# A LOW-COST IMPLEMENTATION STRATEGY FOR COMBINATIONS OF ADAPTIVE FILTERS

*Vítor H. Nascimento*[†], *Magno T. M. Silva*[†], *and Jerónimo Arenas-García*[‡]

[†] University of São Paulo, Brazil
{vitor,magno}@lps.usp.br

[‡] Univ. Carlos III de Madrid, Spain
jarenas@tsc.uc3m.es

## ABSTRACT

Combinations of adaptive filters have attracted attention as a simple way to improve filter performance, avoiding the compromise between fast adaptation and low excess mean-square error. However, the computational cost of the combined scheme is higher than that of a single filter, since two or more filters must be run in parallel. In this paper, we propose a new approach for reducing the computational cost of combinations of adaptive filters in custom and semi-custom hardware implementations. In the proposed scheme, the fast filter is adapted as usual, but instead of updating the slow filter, we adapt the difference between the slow and fast filters. We show that the coefficients of the difference filter have a lower dynamic range compared to the slow filter, and therefore can be updated using a smaller number of bits. The wordlength used for the difference filter can be chosen to make the performance of this low-cost scheme similar to that of an infinite-precision implementation.

***Index Terms***— Adaptive filters, convex combination, low-cost implementation.

## 1. INTRODUCTION

Cooperative estimation strategies have been applied to improve the performance of adaptive filters in several applications [1–10]. In particular, convex combinations of adaptive filters have proved to be a reliable and robust way of dealing with the problem of parameter adjustment, since they achieve performance close to that of the best filter in the combination, even taking into consideration the on-line estimation of the combination parameter [1, 11, 12].

One drawback of these schemes, however, is the increased computational complexity — since two or more filters must be run in parallel, in most schemes the computational complexity is roughly double that of a single filter. Some attempts to reduce this problem have focused in using filters of different lengths [13], or using variable-length filters [2].

In this paper we propose a different approach for reducing the complexity of convex combinations of adaptive filters in custom or semi-custom hardware implementations, such as field-programmable gate arrays (FPGAs). We consider the case in which the combination is used to avoid the compromise between fast adaptation and low excess mean-square error, by combining a fast with a slow filter [1]. The main idea is to adapt the fast filter as usual, but instead of directly adapting the slow filter, to estimate the *difference* between the slow and fast filters. Since the dynamical range for the coefficients in the difference filter will usually be smaller than the dynamic range for the original filters, its coefficients can be encoded with a smaller number of bits. Taking advantage of this reduced wordlength, several arithmetic operations can be implemented at a smaller cost. We also derive an expression relating the steady-state performance of the resulting algorithm to the number of bits used to represent the difference filter.

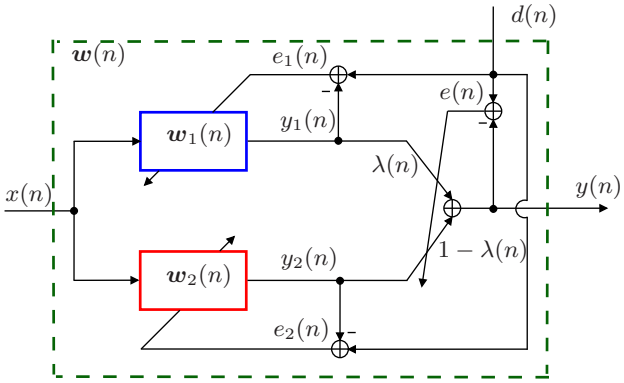


**Fig. 1**. Convex combination of two transversal adaptive filters.

This paper is organized as follows. In the next section, we review convex combinations of a fast and a slow adaptive filter. Next, we review the complexity of fixed-point arithmetic operations as a function of the number of bits allocated to each variable. Following this, we re-derive the convex combination scheme in terms of the fast filter and a difference filter, and show how the difference filter can be encoded using lower-precision variables.

## 2. CONVEX COMBINATION OF TWO LMS FILTERS

For simplicity of presentation, in this paper we consider the convex combination of two real-valued least-mean squares (LMS) filters, but the method can be extended to other algorithms or combinations as well. Consider the case shown in Figure 1, of two independent LMS filters running in parallel with the same inputs and update equations [14, 15]

$$\boldsymbol{w}_i(n+1) = \boldsymbol{w}_i(n) + \mu_i e_i(n)\boldsymbol{x}(n), \quad i = 1, 2, \tag{1}$$

where $\mu_1 > \mu_2$ are the step-sizes, $e_i(n) = d(n) - \boldsymbol{w}_i^T(n)\boldsymbol{x}(n)$ is the error, $(\cdot)^T$ denotes transposition, $d(n)$ is the common scalar desired sequence, and $\boldsymbol{x}(n) = \left[\, x(n)\ x(n-1)\ \ldots\ x(n-M+1) \,\right]^T$ is the common regressor vector. In the convex combination scheme, the overall output is given by

$$y(n) = \lambda(n)y_1(n) + [1-\lambda(n)]\, y_2(n) \tag{2}$$

The work of Nascimento and Silva was partly supported by CNPq under Grants 303921/2010-2 and 302423/2011-7, and FAPESP under Grant 2011/06994-2. The work of Arenas-García was partly supported by MICIAN projects TEC2011-22480 and PRI-PIBIN-2011-1266.

where $y_i(n) = \boldsymbol{w}_i^T(n)\boldsymbol{x}(n)$ are the component filter outputs, the overall error is $e(n) = d(n) - y(n)$, and $0 \leq \lambda(n) \leq 1$ is the mixing parameter, which is updated through an auxiliary variable $a(n)$ as follows [16]. Define the sigmoid function $\mathrm{sgm}(a) \triangleq \left[1 + \mathrm{e}^{-a}\right]^{-1}$

$$\lambda(n) = \frac{\mathrm{sgm}[a(n)] - \mathrm{sgm}[-a_+]}{\mathrm{sgm}[a_+] - \mathrm{sgm}[a_-]}, \tag{3}$$

$$a(n+1) = a(n) + \frac{\mu_a e(n)}{p(n) + \epsilon}\left[y_1(n) - y_2(n)\right]\frac{\partial\lambda(n)}{\partial a(n)}, \tag{4}$$

$$\frac{\partial\lambda(n)}{\partial a(n)} = \frac{\mathrm{sgm}[a(n)]\{1 - \mathrm{sgm}[a(n)]\}}{\mathrm{sgm}[a_+] - \mathrm{sgm}[a_-]}, \tag{5}$$

$$p(n+1) = \eta p(n) + (1-\eta)\left[y_1(n) - y_2(n)\right]^2, \tag{6}$$

where $p(n)$ is an estimate of the power of $y_1(n) - y_2(n)$, used to normalize the adaptation of $a(n)$ [12, 17], $0 \ll \eta < 1$ is the forgetting factor, and $\mu_a$ is the step-size. The definition of $\lambda(n)$ as in (3), proposed in [16], is a simple way to avoid the update of $a(n)$ freezing when $\lambda(n)$ gets close to 0 or 1, due to the zeros in $\partial\lambda(n)/\partial a(n)$. The value of $a_+ = -a_-$ is usually chosen to be 4.

When implemented as in (1)–(6), the combination requires 6 multiplications, 7 additions and 1 division for the evaluation of (2), (4), (6), as well as two searches in lookup-tables (LUTs) for (3) and (5), plus twice the number of operations required for each component filter. In the case of LMS component filters, this amounts to $2M + 1$ multiplications and $2M$ additions for each filter. The total number of operations in the case of LMS component filters is thus $4M + 8$ multiplications, $4M + 7$ additions, 1 division and two lookup-table searches per iteration. Our goal in this paper is to reduce this complexity, taking advantage of finite-precision properties of actual implementations of the method. We begin by reviewing, in the next section, the computational cost of fixed-point arithmetic operations, as a function of the wordlength of the inputs.

## 3. COMPUTATIONAL COST OF FIXED-POINT OPERATIONS

Consider first the case of addition of a variable $u$, represented in fixed-point using $B_u$ bits, to a variable $v$, represented using $B_v$ bits. The exact cost will of course depend on all the details of the hardware implementation [18] — we consider here the most general features and, for ease of exposition, only positive numbers when comparing the complexities of addition and multiplication. The complexity in the case of negative numbers is similar. Let then $u$ be represented in fixed-point by a binary word $b_1^u b_2^u \ldots b_{B_u}^u$ (in general, there would be an extra bit for the sign), and similarly for $v$, such that

$$u = \sum_{i=1}^{B_u} b_i^u 2^{-k_u - i}, \qquad v = \sum_{i=1}^{B_v} b_i^v 2^{-k_v - i},$$

where $k_u$ and $k_v$ are fixed exponents that may differ for $u$ and $v$.

Consider first the operation $u+v$, for two positive numbers with $B_u = B_v$ and $k_u = k_v$ (as in a fixed-point DSP), each pair of bits must be added, taking the carries into account. Let us denote the complexity of carrying on this operation as $\mathcal{A}(B_u)$.

Consider now the case of different wordlengths, for example, $B_v \leq B_u$ and $k_u = 0$, and $0 \leq k_v \leq B_u - B_v$. If dedicated hardware is used to compute $u+v$, the two variables must be aligned as in Figure 2. Variable $v$ must be completed with zeros, as shown in the figure, before the operations are performed. However, because of the carries, the final complexity for computing $u + v$ will in general be comparable to $\mathcal{A}(B_u)$.

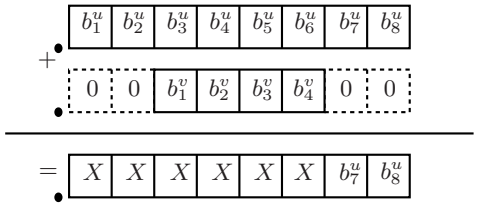


**Fig. 2**. Addition of two fixed-point numbers with different word lengths. In this example, $B_u = 8$, $k_u = 0$, $B_v = 4$, $k_v = 2$. $X$ denotes undetermined values in the result.

The situation is different for multiplications. Consider the multiplication of $u$ and $v$, defined as before, as shown in Figure 3[1]. The final result is the addition of $B_v$ shifted versions of $u$. The result will have $B_u + B_v$ bits, and the complexity will be of the order of $\mathcal{M}(B_u, B_v) = (B_v - 1)\mathcal{A}(B_u + B_v)$.

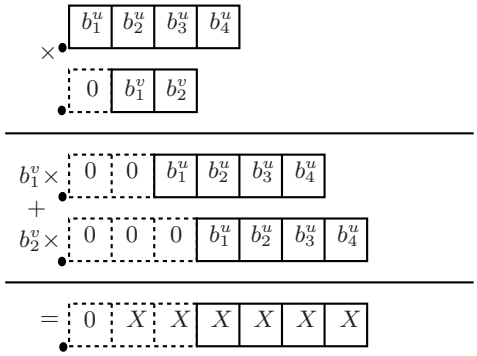


**Fig. 3**. Multiplication of two fixed-point numbers with different wordlengths. $X$ denotes undetermined values in the result. In this example, $B_u = 4$, $k_u = 0$, $B_v = 2$, $k_v = 1$.

Note that, if $B_v < B_u$, the complexity for computing $u + v$ will not be significantly smaller from the complexity of computing the sum of two variables with $B_u$ bits (that is, $\mathcal{A}(B_u)$). However, the cost for computing the product of two $B_u$-bit variables is $(B_u - 1)\mathcal{A}(2B_u)$, which might be much larger than the cost of computing the product $uv$ if $B_v \ll B_u$.

Divisions are considerably more costly than multiplications, but the single division necessary for normalization in (4) does not need to be implemented with high precision, and could be implemented adequately using a few iterations of the Dichotomous Coordinate Descent (DCD) algorithm as proposed in [19, 20].

Looking back to the number of operations required for implementing the convex combination algorithm, we see that the largest cost is in implementing the $4M + 8$ multiplications. In the next section, we use this observation to develop a low-cost algorithm for implementing convex combinations of adaptive filters, by reducing the cost of the multiplications required by the second filter.

## 4. DIFFERENCE FILTER

Our goal in this section is to rewrite the recursions (1)–(6) in terms not of the fast ($\boldsymbol{w}_1(n)$) and slow ($\boldsymbol{w}_2(n)$) filters, but in terms of the

[1] Other multiplication algorithms are also available [18].

fast filter and the *difference* filter

$$\boldsymbol{\delta w}(n) \triangleq \boldsymbol{w}_2(n) - \boldsymbol{w}_1(n). \tag{7}$$

The rationale is that, since (at least in steady-state) $\boldsymbol{w}_1(n)$ and $\boldsymbol{w}_2(n)$ are close to each other, we could use a smaller number of bits to represent each coefficient of $\boldsymbol{\delta w}(n)$. In this way, all multiplications involving $\boldsymbol{\delta w}(n)$ would be performed at a smaller cost, and since multiplications represent the larger share of the cost in custom and semi-custom hardware implementations (compare $\mathcal{A}(B_u)$ with $\mathcal{M}(B_u, B_v)$, see also [18, 20, 21]), the final complexity would be considerably reduced.

Let us then derive a recursion for $\boldsymbol{\delta w}(n)$. Subtracting (1) for $i = 2$ from itself for $i = 1$, we obtain

$$\boldsymbol{\delta w}(n+1) = \boldsymbol{\delta w}(n) + e_\delta(n)\boldsymbol{x}(n), \tag{8}$$

where

$$e_\delta(n) \triangleq \mu_2 e_2(n) - \mu_1 e_1(n). \tag{9}$$

The errors $e_\delta(n)$ and $e_2(n)$ should be computed using the lower precision, and $e_2(n)$ should be computed as

$$e_2(n) = e_1(n) + \boldsymbol{\delta w}^T(n)\boldsymbol{x}(n). \tag{10}$$

If $e_\delta(n)$ and $\boldsymbol{\delta w}(n)$ require $B_\delta$ bits for storage and the other variables $B$ bits, the total complexity of the combination reduces to

$$\begin{aligned} C = {} & (2M+7)\mathcal{M}(B,B) + (2M+7)\mathcal{A}(B) + 2M\mathcal{A}(B_\delta) \\ & + (2M+1)\mathcal{M}(B,B_\delta) + 1 \text{ div.} + 2 \text{ LUTs.} \end{aligned} \tag{11}$$

We must now compare the dynamic range of $\boldsymbol{\delta w}(n)$ with that of $\boldsymbol{w}_1(n)$ and $\boldsymbol{w}_2(n)$. Assume that the data $d(n), \boldsymbol{x}(n)$ satisfy a linear model [14, 15]

$$d(n) = \boldsymbol{w}_\text{o}^T\boldsymbol{x}(n) + v(n), \tag{12}$$

where $\boldsymbol{w}_\text{o} \in \mathbb{R}^M$ is an unknown vector, and $v(n)$ is white noise, independent of $\boldsymbol{x}(n)$ and with variance $\sigma_0^2$.

Define the weight-error vectors for each filter $\tilde{\boldsymbol{w}}_i(n) = \boldsymbol{w}_\text{o} - \boldsymbol{w}_i(n)$. Assuming that all computations are made in fixed-point with $B$ bits and fixed exponent $k_b$, in steady-state and for small step-sizes, it can be shown that $\tilde{\boldsymbol{w}}_i(n)$ is a zero-mean random vector with autocovariance matrix [15]

$$\boldsymbol{K}_i \triangleq \lim_{n\to\infty} \mathrm{E}\{\tilde{\boldsymbol{w}}_i(n)\tilde{\boldsymbol{w}}_i^T(n)\} \approx \frac{1}{2}\mu_i(\sigma_0^2+\sigma_b^2)\boldsymbol{I} + \frac{\sigma_b^2}{2\mu_i}\boldsymbol{R}^{-1}, \tag{13}$$

where $\sigma_b^2 = 2^{-2(B-k_b)}/12$, $\boldsymbol{I}$ is the $M \times M$ identity matrix, and $\boldsymbol{R} = \mathrm{E}\{\boldsymbol{x}(n)\boldsymbol{x}^T(n)\}$ is the regressor autocovariance matrix.

Let $\xi_i = \max\{[K_i]_{jj}\}$, the maximum diagonal entry of $\boldsymbol{K}_i$. Assume that $B$ was chosen so that a certain desired performance is obtained for the slow filter. Note that $B$ may be specified through $\xi_2$, or the mean-square deviation $\text{MSD}_2 = \mathrm{Tr}(\boldsymbol{K}_2)$, where $\mathrm{Tr}(\cdot)$ represents the trace of its argument, or in terms of the excess mean-square error (EMSE) $\zeta_2$ [15],

$$\zeta_2 = \mathrm{Tr}(\boldsymbol{R}\boldsymbol{K}_2) = \frac{1}{2}\mu_2(\sigma_0^2+\sigma_b^2)\,\mathrm{Tr}(\boldsymbol{R}) + \frac{M\sigma_b^2}{2\mu_2}. \tag{14}$$

If the largest variance of the entries of $\tilde{\boldsymbol{w}}_1(n)$ is $\xi_1$, we can expect the $B_1$ most significant bits of each entry of $\boldsymbol{w}_1(n)$ to remain approximately constant at their means (or equivalently, the $B_1$ most significant bits of each entry of $\tilde{\boldsymbol{w}}_1(n)$ remain approximately equal to zero). Comparing $\xi_1$ with the variance for simply rounding a variable with $B_1$ bits ($2^{-2B_1}/12$), we can approximate

$$B_1 \approx -0.5\log_2(12\xi_1). \tag{15}$$

If fixed point is used, the fixed exponent used for representing $\boldsymbol{\delta w}(n)$ and $e_\delta(n)$ should be $k_\delta \approx k_b - \lfloor B_1 \rfloor$, where $\lfloor B_1 \rfloor$ represents the largest integer smaller than $B_1$ and $k_b$ is the exponent used for the other variables. Note that $k_\delta$ can be reduced if the probability of overflow is small.

Now, since the desired performance for the slow filter requires a number $B$ of bits, the difference filter should have approximately

$$B_\delta \approx B - (k_b - k_\delta) \text{ bits.} \tag{16}$$

This means that the number of bits required to represent the entries of $\boldsymbol{\delta w}(n)$ need not be $B$, but rather $B_\delta < B$. This reduction leads to a reduced complexity, since multiplications can be performed at a lower cost. The error $e_\delta(n)$, defined in (9), can also be computed with the smaller number of bits.

Although we concentrated our discussion on fixed-point implementations, the scheme proposed here can also be used with floating-point implementations, or even with mixed implementations in which only $\boldsymbol{\delta w}(n)$ and $e_\delta(n)$ are implemented using floating point. In full fixed-point implementations the number of bits required for $\boldsymbol{\delta w}(n)$ and $e_\delta(n)$ will be larger than for floating point for a given performance level, to reduce the probability of overflow. This will be illustrated in Section 5.

## 5. SIMULATIONS

In this section we illustrate the performance of the proposed low-complexity scheme in a few examples. The filters estimate an $M = 7$ weight vector, initially equal to

$$\boldsymbol{w}_\text{o} = \begin{bmatrix} 0.9 & -0.5 & 0.2 & -0.1 & 0.8 & 0.5 & -0.1 \end{bmatrix}^T,$$

but changed to

$$\boldsymbol{w}_\text{o} = \begin{bmatrix} 0.2 & -0.5 & -0.4 & 0 & -0.2 & 0.8 & 0.2 \end{bmatrix}^T$$

after $50,000$ iterations. The regressor is a tap-delay line $\boldsymbol{x}(n) = \begin{bmatrix} x(n) & \dots & x(n-M+1) \end{bmatrix}^T$ in which $x(n)$ has variance $\sigma_x^2 = 0.01$ and is generated through

$$x(n+1) = \alpha x(n) + \sigma_x\sqrt{1-\alpha^2}u(n),$$

with $\alpha = 0.5$ and $u(n)$ is zero-mean white noise with unit variance. In all simulations, we consider the convex combination of two LMS filters with step-sizes $\mu_1 = 0.5$, $\mu_2 = 0.05$, and $\mu_a = 0.1$. We also set $\eta = 0.9$, $\epsilon = 0$, and $\sigma_0^2 = 0.01$.

For comparison, we show in Figure 4 the EMSE of the component filters and of their combination along the iterations, assuming the Matlab precision (floating point with 64 bits: 11 bits for the exponent, 52 bits for the mantissa, and one bit for the sign). As expected, at each time instant the convex combination follows the component filter that achieves the lower EMSE, behavior that can be confirmed through the mixing parameter, also shown in the figure. We should notice that with this arithmetic precision, the steady-state EMSE for the combination, given by (14), is $\zeta_2 = -47.6$ dB. As can be observed in the figure, this value was achieved by the slow filter and also by the combination at the steady-state.

Now, we use a fully fixed-point implementation with $B = 14$ bits for the fast filter and common variables, and different values $B_\delta$

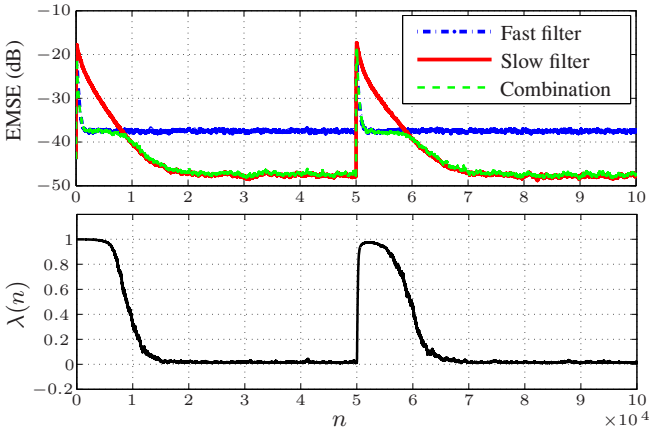


**Fig. 4**. Combination in Matlab precision (floating point with 64 bits: 11 bits for the exponent, 52 bits for the mantissa, and one bit for the sign). Average of $L = 100$ realizations.

for the number of bits for the difference filter. The fixed exponent used for the fast filter is $k_b = 0$, so that $\boldsymbol{w}_1(n)$ is represented by variables in the range $[-1, 1)$. Overflow is handled by saturation. The auxiliary variable $a(n)$ is also represented with $B = 14$ bits, but in the interval $[-4, 4)$. Computing $B_1$ through (15), we obtain $B_1 = 2.5$. We used $k_\delta = 2$. Under these conditions, for $B_\delta = 9$ bits we should obtain a steady-state EMSE for the combination as given by (14) of $\zeta_2 = -47.2$ dB, while the value observed in Figure 5 is $-46.5$dB. This corresponds to a performance deterioration of approximately 1.1 dB in terms of EMSE, when compared to the case of Matlab precision (Figure 4). Figure 6 presents results for $B_\delta = 8$ bits, showing a steady-state EMSE of $-44.3$ dB, while the theoretical value from (14) is $-46.4$dB. Comparing to the case of Matlab precision, we have now a performance deterioration of approximately 3.3 dB in terms of EMSE.

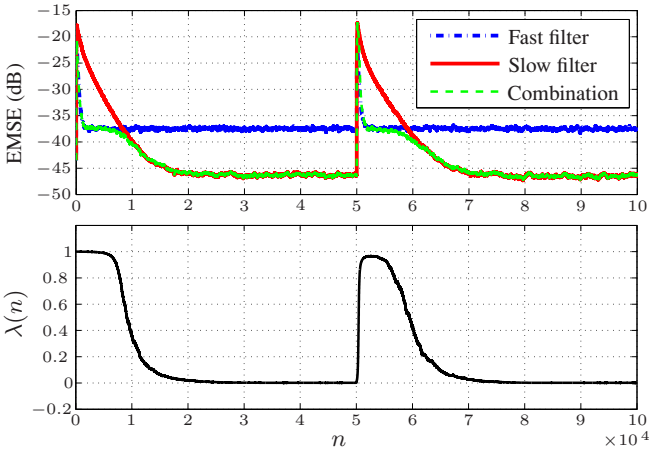


**Fig. 5**. Low-cost combination with $B_\delta = 9$ bits and $k_\delta = 2$ for the difference filter, $B = 14$ bits and $k_b = 0$ for other variables. Average of $L = 100$ realizations.

The reduction in complexity is higher if we use floating point. Using only $B_d = 6$ bits (mantissa only) and floating point to implement $\boldsymbol{\delta w}(n)$ and $e_\delta(n)$ (but still fixed-point with $B = 14$ for the other variables), the result is equivalent to $B_d = 8$ bits in fixed-point, as shown in Figure 7, where the steady-state EMSE of the

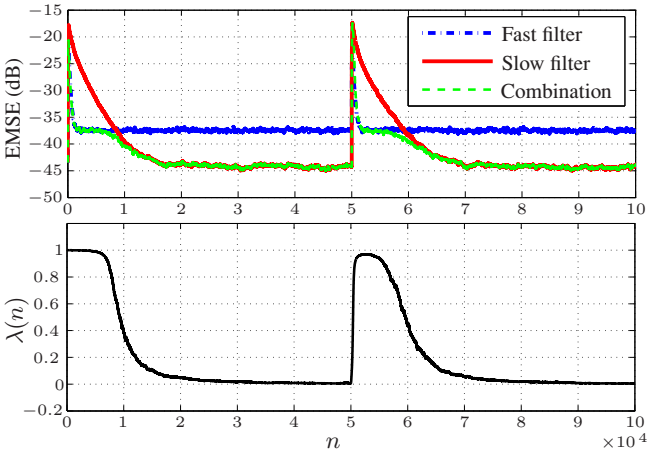


**Fig. 6**. Low-cost combination with $B_\delta = 8$ bits and $k_\delta = 2$ for the difference filter, $B = 14$ bits and $k_b = 0$ for other variables. Average of $L = 100$ realizations.

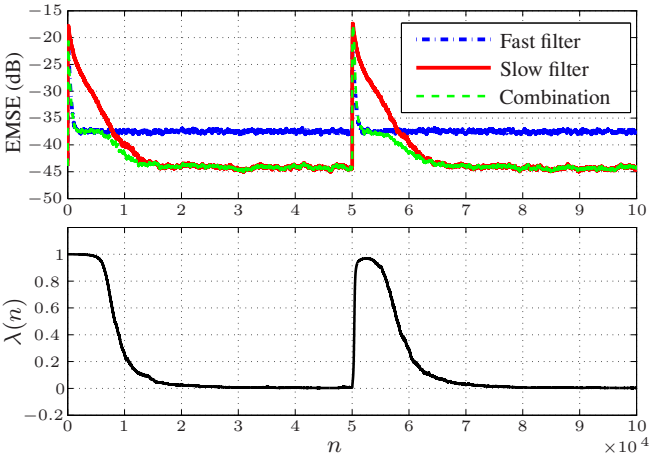


**Fig. 7**. Low-cost combination with $B_\delta = 6$ bits and floating point for the difference filter, $B = 14$ bits and $k_b = 0$ for other variables. Average of $L = 100$ realizations.

combination is $\zeta = -44.3$ dB. Again, this corresponds to a performance deterioration of approximately 3.3 dB in terms of EMSE, when compared to the case of Matlab precision (Figure 4).

We observe that the proposed scheme (Figures 5–7) presents approximately the same convergence rate of that of Figure 4, and a steady-state performance that depends on the number of bits used to represent the difference filter.

## 6. CONCLUSION

In this paper, we proposed a combination of adaptive filters with reduced computational complexity. Instead of adapting the slow filter as usual, we update the difference between the slow and fast filters, which can be represented using lower-precision arithmetic. The new scheme can be used with fixed-point, floating-point, or even mixed implementations. Depending on the number of bits used for representing the difference filter, a small degradation on steady-state EMSE will be observed, but the convergence rate is not affected.

## 7. REFERENCES


[1] J. Arenas-García, A. R. Figueiras-Vidal, and A. H. Sayed, "Mean-square performance of a convex combination of two adaptive filters," *IEEE Trans. Signal Process.*, vol. 54, no. 3, pp. 1078–1090, Mar. 2006.

[2] Y. Zhang and J. Chambers, "Convex combination of adaptive filters for a variable tap-length LMS algorithm," *IEEE Signal Process. Lett.*, vol. 13, no. 10, pp. 628–631, Oct. 2006.

[3] N. J. Bershad, J. C. M. Bermudez, and J.-Y. Tourneret, "An affine combination of two LMS adaptive filters—transient mean-square analysis," *IEEE Trans. Signal Process.*, vol. 56, no. 5, pp. 1853–1864, May 2008.

[4] M. T. M. Silva and V. H. Nascimento, "Improving the tracking capability of adaptive filters via convex combination," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3137–3149, Jul. 2008.

[5] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, "Steady-state MSE performance analysis of mixture approaches to adaptive filtering," *IEEE Trans. Signal Process.*, vol. 58, no. 8, pp. 4050–4063, Aug. 2010.

[6] R. Fa, R. C. de Lamare, and V. H. Nascimento, "Knowledge-aided stap algorithm using convex combination of inverse covariance matrices for heterogenous clutter," in *Proc. IEEE Int Acoustics Speech and Signal Processing (ICASSP) Conf*, 2010, pp. 2742–2745.

[7] V. H. Nascimento, M. T. M. Silva, L. A. Azpicueta-Ruiz, and J. Arenas-García, "On the tracking performance of combinations of least mean squares and recursive least squares adaptive filters," in *Proc. IEEE Int Acoustics Speech and Signal Processing (ICASSP) Conf*, 2010, pp. 3710–3713.

[8] W. B. Lopes and C. G. Lopes, "Incremental-cooperative strategies in combination of adaptive filters," in *Proc. IEEE Int Acoustics, Speech and Signal Processing (ICASSP) Conf*, 2011, pp. 4132–4135.

[9] J. C. M. Bermudez, N. J. Bershad, and J.-Y. Tourneret, "Stochastic analysis of an error power ratio scheme applied to the affine combination of two LMS adaptive filters," *Signal Processing*, vol. 91, no. 11, pp. 2615–2622, 2011.

[10] L. A. Azpicueta-Ruiz, M. Zeller, A. R. Figueiras-Vidal, J. Arenas-García, and W. Kellermann, "Adaptive combination of Volterra kernels and its application to nonlinear acoustic echo cancellation," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 19, pp. 97–110, Jan. 2011.

[11] V. H. Nascimento, M. T. Silva, R. Candido, and J. Arenas-García, "A transient analysis for the convex combination of adaptive filters," in *Proc. IEEE Workshop on Statistical Signal Process.*, Cardiff, UK, 2009, pp. 53–56.

[12] R. Candido, M. T. M. Silva, and V. H. Nascimento, "Transient and steady-state analysis of the affine combination of two adaptive filters," *IEEE Trans. Signal Process.*, vol. 58, no. 8, pp. 4064–4078, 2010.

[13] A. Gonzalo-Ayuso, M. T. M. Silva, V. H. Nascimento, and J. Arenas-García, "Improving sparse echo cancellation via convex combination of two nlms filters with different lengths," in *2012 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, sept. 2012, pp. 1–6.

[14] A. H. Sayed, *Adaptive Filters*. NJ: Wiley, 2008.

[15] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, 3rd ed. Springer, 2008.

[16] M. Lázaro-Gredilla, L. A. Azpicueta-Ruiz, A. R. Figueiras-Vidal, and J. Arenas-García, "Adaptively biasing the weights of adaptive filters," *IEEE Trans. Signal Process.*, vol. 58, no. 7, pp. 3890–3895, 2010.

[17] L. Azpicueta-Ruiz, A. Figueiras-Vidal, and J. Arenas-García, "A normalized adaptation scheme for the convex combination of two adaptive filters," in *Proc., ICASSP 2008*, Las Vegas, NV, USA, pp. 3301–3304.

[18] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 3rd ed. NY: Springer, 2007.

[19] F. Auger, Z. Lou, B. Feuvrie, and F. Li, "Multiplier-free divide, square root, and log algorithms [dsp tips and tricks]," *IEEE Signal Process. Mag.*, vol. 28, no. 4, pp. 122–126, 2011.

[20] Y. Zakharov and T. Tozer, "Multiplication-free iterative algorithm for LS problem," *Electronics Letters*, vol. 40, no. 9, pp. 567–569, 2004.

[21] Y. Zakharov, G. White, and J. Liu, "Low-complexity RLS algorithms using dichotomous coordinate descent iterations," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3150–3161, 2008.