Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

The MMFxLMS algorithm for active noise control with on-line secondary path modelling

Paulo A.C. Lopes, José A.B. Gerald, Moisés Piedade

Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento, Instituto Superior Técnico, Universidade de Lisboa, INESC-ID/IST/UL, Rua Alves Redol n. 9, 1000-029 Lisbon, Portugal

A R T I C L E   I N F O

Article history:
Available online xxxx

Keywords:
Active noise control
On-line secondary path modelling
LMS
FxLMS
MFxLMS
Sudden changes

A B S T R A C T

This paper presents an architecture namely the Mirror MFx (MMFx) for adapting adaptive filtering algorithms for Active Noise Control (ANC) with on-line secondary path modelling. The proposed architecture is used in conjunction with the LMS algorithm, resulting in the MMFxLMS algorithm. A time domain analysis of the algorithm is presented, showing that the algorithm converges regardless of secondary path modeling errors. Simulations of the algorithm in different conditions but with the same parameters result in 100% convergence. The algorithm is especially suited to deal with large and sudden changes in the secondary path when the ANC system is in operation. Comparisons with competing algorithms are made, showing that they do not reach the same performance.

1. Introduction

In active noise control (ANC) [1–8] a sound wave (anti-noise) with opposite phase to a noise wave is used to reduce the noise. It works well at low frequencies, acting as a complement to traditional passive techniques. The most used algorithm in ANC is the filtered-x least mean squares (FxLMS) [1,8] algorithm. The modified FxLMS (MFxLMS) algorithm [9] is also common. Both algorithms require a secondary path model, but none is very sensitive to secondary path modelling errors [10,1]. The secondary path model can be obtained before the starting of the ANC system (off-line). But if the secondary path varies significantly while the ANC system in on then on-line modelling is required. There are several of on-line secondary path modelling algorithms. These can be based on the overall modelling algorithm (OMA) [1,2,8] in the simultaneous equations method [11,12] or with auxiliary noise [13–22]. However, most of these algorithms are not suitable for dealing with sudden secondary path changes, but only deal with slow changes. They can become unstable after sudden changes. The proposed algorithm however, is stable even with incorrect secondary path models and deals very well with sudden changes.

The proposed algorithm is represented in Figs. 1 and 2 and in Table 1. Fig. 2 is the detail of the block named OMA in Fig. 1. This corresponds to equations (5) to (8) in Table 1. Vectors are shown in bold face letters, and $\mathbf{x}^T$ represents the transpose of vector $\mathbf{x}$. The vectors $\mathbf{x}(n)$, $\mathbf{x}'(n)$ and $\mathbf{y}(n)$ are formed by past samples of the

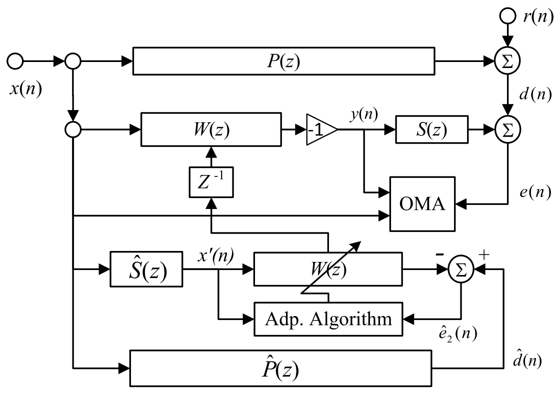

Fig. 1. The MMFxLMS algorithm.

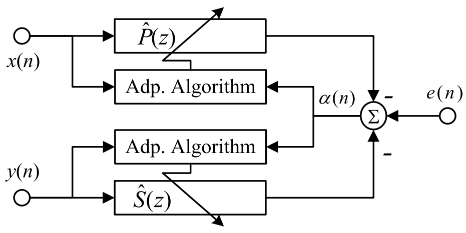

Fig. 2. The overall modelling algorithm (OMA).

E-mail address: paulo.lopes@tecnico.ulisboa.pt (P.A.C. Lopes).

http://dx.doi.org/10.1016/j.dsp.2016.08.012
1051-2004/© 2016 Elsevier Inc. All rights reserved.

Table 1
The MMFxLMS algorithm.

$$\hat{\mathbf{s}}(0) = [1, 0 \dots 0]^{\mathrm{T}} \quad (1)$$
$$\hat{\mathbf{p}}(0) = \mathbf{w}(0) = 0 \quad (2)$$
$$\mathbf{x}(0), \mathbf{x}'(0), \mathbf{y}(0) = 1 \quad (3)$$

$$y(n) = -\mathbf{x}^{\mathrm{T}}(n)\mathbf{w}(n) \quad (4)$$
$$\hat{e}_1(n) = \hat{\mathbf{p}}^{\mathrm{T}}(n)\mathbf{x}(n) + \hat{\mathbf{s}}^{\mathrm{T}}(n)\mathbf{y}(n) \quad (5)$$
$$\alpha(n) = e(n) - \hat{e}_1(n) \quad (6)$$
$$\hat{\mathbf{p}}(n+1) = \hat{\mathbf{p}}(n) + \frac{\mu_{sp}\mathbf{x}(n)\alpha(n)}{\mathbf{x}^{\mathrm{T}}(n)\mathbf{x}(n) + \mathbf{y}^{\mathrm{T}}(n)\mathbf{y}(n)} \quad (7)$$
$$\hat{\mathbf{s}}(n+1) = \hat{\mathbf{s}}(n) + \frac{\mu_{sp}\mathbf{y}(n)\alpha(n)}{\mathbf{x}^{\mathrm{T}}(n)\mathbf{x}(n) + \mathbf{y}^{\mathrm{T}}(n)\mathbf{y}(n)} \quad (8)$$
$$x'(n) = \hat{\mathbf{s}}^{\mathrm{T}}(n)\mathbf{x}(n) \quad (9)$$
$$\hat{d}(n) = \hat{\mathbf{p}}^{\mathrm{T}}(n)\mathbf{x}(n) \quad (10)$$
$$\hat{e}_2(n) = \hat{d}(n) - \mathbf{w}^{\mathrm{T}}(n)\mathbf{x}'(n) \quad (11)$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu\mathbf{x}'(n)\hat{e}_2(n)}{\mathbf{x}'^{\mathrm{T}}(n)\mathbf{x}'(n)} \quad (12)$$

corresponding scalar signals, as in $\mathbf{x}(n) = [x(n) \dots x(n-N+1)]^{\mathrm{T}}$. These vectors should be carefully initialized.

The idea is to model the primary and secondary paths using the OMA, and use the model of the primary path to estimate the desired signal, $d(n)$. This is the desired output of the concatenation of the secondary path, $S(z)$, and controller filter, $W(z)$. Then the controller and secondary path filters are exchanged resulting in the algorithm as presented. This can also be derived by creating a mirror copy of the acoustics paths using the obtained models, and then using the MFxLMS algorithm. Due to the mirror operation the algorithm is called Mirror MFxLMS or MMFxLMS. The adaptive algorithm used is the NLMS [23] as presented in Table 1. Note that in the proposed algorithm there is no need for prior knowledge of the primary and secondary paths.

2. State of the art

In [13] Eriksson proposes the original on-line secondary path modelling technique using auxiliary noise. In [24] a comparison of the overall modelling technique, with the auxiliary noise technique, is presented. The primary noise is removed from the modelling of the secondary path using an auxiliary filter in [14]. [25] also reduces the disturbance in the modelling of the secondary path by removing the delayed residual error signal. [15] extends [14] by removing the additive noise signal from the adaptation of the auxiliary filter. [16] introduces auxiliary noise power scheduling and imposes a constraint on the norms of the adaptive filters. [17] compares [15] with [14]. [18] uses a variable step size that increases with the convergence of the ANC system and in [19] a convergence measure is used to control the auxiliary noise power. [20] makes the ratio of the auxiliary noise at the error microphone to the residual noise a constant, and uses optimum values for the step size of the controller and secondary path filters. [21] proposes two algorithms, an on–off algorithm that turns the additive noise off when the current estimate performs worse than the best so far, and on when the noise level increases; and an auxiliary noise power scheduling algorithm where the auxiliary noise signal level is proportional to the power of the residual noise signal. [22] proposes a two-stage auxiliary noise power scheduling algorithm. At stage one the auxiliary noise signal level is similar to [20] and in stage two it is proportional to the square of the residual noise signal power. In [26] the auxiliary noise level is similar to [20], but the ratio is variable. The ratio is greater when noise reduction is high and low when noise reduction is lower. This allows for faster convergence, resulting in better performance when dealing with sudden changes in the secondary path. The proposed algorithm does not use auxiliary noise, and is based on the overall modelling technique.

3. Time domain analysis

Take the algorithm in Table 1. Most of the signals depend on n, so we dropped the dependence on n on some signals, in order to simplify the notation. Let,

$$\mathbf{A}(n) = \begin{pmatrix} \mu'_{sp}\mathbf{x}\mathbf{x}^{\mathrm{T}} & \mu'_{sp}\mathbf{x}\mathbf{y}^{\mathrm{T}} & 0 \\ \mu'_{sp}\mathbf{y}\mathbf{x}^{\mathrm{T}} & \mu'_{sp}\mathbf{y}\mathbf{y}^{\mathrm{T}} & 0 \\ 0 & 0 & \mu'\mathbf{x}'\mathbf{x}'^{\mathrm{T}} \end{pmatrix} \quad (13)$$

where $\mu'_{sp} = \mu_{sp}/(\mathbf{x}^{\mathrm{T}}(n)\mathbf{x}(n) + \mathbf{y}^{\mathrm{T}}(n)\mathbf{y}(n))$ and $\mu' = \mu/(\mathbf{x}'^{\mathrm{T}}(n)\mathbf{x}'(n))$. This is a block diagonal matrix, formed by two blocks, $\mathbf{A}_1(n)$ and $\mathbf{A}_2(n)$, with $\mathbf{A}_2(n) = \mu'\mathbf{x}'\mathbf{x}'^{\mathrm{T}}$ and

$$\mathbf{A}_1(n) = \begin{bmatrix} \mu'_{sp}\mathbf{I} & 0 \\ 0 & \mu'_{sp}\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} [\mathbf{x}^{\mathrm{T}}\mathbf{y}^{\mathrm{T}}]. \quad (14)$$

And let,

$$\boldsymbol{\pi}(n) = [\hat{\mathbf{p}}^{\mathrm{T}}(n), \hat{\mathbf{s}}^{\mathrm{T}}(n), \mathbf{w}^{\mathrm{T}}(n)]^{\mathrm{T}} \quad (15)$$
$$\boldsymbol{\pi}_o(n) = [\mathbf{p}^{\mathrm{T}}(n), \mathbf{s}^{\mathrm{T}}(n), \mathbf{w}_o^{\mathrm{T}}(n)]^{\mathrm{T}} \quad (16)$$
$$\mathbf{r}(n) = [\mu'_{sp}\mathbf{x}^{\mathrm{T}}(n)r(n), \mu'_{sp}\mathbf{y}^{\mathrm{T}}(n)r(n), 0]^{\mathrm{T}}. \quad (17)$$

The proposed algorithm can be written as,

$$\boldsymbol{\pi}(n+1) = (\mathbf{I} - \mathbf{A}(n))\boldsymbol{\pi}(n) + \mathbf{A}(n)\boldsymbol{\pi}_o(n) + \mathbf{r}(n), \quad (18)$$

where $\mathbf{w}_o(n)$ was defined so that,

$$\mathbf{x}^{\mathrm{T}}(n)\hat{\mathbf{p}}(n) = \mathbf{x}'^{\mathrm{T}}(n)\mathbf{w}_o(n). \quad (19)$$

Taking the z-transform, this corresponds to having $\hat{S}(z)W_o(z) \approx \hat{P}(z)$. Note, however, that in general $\mathbf{w}_o(n)$ will be time varying. This will degrade the performance of the algorithm, because it makes $\boldsymbol{\pi}_o(n)$ also time varying. Defining $\Delta\boldsymbol{\pi}(n) = \boldsymbol{\pi}(n) - \boldsymbol{\pi}_o(n)$ and $d\boldsymbol{\pi}_o(n+1) = \boldsymbol{\pi}_o(n+1) - \boldsymbol{\pi}_o(n)$ we can rewrite (18) as,

$$\Delta\boldsymbol{\pi}(n+1) = (\mathbf{I} - \mathbf{A}(n))\Delta\boldsymbol{\pi}(n) - d\boldsymbol{\pi}_o(n+1) + \mathbf{r}(n) \quad (20)$$

As long as the absolute value of eigenvalues of the matrix $\mathbf{I} - \mathbf{A}(n)$ are less than one, $\boldsymbol{\pi}(n+1)$ will converge to $\boldsymbol{\pi}_o(n)$, as soon as this stabilizes so that $d\boldsymbol{\pi}_o(n+1)$ is zero, apart from a noise term due to $r(n)$. This is equivalent to having the eigenvalues of $\mathbf{A}$ between zero and two. Note, that the eigenvalues of $\mathbf{A}$ are equal to the union of the eigenvalues of $\mathbf{A}_1$ and $\mathbf{A}_2$. Both matrices have characteristic one. The eigenvalues of $\mathbf{A}_1$ are $\lambda = \mu'_{sp}\mathbf{x}^{\mathrm{T}}\mathbf{x} + \mu'_{sp}\mathbf{y}^{\mathrm{T}}\mathbf{y}$, corresponding to eigenvector $u = [\mu'_{sp}\mathbf{x}^{\mathrm{T}}, \mu'_{sp}\mathbf{y}^{\mathrm{T}}]^{\mathrm{T}}$, and $\lambda = 0$. The eigenvalues of $\mathbf{A}_2$ are $\lambda = \mu'\mathbf{x}'^{\mathrm{T}}\mathbf{x}'$, corresponding to eigenvector $u = \mathbf{x}'^{\mathrm{T}}$, and $\lambda = 0$. So for convergence one should have $\mu'_{sp}(\mathbf{x}^{\mathrm{T}}\mathbf{x} + \mathbf{y}^{\mathrm{T}}\mathbf{y}) < 2$ or $\mu_{sp} < 2$; and $\mu' < 2/(\mathbf{x}'^{\mathrm{T}}\mathbf{x}')$ or $\mu < 2$. The zero eigenvalues require special attention. This will result in eigenvalues of $\mathbf{I} - \mathbf{A}(n)$ equal to one. So one gets decreasing modes and constant modes at each iteration of the algorithm. Still, as long as there are persistent excitation signals, the decreasing modes will spread through all the modes of the state, resulting in global convergence. However, if some modes are not excited then they will get stuck on their initial values. Note that for constant modes $\mathbf{r}(n)$ is zero, as shown in the following.

Since $\mathbf{A}$ is a block diagonal matrix, then (20) can be split into two equations, one related to the update of $\hat{\mathbf{p}}$ and $\hat{\mathbf{s}}$ and the other