPU and CPU evaluation results were also compared with a theoretical processing machine that was limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ CM per buffering time. The results of this comparison are illustrated in Fig. 8 for the case when $I = 1$, $M = L = 4096$, $B = 2048$, and $R = 2$ (Fig. 8.a) or $R = 4$ (Fig. 8.b). In this case, the buffering time is $B/f_s = 2048/44100 = 0.0464$ seconds. For example, the curve labeled as 'CM=$1 \cdot 10^7$' in Fig. 8 represents the maximum number of channels for each value of K that a given machine would process if this machine was able to carry out $1 \cdot 10^7$ CM every 46.4 milliseconds. Finally, the theoretical maximum performance of our CPU was found by calculating the maximum number of CM that this CPU could handle in a real-time execution. Since a floating point multiplication and a floating point addition are performed by our CPU in 5 and 3 clock cycles, respectively (see annex 3 of [35]), a CM involves 26 clock cycles. The CPU operates at 3.07GHz, which means that our CPU could make

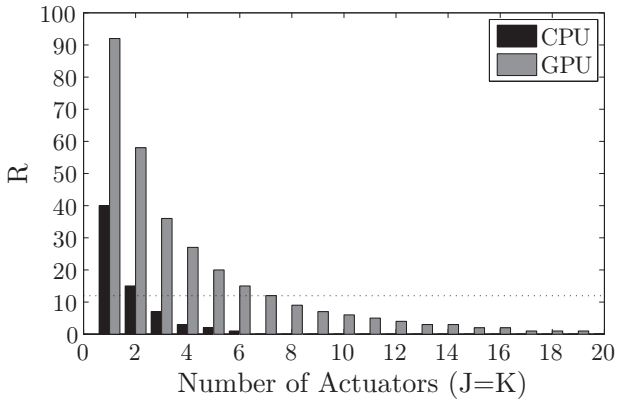

Figure 9. Maximum number of input vectors (R) that can be used in the tap weight update by the CPU and GPU configurations for square I:J:K configurations when $I = 1$, $B = 2048$, and L=M=4096.

$3.07 \cdot 10^9/26$ CM per second and $(3.07 \cdot 10^9/26) \cdot 0.0464 \simeq 5.5 \cdot 10^6$ CM per buffering time. This is represented in the figure with the curve labeled as 'theoretical CPU'. Since our CPU also performs memory transactions and flow control instructions, the practical CPU implementation handles fewer channels than the 'theoretical CPU'. Furthermore, the theoretical processing machine that processes $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ CM per buffering time would has also to perform memory transactions and flow control instructions in addition to the complex multiplications. Therefore, in practice, these three curves would be lower.

Fig. 8 shows that this algorithm can not be used with $R > 0$ for multichannel ANC systems using the i7 CPU as the processor. Therefore, expensive CPUs would be needed. On the other hand, the figure illustrates that the GPU implementation outperforms the CPU implementation for both values of R and also shows the number of CM that a processing machine would have to carry out each buffering time to outperform the GPU implementation. Moreover, the maximum benefit of the GPU is obtained when low values of K and high values of J are used, where the GPU carries out $3 \cdot 10^7$ CM per buffering time. As in previous sections, this can be explained by the fact that an increase in the value of K involves an increase in the time required to handle the estimated undesired signals and the estimated error signals in the frequency domain. This fact is accentuated with the increase in the value of R. As a conclusion, Fig. 8 shows that even though the increase of the value of R results in a high increase in the computational complexity, the algorithm can be implemented in a real-time ANC prototype using a GPU and a proper parallelization of the operations of the multiple channels.

Finally, in order to better clarify the computational limitations of the CPU implementation, Fig. 9 depicts the maximum value of R that can be used by both the CPU implementation and the GPU implementation for different I:J:K configurations. It has been calculated for square I:J:K configurations because they are the most representative. This means configurations with one reference signal (I=1) and the same number of loudspeakers as microphones (J=K). The figure also shows (with a dotted line) the saturation value of R where it saturates; in this case, $R = 12$. It is shown that the CPU implementation can reach the saturation value of R onky for the single channel configuration and the

1:2:2 configuration, whereas the GPU configuration can reach up to the 1:7:7 configuration. Moreover, the maximum channel configuration that can be implemented on a CPU with a value of $R = 1$ is the 1:6:6 configuration, which involves 36 channels processed in real time. In contrast, the GPU implementation reaches the 1:19:19 configuration, which means 361 channels processed in real time.

6. Conclusions

This work presents the M-OCF algorithm for a multichannel active noise control system. The algorithm is derived from the NLMS-OCF algorithm previously introduced for system identification. Moreover, this paper studies the feasibility of its implementation in real time. To do this, the algorithm has been implemented using a GPU. To meet the hardware requirements, the algorithm has been developed in the frequency domain, working with blocks of data and partitioning the adaptive filters, resulting in the FPM-OCF algorithm. As a result, a prototype of multichannel sound-control has been successfully implemented over GPU using the CUDA language and taking advantage of the parallelization of the multiple channels involved. For this purpose, we have used the Nvidia CUDA in order to obtain the most efficient implementation. We have avoided memory copies in the GPU and we have analyzed some CUDA aspects such as the number of threads per block and the distribution of threads within the blocks.

The work in this paper has proved that the convergence rate of the FPM-OCF algorithm improves with the increase in the value of R; therefore, this algorithm converges faster than the FPBFxNLMS algorithm ($R = 0$). Even though the increase in the value of R results in a high increase in the computational complexity (especially for massive multichannel systems), it has been demonstrated that by taking advantage of the parallelization capabilities of the GPU, the increase in computational complexity due to the increase of channels produces a reduced increase in the processing delay. Therefore, the use of a GPU platform can help to overcome the disadvantage of the FPM-OCF in real-time for multichannel ANC systems. Moreover, both the improvement in convergence rate and the increase in computational complexity can be controlled and adjusted with the number of correction factors used in the execution of the algorithm, and, therefore, this value can be set depending on the performance requirements. As a conclusion, we have demonstrated that by using a current cheap GPU (GeForce GTX 580), the implementation of the FPM-OCF algorithm in a real-time multichannel ANC prototype is feasible, and therefore the GPU is a meaningful and versatile solution for massive multichannel ANC systems, that is capable of processing hundreds of channels in real time, depending on the value of R.

Finally, the computational limits of the ANC system have been studied. It has been shown that a commitment relation between the convergence needing (number of correction factors used) and the available computing resources (CPU/GPU) is necessary depending on the size of the multichannel structure (I:J:K configuration). For a single channel structure or small multichannel structures (e.g 1:2:2), the FPM-OCF algorithm can be implemented sequentially in a CPU using the maximum number of correction factors without saturating. For bigger multichannel structures, a parallelization of the operations is needed due to the big computational cost, and therefore, the GPU could be used.

It is also important to note that with a considerable number of R, for example $R = 8$ in the case of section 5.2, the gain in speed of convergence from $R = 8$ to $R = 12$ is not significantly, and depending on the application needing, the value $R = 8$ could be considered as the maximum after saturation, and larger multichannel structures could be obtained with $R = 8$ instead of $R = 12$. As a conclusion, the optimum relation between R and I:J:K depends on the application needing and the available computing resources. Moreover, it is important to note that the results of this paper in terms of number of processed channels could be larger by using a different audio card with lower frequency sampling, decimating, or even using newer GPUs with better computational capacity.

References

[1] P. A. Nelson, S. J. Elliot, Active control of sound, Imperial College Press, New York, 1992.

[2] S. M. Kuo, D. R. Morgan, Active noise control: a tutorial review, Proceedings of the IEEE 87 (1999) 943–973.

[3] S. J. Elliot, P. A. Nelson, Active noise control, IEEE Signal Processing Magazine 10 (4) (1994) 12–35.

[4] S. Haykin, Adaptive filter theory, Prentice Hall, 1986.

[5] S. J. Elliott, I. M. Stothers, P. A. Nelson, A multiple error LMS algorithm and its application to the active control of sound and vibration, IEEE Transactions on Acoustics, Speech and Signal Processing 35 (10) (1987) 1423–1434.

[6] B. Widrow, S. D. Stearns, Adaptive signal processing, Prentice-Hall, Englewood Cliffs, New York, 1985.

[7] k. Ozeki, T. Umeda, An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties, Electronics Communications 67-A (1994) 19–27.

[8] S. L. Gay, J. Benesty, Acoustic signal processing for telecommunication, Springer, 2000.

[9] S. G. Kratzer, D. R. Morgan, The partial-rank algorithm for adaptive beamforming, in: 29th Annual Technical Symposium, International Society for Optics and Photonics, 1986, pp. 9–14.

[10] M. Rupp, A family of adaptive filter algorithms with decorrelating properties, Signal Processing, IEEE Transactions on 46 (3) (1998) 771–775.

[11] S. G. Sankaran, A. Beex, Normalized lms algorithm with orthogonal correction factors, in: Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on, Vol. 2, IEEE, 1997, pp. 1670–1673.

[12] S. G. Sankaran, A. Beex, Convergence behavior of affine projection algorithms, Signal Processing, IEEE Transactions on 48 (4) (2000) 1086–1096.

[13] H.-C. Shin, A. H. Sayed, Mean-square performance of a family of affine projection algorithms, Signal Processing, IEEE Transactions on 52 (1) (2004) 90–102.

[14] L. Savioja, V. Välimäki, J. O. Smith, Audio signal processing using graphics processing units, Journal Audio Eng. Soc 59 (2011) 3–19.

[15] N. Tsingos, W. Jiang, Using programmable graphics hardware for acoustics and audio rendering, Journal Audio Eng. Soc 59 (2011) 628–648.

[16] C. J. Webb, S. Bilbao, Computing room acoustics with CUDA-3D FDTD schemes with boundary losses and viscosity, in: Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), Prague, 2011, pp. 317–320.

[17] J. Vaněk, J. Trmal, J. V. Psutka, J. Psutka, Optimized acoustic likelihoods computation for NVIDIA and ATI/AMD graphics processors, IEEE Transactions on Audio, Speech, and Language Processing 20 (6) (2012) 1818–1828.

[18] R. Mazur, J. O. Jungmann, A. Mertins, On CUDA implementation of a multichannel room impulse response reshaping algorithm based on p-norm optimization, in: Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), Vol. 24, 2011, pp. 305–308.

[19] S. Zhao, S. Ahmed, Y. Liang, K. Rupnow, D. Chen, D. L. Jones, A real-time 3D sound localization system with miniature microphone array for virtual reality, in: Proceedings of the IEEE Conference on Industrial Electronics and Applications (ICIEA), Singapore, 2012, pp. 1853 – 1857.

[20] D. Theodoropoulos, G. Kuzmanov, G. Gaydadjiev, Multi-core platforms for beamforming and wave field synthesis, IEEE Transactions on Multimedia 99 (2010) 235–245.

[21] M. Schneider, F. Schuh, W. Kellerman, The generalized frequency-domain adaptive filtering algorithm implemented on a GPU for large-scale multichannel acoustic echo canceller, in: Proceedings of the TG Conference on Speech Communication, 2012, pp. 1–4.

[22] J. Lorente, A. Gonzalez, M. Ferrer, J. A. Belloch, M. de Diego, G. Piñero, A. Vidal, Active noise control using graphics processing units, in: Proceedings of the 19th International Congress on Sound and Vibration, 2012, pp. 138–146.

[23] J. Lorente, J. A. Belloch, M. Ferrer, A. Gonzalez, M. de Diego, G. Piñero, A. Vidal, Multichannel active noise control system using a GPU accelerator, in: Proceedings of the Inter-Noise conference, New York, 2012, pp. 7768–7780.

[24] S. M. Kuo, D. R. Morgan, Active noise control, algorithms and DSP implementations, John Wiley & Sons, Inc., New York, 1996.

[25] J. Páez Borrallo, M. Garcia Otero, On the implementation of a partitioned block frequency domain adaptive filter (PBFDAF) for long acoustic echo cancellation, Signal Processing 27 (3) (1992) 301–315.

[26] P. Sommen, Partitioned frequency domain adaptive filters, in: Proceedings of the 23rd Asilomar Conference on Signals, Systems and Computers, Vol. 2, 1989, pp. 677–681.

[27] B. Farhang-Boroujeny, Adaptive Filters Theory and Applications, John Wiley & Sons, Inc., New York, 1998.

[28] M. Ferrer, A. Gonzalez, M. de Diego, G. Piñero, Fast affine projection algorithms for filtered-x multichannel active noise control, Audio, Speech, and Language Processing, IEEE Transactions on 16 (8) (2008) 1396–1408.

[29] E. Bjarnason, Active noise cancellation using a modified form of the filtered-x LMS algorithm, in: Proceedings of the 6th European Signal Processing Conference, Vol. 2, 1992, pp. 1053–1056.

[30] G. H. Golub, C. F. Van Loan, Matrix computations, Vol. 1, Johns Hopkins University Press, 1996.

[31] J. Lorente, M. Ferrer, M. de Diego, J. A. Belloch, A. Gonzalez, GPU implementation of a frequency-domain modified filtered-x LMS algorithm for multichannel local active noise control, in: Proceedings of the 52nd Audio Engineering Society International Conference, Guilford, UK, 2013.

[32] R. Burdisso, J. Vipperman, C. Fuller, Causality analysis of feedforward-controlled systems with broadband inputs, The Journal of the Acoustical Society of America 94 (1) (1993) 234–242.

[33] Audio and communications signal processing group (gtac), online at: http://www.gtac.upv.es.

[34] S. G. Sankaran, On ways to improve adaptive filter performance, Ph.d. thesis, Virginia Polytechnic Institute and State University (1999).

[35] Intel 64 and ia-32 architectures optimization reference manual, online at: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf.