Fast Implementations of the Filtered-X LMS and LMS Algorithms for Multichannel Active Noise Control

Scott C. Douglas, Senior Member, IEEE

Abstract—In some situations where active noise control could be used, the well-known multichannel version of the filtered–X least mean square (LMS) adaptive filter is too computationally complex to implement. In this paper, we develop a fast, exact implementation of this adaptive filter for which the system’s complexity scales according to the number of filter coefficients within the system. In addition, we extend computationally efficient methods for effectively removing the delays of the secondary paths within the coefficient updates to the multichannel case, thus yielding fast implementations of the LMS adaptive algorithm for multichannel active noise control. Examples illustrate both the equivalence of the algorithms to their original counterparts and the computational gains provided by the new algorithms.

Index Terms—Acoustic noise, active noise control, adaptive control, adaptive filters, adaptive signal processing, least mean square methods, vibration control.

I. INTRODUCTION

I

terest in active methods for the suppression of noise and vibration has grown recently, as evidenced by the numerous review articles and books that have appeared on the subject [1]–[9]. Although the potential for active noise and vibration control has long been recognized [10], successful implementations of these techniques have begun to appear only recently. Such success can be attributed to the rapid maturation of technology in three areas: 1) novel electroacoustic transducers, 2) advanced adaptive control algorithms, and 3) inexpensive and reliable digital signal processing (DSP) hardware. As advances in these areas are developed, active suppression of noise and vibration can be expected to find wider use in a number of commercial, industrial, and military applications. In this paper, we focus on the algorithms used in multichannel active noise and vibration control systems as implemented in DSP hardware.

Perhaps the most popular adaptive control algorithm used in DSP implementations of active noise and vibration control systems is the filtered–X least-mean-square (LMS) algorithm [11]. There are several reasons for this algorithm’s popularity. First, it is well-suited to both broadband and narrowband control tasks, with a structure that can be adjusted according to the problem at hand. Second, it is easily described and understood, especially given the vast background literature on adaptive filters upon which the algorithm is based [12], [13]. Third, its structure and operation are ideally suited to the architectures of standard DSP chips, due to the algorithm’s extensive use of the multiply/accumulate (MAC) operation. Fourth, it behaves robustly in the presence of physical modeling errors and numerical effects caused by finite-precision calculations. Finally, it is relatively simple to set up and tune in a real-world environment.

Despite its popularity, the standard filtered–X LMS algorithm suffers from one drawback that makes it difficult to implement when a multichannel controller is desired: the complexity of the coefficient updates for the finite impulse response (FIR) filters within the controller in these situations is much greater than the complexity of the input–output calculations. It is not unusual for the coefficient updates of the standard implementation to require more than ten times the number of MAC’s needed to compute the outputs of the controller for fixed coefficient values, and the situation worsens as the number of error sensors is increased. For this reason, recent efforts have focused on ways to reduce the complexity of the filtered–X LMS algorithm in a multichannel context. Suggested changes include: 1) block processing of the coefficient updates using fast convolution techniques [14], 2) partial updating of the controller coefficients [15], and 3) filtered-error methods [16]–[18]. While useful, these methods often reduce the overall convergence performance of the controller, either because they introduce additional delays into the coefficient update loop or because they throw away useful information about the state of the control system. Such a performance loss may not be tolerable in some applications.

In addition to these computational difficulties, the multichannel filtered–X LMS algorithm also suffers from excessive data storage requirements. This algorithm employs filtered input signal values that are created by filtering every input signal by every output-actuator-to-error-sensor channel of the acoustic plant. The number of these terms can be an order-of-magnitude greater than the number of controller coefficients and input signal values used in the input–output calculations. As typical DSP chips have limited on-chip memory, system designers may be forced to use costly off-chip memory within their controller architectures that can further slow the operation of the system due to limits in input/output data throughput.
While some of the aforementioned techniques for complexity reduction also have reduced memory requirements, the performance of the overall system is effectively limited by these methods.

A third limitation of the multichannel filtered–X LMS adaptive controller is due to the propagation delays caused by the physical distances between the output actuators and the error sensors. Because of these delays, the error signals contain delayed versions of the controller coefficients, and these delays lead to a reduced stability range for the stepsize parameter and slower convergence speeds [19]. If the impulse responses of the secondary paths between the output actuators and the error sensors can be accurately estimated, then it is possible to approximately calculate the true LMS adaptive updates for the controller filters, as described in [20] and [21] in the single-channel case. However, a straightforward extension of this idea to the multichannel case yields an algorithm with approximately twice the complexity of the original filtered–X LMS controller. More recently, techniques for efficiently calculating the LMS adaptive updates for a single-channel controller have been provided in [22]–[24]. These techniques have not been extended to the multichannel case, however, and any additional simplifications resulting from such an extension have not been explored.

In this paper, we present novel methods for reducing the computational and memory requirements of the multichannel filtered–X LMS and multichannel LMS adaptive controllers. Our solutions are alternative implementations of these systems that are mathematically equivalent to the original implementations, and thus they preserve the characteristic robust and accurate behaviors of the algorithms. The complexity and memory requirements of the new implementations, however, are significantly reduced over those of the original implementations, especially for controllers with a large number of channels. Moreover, since the filtered-input signals are not needed in our implementations, the excessive memory requirements of the original implementations are avoided.

This paper is organized as follows. For simplicity of discussion, Section II presents the original as well as our novel implementation of the filtered–X LMS algorithm in the single-channel case, although the new implementation’s computational savings are only realized in the multichannel case. The multichannel extensions are provided in Section III, along with illustrative examples indicating the computational savings obtained with the new implementation. In Section IV, we provide two extensions of the method of calculating the LMS coefficient updates for an adaptive controller in [23] to the multichannel case, showing how the algorithm can be integrated with the efficient multichannel algorithm in Section III. Example simulations in Section V show the equivalence of the new algorithms to their more complex counterparts, and simple methods for mitigating the marginal stabilities of the sliding-window calculations within the new algorithms are provided. As for mathematical notation, scalar variables are employed throughout the paper to enable the algorithms’ direct translation to DSP processor code, and indices of parameter sets are for the most part lower-case versions of the variable designating the number of parameters; e.g., \( u_l(n) \) for \( 0 \leq l \leq L - 1 \).

II. SINGLE-CHANNEL FILTERED–X LMS ALGORITHMS

A. Standard Implementation

To simplify our discussion, we initially present the single-channel filtered–X LMS adaptive feedforward controller; the multichannel filtered–X LMS algorithm is described in Section III. Fig. 1 shows a block diagram of this system, in which a sensor placed near a sound source collects samples of the input signal \( x(n) \) for processing by the system. This system computes an actuator output signal \( y(n) \) using a time-varying FIR filter of the form

\[
y(n) = \sum_{l=0}^{L-1} w_l(n) x(n-l)
\]

where \( w_l(n) \), \( 0 \leq l \leq L - 1 \) are the controller coefficients at time \( n \) and \( L \) is the controller filter length. The acoustic output signal produced by the controller combines with the sound as it propagates to the quiet region, where an error sensor collects the combined signal. We model this error as

\[
\varepsilon(n) = d(n) + \sum_{m=-\infty}^{\infty} h_m y(n-m)
\]

where \( d(n) \) is the undesired sound as measured at the error sensor and \( h_m \), \( -\infty < m < \infty \) is the plant impulse response. Note that (2) is never computed as \( \varepsilon(n) \) is a measurement of a physical quantity. In addition, (2) assumes that the secondary propagation path is linear and time-invariant. Although changes in room acoustics can occur over time and loudspeakers often have nonlinear transfer characteristics at low frequencies and high driving levels, we assume for simplicity throughout this paper that (2) is an accurate model for the error sensor signal.

The filtered–X LMS coefficient updates are given by

\[
u_l(n+1) = u_l(n) - \mu(n) \varepsilon(n) f(n-l)
\]

where \( \mu(n) \) is the algorithm step size at time \( n \), the filtered input sequence \( f(n) \) is computed as

\[
f(n) = \sum_{m=1}^{M} h_m x(n-m)
\]

and \( M \) is the FIR filter length of an appropriate estimate of the plant impulse response. In practice, the values of \( h_m \) used in (4) are estimates of the actual \( h_m \) in (2) and are usually obtained in a separate estimation procedure that is performed.
prior to the application of control. For notational simplicity, we will not distinguish the differences in these two parameter sets in what follows. A discussion of the performance effects caused by errors in the estimates of \( h_m \) can be found in [25].

A study of (1), (3), and (4) shows that the filtered–X LMS algorithm requires \( 2L + M + 1 \) MAC operations and \( 2L + \max\{L, M + 1\} + 1 \) memory locations to store the necessary \( \varphi_k(n), h_m, x(n-l), \) and \( f(n-l) \) for the algorithm at each step. For typical choices of the controller and plant filter lengths, the complexity and memory requirements of this algorithm are reasonable. As will be shown, however, such is not the case for the natural extension of this algorithm to the multichannel control task.

B. New Implementation

We now describe a new implementation of the single-channel filtered–X LMS algorithm [26]–[28]. This method combines the adjoint LMS/corrected phase filtered error (CPFE) algorithm [17], [18] with a method for delay compensation used in fast projection adaptive filters [29], [30]. To derive the implementation, we write the coefficient updates of the original algorithm in the form

\[
\varphi(n+1) = \varphi(n) - \mu(n) \epsilon(n) \sum_{m=1}^{M} h_m x(n-l-m), \tag{5}
\]

Define \( \epsilon_m(n) \) for \( 1 \leq m \leq M \) as

\[
\epsilon_m(n) = h_m \mu(n) \epsilon(n). \tag{6}
\]

Then, (5) becomes

\[
\varphi(n+1) = \varphi(n) - \sum_{m=1}^{M} \epsilon_m(n) x(n-l-m). \tag{7}
\]

We can represent the relation in (7) for \( M \) successive time steps as

\[
\varphi(n+1) = \varphi(n-M+1) - \sum_{m=1}^{M-1} \sum_{p=1}^{M} \epsilon_m(n-p) \cdot x(n-l-m-p). \tag{8}
\]

We can expand the summation on the right-hand-side of (8) in a particularly useful way as in (9), shown at the bottom of the page, where we define the \( l \)th auxiliary coefficient \( \hat{\varphi}_l(n) \) as

\[
\hat{\varphi}_l(n) = \varphi(n-M+1) - \sum_{p=1}^{M-1} \sum_{m=p+1}^{M} \epsilon_m(n-M+m-p) \cdot x(n-l-M-p). \tag{10}
\]

The expression in (9) indicates an important fact about the structure of the filtered–X LMS updates: the same input sample \( x(n-l-m) \) is used in successive time instants to update the same coefficient \( \varphi_k(n) \). We can exploit this structure to develop a set of coefficient updates that are grouped according to the individual \( x(n-l-m) \) values appearing on the right-hand-side of (9). Such a scheme updates the \( l \)th auxiliary coefficient \( \hat{\varphi}_l(n) \) rather than the actual controller coefficient \( \varphi_k(n) \). Define \( \epsilon(n) \) as

\[
\epsilon_m(n) = \sum_{p=1}^{m} \epsilon_p(n-M+p). \tag{11}
\]

Then, it is straightforward to show that \( \hat{\varphi}_l(n) \) can be updated as

\[
\hat{\varphi}(n+1) = \hat{\varphi}(n) - \epsilon_m(n-1) x(n-l-m-1). \tag{12}
\]

Thus, \( \hat{\varphi}(n+1) \) is obtained by subtracting from \( \hat{\varphi}(n) \) the last column of terms on the RHS of (9).

Since \( \epsilon_m(n) \) is obtained by filtering \( \mu(n) \epsilon(n) \) by the time-reversed plant impulse response \( \{h_M, h_{M-1}, \ldots, h_1\} \), (12) is the single-channel version of the adjoint LMS/CPFE algorithm [17], [18]. What is novel is the relationship in (9) that provides the link between \( \hat{\varphi}_l(n) \) and \( \varphi_k(n) \), or, equivalently, the link between the adjoint LMS/CPFE and filtered–X LMS algorithms. We can use (9) to compute \( y(n) \) for the filtered–X LMS algorithm using \( \hat{\varphi}_l(n) \) as calculated by (12). To proceed, we substitute the expression for \( \varphi(n+1) \) in (7) into (9). Using (11), we obtain

\[
y(n) = \sum_{l=0}^{L-1} \hat{\varphi}_l(n) x(n-l) - \sum_{l=0}^{L-1} \sum_{m=1}^{M-1} \epsilon_m(n-1) \cdot x(n-l-m-1). \tag{13}
\]

Substituting the expression for \( \varphi(n) \) in (13) into (1), we produce the equivalent expression

\[
y(n) = \sum_{l=0}^{L-1} \hat{\varphi}_l(n) x(n-l) - \sum_{l=0}^{L-1} \sum_{m=1}^{M-1} \epsilon_m(n-1) \cdot x(n-l-m-1) \cdot x(n-l). \tag{14}
\]

Define the correlation term \( r_m(n) \) as

\[
r_m(n) = \sum_{l=0}^{L-1} x(n-l-m) x(n-l). \tag{15}
\]

Then, (14) becomes

\[
y(n) = \sum_{l=0}^{L-1} \hat{\varphi}_l(n) x(n-l) - \sum_{m=1}^{M-1} \epsilon_m(n-1) r_{m+1}(n). \tag{16}
\]
Such a calculation is of reasonable complexity because $r_m(n)$ can be recursively updated as

$$r_m(n) = r_m(n-1) + x(n)x(n-m) - x(n-L)x(n-L-m), \quad (17)$$

Moreover, $e_m(n)$ has a simple order-recursive update of the form

$$e_m(n) = \begin{cases} h_k e_k(n), & \text{if } m = 1 \\ e_{m-1}(n-1) + h_m e_k(n), & \text{if } 2 \leq m \leq M \end{cases} \quad (18)$$

where

$$e_k(n) = \mu(n)e(n). \quad (19)$$

Collecting (12) and (16)–(19), we obtain a set of equations that exactly computes the output signal of the filtered–X LMS adaptive controller. This algorithm requires $2L + 4M - 1$ MAC’s to implement at each iteration. Thus, this version is more computationally complex than the original implementation of the filtered–X LMS algorithm, which only requires $2L + M + 1$ MAC’s per iteration. In the multichannel case, however, the alternative implementation can save operations and memory storage, as we now show.

III. MULTICHANNEL FILTERED–X LMS ALGORITHMS

A. Standard Implementation

We now describe the multichannel version of the filtered–X LMS algorithm in its original implementation [7], [8]. In multichannel control, $I$ input sensors are used to collect $I$ input signals $x_i(n), 1 \leq i \leq I$. The controller computes $J$ output signals $y_j(n), 1 \leq j \leq J$ as

$$y_j(n) = \sum_{i=1}^{I} \sum_{k=0}^{L-1} w_{k}^{(i,j)}(n)x_i(n-I) \quad (20)$$

where $w_{k}^{(i,j)}(n), 0 \leq k \leq L-1$ are the $L$ FIR filter coefficients for the $i$th-input-to-$j$th-output channel of the controller. The $J$ controller output signals propagate to the desired quiet region, where $K$ error sensors measure the error signals $e_k(n), 1 \leq k \leq K$ as

$$e_k(n) = d_k(n) + \sum_{j=1}^{J} \sum_{m=-\infty}^{\infty} h_{m}^{(j,k)}y_j(n-m) \quad (21)$$

and $h_{m}^{(j,k)}, -\infty < m < \infty$ is the $j$th-output-to-$k$th-error plant impulse response channel.

In the original filtered–X LMS algorithm, $IJK$ filtered input signals $f^{(i,j,k)}(n)$ are computed as

$$f^{(i,j,k)}(n) = \sum_{m=1}^{M} h_{m}^{(j,k)}x_i(n-m) \quad (22)$$

from which $w_{k}^{(i,j)}(n)$ is updated as

$$w_{k}^{(i,j)}(n+1) = w_{k}^{(i,j)}(n) - \sum_{k=1}^{K} \epsilon_{k}^{(i,j)}(n)f^{(i,j,k)}(n-I) \quad (23)$$

where

$$\epsilon_{k}^{(i,j)}(n) = \mu(n)e^{(i,j,k)}(n). \quad (24)$$

A careful study of the filtered–X LMS algorithm described by (20) and (22)–(24) reveals the fact that this implementation requires $IJK(L+M)+K$ MAC’s to compute the coefficient updates, even though computing the controller outputs only requires $IJL$ MAC’s. Thus, the complexity of the update calculations is more than $K$ times the complexity of the input–output calculations. For systems with a large number of error sensors, the computational burden of the coefficient updates can overwhelm the capabilities of the processor chosen for the control task.

The standard implementation of the filtered–X LMS algorithm also has memory requirements that can exceed the capabilities of a chosen processor. The total storage needed is $IJ(K+1)L + IJKM + I\max(L, M+1) + K$, and for long controller filter lengths, the bulk of this storage is for the $IJKL$ filtered input signals $f^{(i,j,k)}(n-I)$. Clearly, it is desirable to find alternative implementations of the filtered–X LMS algorithm that have reduced computational and memory requirements. We now present an algorithm that is based on the method described in Section II.

B. New Implementation

We consider the multichannel extension of the new version of the filtered–X LMS algorithm in Section II-B. To determine the appropriate grouping of terms for the updates, we substitute the expression for $f^{(i,j,k)}(n-I)$ in (22) into the update in (23) to get

$$w_{k}^{(i,j)}(n+1) = w_{k}^{(i,j)}(n) - \sum_{k=1}^{K} \epsilon_{k}^{(i,j)}(n)e^{(i,j,k)}(n-I) \quad (25)$$

$$= w_{k}^{(i,j)}(n) - \sum_{m=1}^{M} e_{m}^{(j)}(n)h_{m}^{(j,k)}x^{(i,j,k)}(n-I-m) \quad (26)$$

where we have defined the $JM$ terms $e_{m}^{(j)}(n)$ for $1 \leq j \leq J$ and $1 \leq m \leq M$ as

$$e_{m}^{(j)}(n) = \sum_{k=1}^{K} h_{m}^{(j,k)}\epsilon_{k}^{(i,j)}(n). \quad (27)$$

Because (26) and (7) are similar in form, we can use a method analogous to that in Section II-B to implement the multichannel filtered–X LMS algorithm. Define

$$e_{m}^{(j)}(n) = \sum_{k=1}^{m} \epsilon_{m}^{(j)}(n-m+p). \quad (28)$$

Then, we can define a set of $IJK$ auxiliary coefficients $\tilde{w}_{k}^{(i,j)}(n)$ whose updates are given by

$$\tilde{w}_{k}^{(i,j)}(n+1) = \tilde{w}_{k}^{(i,j)}(n) - e^{(i,j,k)}(n-I) \quad (29)$$
To compute the controller outputs, the multichannel equivalent of (16) is

\[
y^{(j)}(n) = \sum_{i=1}^{I} \sum_{k=0}^{L-1} w^{(j),i}(n) x^{(i)}(n-I) - \sum_{m=1}^{M-1} e_{m+1}^{(j)}(n) r_{m+1}(n)
\]  

(30)

where \( r_{m}(n) \) in this case is defined as

\[
r_{m}(n) = \sum_{i=1}^{I} \sum_{k=0}^{L-1} x^{(i)}(n-I) x^{(i)}(n-I-m).
\]  

(31)

In analogy with (17), \( r_{m}(n) \) can be recursively computed as

\[
r_{m}(n) = r_{m}(n-1) + \sum_{i=1}^{I} \left\{ x^{(i)}(n) x^{(i)}(n-m) - x^{(i)}(n-L) x^{(i)}(n-L-m) \right\}.
\]  

(32)

Similarly \( e_{m}^{(j)}(n) \) has an update similar to that in (18), as given by

\[
e_{m}^{(j)}(n) = \begin{cases} 
\sum_{k=1}^{K} l_{i}^{(j),k}(k) e_{m}^{(j)}(n), & \text{if } m = 1 \\
\sum_{k=1}^{K} l_{m}^{(j),k}(k) e_{m-1}^{(j)}(n-1), & \text{if } 2 \leq m \leq M.
\end{cases}
\]  

(33)

Collecting (24), (29), (30), (32), and (33), we obtain an alternative, equivalent implementation of the multichannel filtered–X LMS algorithm. Table I lists the operations of this implementation, along with the number of MAC’s required to implement each operation. The algorithm employs \( 2IJL + JKM + (2I + J)(M-1) + K \) MAC’s per iteration, and it requires \( IJL + JKM + IL + (I + J + 1)M + K - 1 \) memory locations to implement.

Remark: This implementation of the multichannel filtered–X LMS adaptive controller modifies the adjoint LMS/CPFE adaptive controller by including the second summation on the RHS of (30) and the supporting updates for \( \epsilon_{m}^{(j)}(n) \) and \( r_{m}(n) \), respectively. Since \( \epsilon_{m}^{(j)}(n) \) is of \( O(g(n)) \), the performance difference between the multichannel filtered–X and adjoint LMS/CPFE algorithms can only be expected to be significant for large stepsizes, a fact that has been pointed out in [17], [18]. Because the adjoint LMS/CPFE algorithm is a filtered-error technique with an approximate group delay of \( M \) samples in the update rule, however, its performance is often worse than that of the filtered–X LMS algorithm. Moreover, the complexity difference between the two algorithms is relatively insignificant for systems with a large number of channels, as will now be shown.

C. Complexity Comparisons

We now compare the computational and memory requirements of the original and fast implementations of the multichannel filtered–X LMS algorithm. In this comparison, we consider three different problem scenarios. Each scenario is defined by specific choices of the controller filter length \( L \) and plant model filter length \( M \) that might be appropriate for a particular type of noise or vibration control task. In each case, we present the quantities \( R_{F}^{(C)} \) and \( R_{F}^{(M)} \) that denote the ratios of the numbers of MAC’s and memory locations,
respectively, required by the fast implementation with respect to the numbers of MAC's and memory locations needed for the original implementation. For comparison, we also provide the corresponding ratios $R_A^{(C)}$ and $R_A^{(M)}$ for the adjoint LMS/CPFE algorithm [17], [18] with respect to the original filtered–X LMS algorithm. Since the adjoint LMS/CPFE algorithm equations are used within the fast implementation, we have that $R_A^{(C)} = R_A^{(C)}$ and $R_A^{(M)} = R_A^{(M)}$, although the two algorithms' requirements are similar for systems with a large number of channels.

The first situation considered is a broadband noise control task in which the controller and plant model filter lengths are $L = 50$ and $M = 25$, respectively. The ratio $L/M = 2$ offers a reasonable balance between the performance and the robustness of the controller for fixed hardware resources in many applications. Table II shows the complexity and memory ratios for the different cases considered. As can be seen for all of the cases considered, the number of multiplies required for the new implementation of the multichannel filtered–X LMS algorithm is less than that of the original algorithm, and this difference is significant for systems with a large number of channels. In fact, for an $N$-input, $N$-output, $N$-error system, the complexity of the new implementation is approximately 80% of the original implementation when $N = 2$, 40% of the original when $N = 4$, 20% of the original when $N = 8$, and 10% of the original when $N = 16$. In addition, the number of memory locations required by the new implementation is also reduced and is less than 10% of the original algorithm’s memory requirements for $I = J = K = N = 16$. These savings are significant, as they allow a multichannel control system to be implemented on a much simpler hardware platform.

We now consider tasks in which $L = 2$ and $M = 10$. Such a situation is typical of narrowband noise control problems in which each input signal is a single sinusoid of a different frequency; thus, each channel of the controller is dedicated to one tonal component of the unwanted acoustic field. Table III lists the ratio of MAC’s and memory locations for the two algorithms with respect to the original filtered–X LMS algorithm in this situation. As can be seen, except for systems with a small number of channels, the new implementation requires

### Table II

<table>
<thead>
<tr>
<th>Channels</th>
<th>Complexity Ratio</th>
<th>Memory Ratio</th>
<th>Channels</th>
<th>Complexity Ratio</th>
<th>Memory Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$J$</td>
<td>$K$</td>
<td>$R_A^{(C)}$</td>
<td>$R_A^{(M)}$</td>
<td>$R_E^{(C)}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0.7512</td>
<td>0.7765</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0.7508</td>
<td>0.7301</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0.7506</td>
<td>0.7054</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>0.7504</td>
<td>0.6897</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.6259</td>
<td>0.6559</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.6256</td>
<td>0.6868</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0.6255</td>
<td>0.7132</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0.6253</td>
<td>0.5508</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.5726</td>
<td>0.5323</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0.5722</td>
<td>0.5383</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0.5009</td>
<td>0.5085</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0.5015</td>
<td>0.4888</td>
<td>19</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Channels</th>
<th>Complexity Ratio</th>
<th>Memory Ratio</th>
<th>Channels</th>
<th>Complexity Ratio</th>
<th>Memory Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$J$</td>
<td>$K$</td>
<td>$R_A^{(C)}$</td>
<td>$R_A^{(M)}$</td>
<td>$R_E^{(C)}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.5472</td>
<td>1.0566</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.5433</td>
<td>0.9430</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0.5429</td>
<td>0.8857</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0.4902</td>
<td>0.9349</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.4851</td>
<td>0.6634</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0.4834</td>
<td>0.6325</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0.4631</td>
<td>0.6588</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0.4527</td>
<td>0.5287</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0.2824</td>
<td>0.3870</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0.2585</td>
<td>0.3133</td>
<td>25</td>
</tr>
</tbody>
</table>
TABLE IV

<table>
<thead>
<tr>
<th>Channels</th>
<th>Complexity Ratio</th>
<th>Memory Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$H_A^{(C)}$</td>
<td>$H_F^{(C)}$</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.5735 97.87</td>
<td>0.9098 98.77</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.5735 88.56</td>
<td>0.8779 93.31</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.5730 84.34</td>
<td>0.8604 90.32</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.4656 68.32</td>
<td>0.7488 79.56</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.4636 60.92</td>
<td>0.7102 73.50</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.4629 58.44</td>
<td>0.6963 71.13</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.4226 56.57</td>
<td>0.6381 64.99</td>
</tr>
<tr>
<td>2 4 2</td>
<td>0.4016 49.48</td>
<td>0.6067 61.85</td>
</tr>
<tr>
<td>4 4 4</td>
<td>0.3865 39.37</td>
<td>0.5452 55.56</td>
</tr>
<tr>
<td>4 4 4</td>
<td>0.2639 31.38</td>
<td>0.4760 48.18</td>
</tr>
<tr>
<td>4 4 4</td>
<td>0.2408 27.87</td>
<td>0.4440 44.85</td>
</tr>
</tbody>
</table>

only a fraction of the MAC’s and memory locations used by the original implementation. Thus, the new implementation reduces the controller’s hardware complexity in narrowband control situations as well.

The third problem scenario considered is a task in which $L = 10$ and $N = 20$. These choices are typical for noise and vibration control tasks in which the input signals are measured by physical sensors, but the primary goal of the controller is to attenuate a relatively few number of tonal components. Table IV lists the respective complexity and memory usage ratios for different cases. As in the previous cases, we find that the new implementation of the filtered–X LMS algorithm saves computations and memory locations for systems with a large number of channels.

IV. LMS ALGORITHMS FOR ACTIVE NOISE CONTROL

A. Standard Implementation

In this section, we review the standard method for reducing the effects of the plant delay on the filtered–X LMS algorithm’s operation and the resulting LMS algorithm for active noise control [20], [21]. Considering the single-channel filtered–X LMS adaptive controller, it is seen from (2) that the error signal $e(n)$ depends on the outputs $y(n - m)$ of the controller at different time instants, which in turn depend on the controller coefficients $u_k(n - m)$ at different time instants. Because the plant is typically causal, past coefficients are employed within the gradient-based updates, causing a decrease in the performance of the system not unlike that observed for the delayed LMS algorithm [31]. It is possible to largely mitigate the effects of this delay by computing a delay-compensated error signal that depends on the most-recent coefficients $u_k(n)$. Fig. 2 shows the block diagram of this system, in which $\gamma(k)$ is the delay-compensated error signal given by

$$\gamma(n) = \left\{e(n) - \sum_{m=1}^{M} h_m y(n-m) \right\} + \sum_{l=0}^{L-1} u_l(n) f(n-l) \tag{34}$$

where the term within brackets on the RHS of (34) is nearly the same as the unattenuated noise signal $d(n)$ if the estimated impulse response $h_m$ accurately models the unknown plant’s impulse response. The LMS algorithm for active noise control uses $\gamma(k)$ to update the coefficients [20], [21] as

$$u_k(n+1) = u_k(n) - \mu(k) \gamma(k) f(n-l) \tag{35}$$

This algorithm requires a total of $3L + 2M + 1$ MAC’s per iteration to implement, and it uses $2L + M + \max\{L, M+1\}$ memory locations. Note that this algorithm’s performance depends on how well the estimated plant impulse response models the physical response of the plant. As our focus is on implementation and not performance issues, a performance analysis of the multichannel LMS algorithm for active noise control is beyond the scope of this paper.

We can easily extend the above algorithm to the multichannel case. In this situation, we compute the $K$ delay-compensated error signals

$$\gamma(k) = \gamma(k) - \sum_{j=1}^{J} \sum_{m=1}^{M} h_{m}^{(j)} y_{j}(n-m) \tag{36}$$

at which point $\mu(n) \gamma(k) (n)$ is used in place of $e_k^{(k)}(n)$ in (23). Unfortunately, this modification adds $IKL + JKM$ MAC’s per iteration to the overall requirements of the adaptive system if the necessary $f^{(k,j,k)}(n)$ values are available, and it adds $IKL + (2I + 1) JKM$ MAC’s if $f^{(k,j,k)}(n)$ must be
computed. In addition, the storage requirements for the overall system are significantly increased if the modification is applied to the fast multichannel filtered–X LMS algorithm in Table I.

B. New Implementations

1) A Multichannel Extension of an Existing Algorithm: In [23], a method is presented for reducing the complexity of the single-channel LMS algorithm for active noise control when the secondary path length \( M \) is less than a third of the controller filter length \( L \). We now extend this algorithm to the multichannel case. Define

\[
\gamma^{(n)}_{\mu}(n) = \mu(n)\gamma^{(k)}(n) = \mu(n)\left\{e^{(k)}(n) + \sum_{j=1}^{I} \sum_{m=1}^{M} h^{(j,k)}_{m}\gamma^{(j)}_{\mu}(n-1)\right\} \tag{37}
\]

where \( \gamma^{(j)}_{\mu}(n) \) is defined as

\[
\gamma^{(j)}_{\mu}(n) = y^{(j)}(n-m) - \sum_{i=0}^{I} \sum_{l=0}^{L-1} \gamma^{(i,j)}_{\mu}(n-l-m) x^{(i)}(n-l-m). \tag{38}
\]

Using algebraic manipulations similar to those in [23], an update for \( u^{(j)}_{\mu}(n) \) is found to be

\[
u^{(j)}_{\mu}(n) = \begin{cases} -\sum_{k=1}^{K} f^{(j,k)}_{m}(n)\gamma^{(k)}_{\mu}(n), & \text{if } m = 1 \\ \nu^{(j)}_{\mu}(n-1) - \sum_{k=1}^{K} f^{(j,k)}_{m-1}(n)\gamma^{(k)}_{\mu}(n), & \text{if } 2 \leq m \leq M \end{cases} \tag{39}
\]

where \( f^{(j,k)}_{m}(n) \) is defined as

\[
f^{(j,k)}_{m}(n) = \sum_{i=1}^{I} \sum_{l=0}^{L-1} x^{(i)}(n-l-m) f^{(i,j,k)}(n-l). \tag{40}
\]

Note that \( f^{(j,k)}_{m}(n) \) can be updated as

\[
f^{(j,k)}_{m}(n) = f^{(j,k)}_{m}(n-1) + \sum_{i=1}^{I} \{x^{(i)}(n-m)f^{(i,j,k)}(n) - x^{(i)}(n-m-L) f^{(i,j,k)}(n-L)\} \tag{41}
\]

which greatly reduces the number of operations needed for the algorithm when \( L \) is large. This update also reduces the amount of memory required for the algorithm, as \( f^{(i,j,k)}(n) \) and \( f^{(i,j,k)}(n-L) \) can be computed at each iteration to avoid storing \( f^{(i,j,k)}(n-L) \) for \( 1 \leq i \leq L-1 \).

Collecting (37), (39), and (41), we obtain a multichannel delay compensation technique that requires \((4I + 2)JKM\) MAC’s per iteration and \((J + J + JKM)M\) memory locations to implement when \( L > M \), assuming that \( f^{(i,j,k)}(n) \) and \( f^{(i,j,k)}(n-L) \) are computed at each iteration. Comparing these complexity requirements with those of the original delay compensation technique, if

\[
L > \left(2 + \frac{1}{I}\right)M \tag{42}
\]

then this new technique is more computationally efficient. The new technique also has low memory requirements and thus is an ideal match to the fast algorithm in Table I.

2) An Alternate Implementation: Although useful, the delay-compensation method in (37), (39), and (41) can be prohibitive to implement when the number of channels is large, as its complexity grows as \( O(JKM) \). We now consider an alternate implementation that uses many of the existing quantities within the efficient multichannel filtered–X LMS algorithm in Table I while avoiding the formation of the filtered input signal values. For this derivation, consider the definition of \( \rho^{(j,k)}_{m}(n) \) in (40). Substituting the expression for \( f^{(i,j,k)}(n) \) in (22) into the RHS of (40) and rearranging, we obtain

\[
\rho^{(j,k)}_{m}(n) = \sum_{q=1}^{M} h^{(j,k)}_{q} \left\{\sum_{l=0}^{I} \sum_{i=1}^{I} x^{(i)}(n-l-m) x^{(i)}(n-l-q)\right\} = \sum_{q=1}^{M} h^{(j,k)}_{q} r_{m-q}(n-q) \tag{43}
\]

where \( r_{m}(n) \) is as defined in (32). From the definition of \( r_{m}(n) \), it is straightforward to show that

\[
r_{m-q}(n-q) = r_{m}(n-m) \tag{44}
\]

and thus the necessary values of \( r_{m-q}(n-q) \) to represent \( \rho^{(j,k)}_{m}(n) \) can be obtained from \( r_{m}(n) \), \( 0 \leq m \leq M \) by storing delayed values of these quantities. Define

\[
e^{(j)}_{m}(n) = \sum_{k=1}^{K} f^{(j,k)}_{m}(n)\gamma^{(k)}_{\mu}(n). \tag{45}
\]

Note that \( e^{(j)}_{m}(n) \) appears in the update for \( e^{(j)}_{m}(n) \) in (33) when the delay compensation technique is combined with the fast filtered–X LMS algorithm; thus, it is already available. Then,

\[
\sum_{k=1}^{K} f^{(j,k)}_{m-1}(n)\gamma^{(k)}_{\mu}(n) = \sum_{q=1}^{M} e^{(j)}_{q}(n) r_{m-1-q}(n-q) \tag{46}
\]

and the RHS of (46) can replace the summations on the RHS of the updates for \( u^{(j)}_{\mu}(n) \) in (39).

Table V lists the operations for this alternative form of the LMS algorithm for multichannel active noise control. This algorithm requires \((I + 1)K(M + 1)(M - 1)\) more MAC’s per iteration than does the filtered–X LMS algorithm in Table I. If

\[
M > \frac{1}{J} \tag{47}
\]

then this implementation is more computationally efficient than that in (37), (39), and (41). If

\[
M < IK + \sqrt{IKL + PK^2 + 1} \tag{48}
\]
TABLE V
A Multichannel LMS Algorithm with Reduced Complexity for Active Noise Control

<table>
<thead>
<tr>
<th>Equation</th>
<th>MACs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_m(n) = r_m(n-1) + \sum_{i=1}^{I} \left( x^{(i)}(n)x^{(i)}(n-m) - z^{(i)}(n-L)x^{(i)}(n-L-m) \right)$</td>
<td>$2I(M+1)$</td>
</tr>
<tr>
<td>$\gamma^{(k)}<em>{\mu}(n) = \mu(n) \left[ \hat{e}^{(k)}(n) + \sum</em>{j=1}^{J} \sum_{m=1}^{M} h^{(j,k)}_{\mu}(n-1) \right]$</td>
<td>$JKM + K$</td>
</tr>
<tr>
<td>$y^{(j)}(n) = \sum_{i=1}^{I} \sum_{K=0}^{L-1} a^{(i,j)}_{K}(n)x^{(i)}(n-K)$</td>
<td>$IJL + J(M-1)$</td>
</tr>
<tr>
<td>$e^{(j)}<em>{\mu}(n) = \sum</em>{K=1}^{K} h^{(j,k)}_{\mu}(n)$</td>
<td>$JK$</td>
</tr>
<tr>
<td>$e^{(j)}_{\mu}(n)$</td>
<td>$JK(M-1)$</td>
</tr>
<tr>
<td>$e^{(j)}<em>{m+1}(n) = e^{(j)}</em>{m}(n-1) + e^{(j)}_{m+1}(n)$</td>
<td>$JM$</td>
</tr>
<tr>
<td>$u^{(j)}<em>{\mu}(n) = \sum</em>{i=1}^{M} h^{(j,k)}_{\mu}(n)$</td>
<td>$JM$</td>
</tr>
<tr>
<td>$u^{(j)}<em>{m+1}(n) = u^{(j)}</em>{m}(n-1) - \sum_{q=1}^{M} e^{(j)}<em>{q}(n)r</em>{m-q}(n-q)$</td>
<td>$JM(M-1)$</td>
</tr>
<tr>
<td>$\hat{w}^{(j)}<em>{(n+1)} = \hat{w}^{(j)}</em>{(n)} + \hat{e}^{(j)}_{(n)}z^{(i)}(n-M-L)$</td>
<td>$IJL$</td>
</tr>
</tbody>
</table>

Total: $2IJL + 2JKM + (2I + JM)(M+1) + K$

then this implementation is more computationally efficient than the standard implementation in (36). Considering the system configurations listed in Tables II–IV, we find that the algorithm in Table V is the most computationally efficient method out of the three delay-compensation techniques considered when $IK \geq 7$, $IK \geq 5$, and $IK \geq 8$, respectively. For the remaining configurations, the standard delay-compensation implementation combined with the new filtered–X LMS update method in Table II is the most efficient, although the method in (37), (39), and (41) is the most efficient for the configurations in Table II if the controller filter length is increased to $L = 75$.

Remark: These implementations of the multichannel LMS adaptive controller modify the filtered–X LMS adaptive controller by including the summation within brackets on the RHS of (37) and the supporting updates for $w^{(j)}_{m}(n)$. Since $u^{(j)}_{m}(n)$ is of $O(\mu(n))$, the performance difference between the two multichannel LMS algorithms and the filtered–X LMS algorithm can only be expected to be significant for large stepsizes. Note that the filtered–X LMS algorithm is typically derived assuming “slow adaptation,” so that the derivatives of the error signals with respect to the filter coefficients can be easily calculated [11]. Our multichannel LMS algorithms quantitatively define the difference between the filtered–X LMS and LMS coefficient updates and provide an alternative justification for the former algorithm for situations in which the stepsize is small-valued.

V. SIMULATIONS AND NUMERICAL ISSUES

In this section, we consider the effects that numerical errors due to finite precision calculations have on the performances of the new implementations of the filtered–X LMS and LMS algorithms for active noise control. One important feature of the LMS algorithm in adaptive filtering is its robust behavior in the presence of various approximations and errors that are often introduced in a real-world implementation. Since the original implementation of the filtered–X LMS algorithm and the adjoint LMS/CPFE algorithm are variants of stochastic gradient methods [16], they share many of the robust convergence properties of the LMS algorithm. The new implementations of the filtered–X LMS and LMS algorithms apply one or more forms of delay compensation to the adjoint LMS/CPFE algorithm. As such, the numerical properties of the delay compensation techniques are of immediate interest, particularly as they affect the long-term performances of the systems.
While formal analyses of the numerical properties of the delay compensation techniques used in our implementations are beyond the scope of this paper, extensive simulations of the implementations have indicated that the robust numerical properties of the underlying stochastic gradient algorithms are not fundamentally altered in our new implementations. These behaviors are quite unlike those of fast RLS/Kalman techniques that exhibit an exponential instability unless careful measures are taken [32], [33]. The only possible source of numerical difficulty is the method for calculating $r_m(n)$ in (32), as this update is marginally stable. Thus, numerical errors in $r_m(n)$ can grow linearly over time in a finite-precision environment, particularly in floating-point realizations in which relatively-few bits are allocated for the mantissas of the terms used to update each $r_m(n)$. Fortunately, the growth in these errors can be easily prevented using several well-known procedures. Perhaps the simplest procedure is to periodically recalculate $r_m(n)$ using its definition in (31), a procedure that requires extra additions and memory locations. Moreover, because each $r_m(n)$ has a finite memory by definition, accumulating and copying its value to the appropriate memory location within the controller causes no performance penalty, unlike periodic restart methods in exponentially windowed fast RLS/Kalman filters [32]. Another solution is to introduce a leakage factor into the calculation of $r_m(n)$. One particularly useful method, described in more detail in [34], is in (49), shown at the bottom of the page, where $\lambda$ is slightly less than one. This method alters the value of $r_m(n)$ slightly, but for values of $\lambda$ close to one, the errors introduced into the calculations for $y(n)$ do not significantly affect the overall behaviors of the respective systems. Moreover, the update in (49) adds only $M$ MAC’s and a single comparison to each systems’ overall complexity.

Figs. 3–5 plot the envelope of the sum-of-squared errors $\sum_{k=1}^{R} \{e^{(k)}(n)\}^2$ for a total of seven different four-input, three-output, four-error active noise control algorithms with $L = M = 50$ as applied to air compressor data measured in an anechoic environment [35]. In this case, all calculations were performed in the MATLAB floating-point environment, and the approximate sampling rate of the data was 4 kHz. Stepsizes for each algorithm were chosen to provide the fastest convergence on this data while yielding approximately the same steady-state error power due to limits in noise modeling error. Fig. 3 shows the unattenuated air compressor noise signal, in which the bursty nature of the compressor noise is clearly evident, along with the average error power envelopes of the original filtered–X LMS and LMS algorithms applied to this data, in which the stepsizes for each algorithm were chosen as $\mu = 0.1$ and $\mu = 0.2$, respectively. Shown for comparison in Fig. 4 are the average error power envelopes of the adjoint LMS/CPFE algorithm, the fast filtered–X LMS algorithm in Table I, and the new multichannel LMS algorithm in Table V, in which the stepsizes for each algorithm were chosen as $\mu = 0.05$, $\mu = 0.1$, and $\mu = 0.2$, respectively. As can be seen, the fast multichannel filtered–X LMS algorithm outperforms the adjoint LMS/CPFE algorithm in its convergence rate, and the multichannel LMS algorithm performs the best of the three due to the lack of coefficient delay within the parameter updates. In addition, the differences in the error signals between the original and fast algorithms in Figs. 3 and 4 were found to be about ten times the order of the machine precision used in the simulation ($\approx 10^{-16}$) after 60,000 iterations. A linear growth of the numerical errors was apparent, however.

Shown in Fig. 5 are the behaviors of the fast multichannel filtered–X LMS and fast multichannel LMS algorithms in which the leakage-based update for $r_m(n)$ in (49) is employed, where $\lambda = 0.999$. Comparing the average error powers with those of the corresponding algorithms in Fig. 4, no discernible differences in performance can be seen. In fact, the actual differences between the errors of the corresponding systems were less than $2 \times 10^{-10}$ in magnitude in this example—a negligible difference—and no growth in the numerical errors of $r_m(n)$ was observed. Thus, the method in (49) can be used to stabilize the marginal instability of the sliding-window $r_m(n)$ updates without altering the observed performances of the proposed systems.

VI. CONCLUSIONS

We have described new implementations of the multichannel filtered–X LMS and LMS algorithms for feedforward

$$r_m(n) = \begin{cases} 
\lambda \left\{ r_m(n-1) - \sum_{i=1}^{I} x^{(i)}(n-L)x^{(i)}(n-L-m) \right\} + \sum_{i=1}^{I} x^{(i)}(n)x^{(i)}(n-m), & \text{if } n \mod L = 0 \\
 r_m(n-1) + \sum_{i=1}^{I} x^{(i)}(n)x^{(i)}(n-m) - \lambda \left\{ \sum_{i=1}^{I} x^{(i)}(n-L)x^{(i)}(n-L-m) \right\}, & \text{otherwise}
\end{cases}$$

(49)
active noise and vibration control tasks. These implementations provide the same input–output behaviors of the original implementations while requiring only a fraction of the computational effort and memory of the original implementations. Because of the pervasiveness of stochastic-gradient-based algorithms for active noise and vibration control systems, the new implementations are expected to have a significant impact on the practicality and cost of these schemes in real-world applications.

ACKNOWLEDGMENT

The author would like to thank J. K. Soh and the numerous anonymous reviewers for helpful comments on this material. Data was provided by SRI International, Menlo Park, CA.

REFERENCES


Scott C. Douglas (S’88–M’92–SM’98) received the B.S. (with distinction), M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1988, 1989, and 1992, respectively. From 1992 to 1998, he was an Assistant Professor in the Department of Electrical Engineering, University of Utah, Salt Lake City. Since August 1998, he has been with the Department of Electrical Engineering, Southern Methodist University, Dallas, TX, as an Associate Professor. He is a frequent consultant to industry in the areas of signal processing and adaptive filtering. His research activities include adaptive filtering, active noise control, blind deconvolution and source separation, and VLSI/hardware implementations of digital signal processing systems. He is the author or co-author of four book chapters and more than 80 articles in journals and conference proceedings; he also served as a Section Editor for The Digital Signal Processing Handbook (Boca Raton, FL: CRC, 1998). He has one patent pending.

Dr. Douglas received the Hughes Masters Fellowship Award in 1988 and the NSF Graduate Fellowship Award in 1989. He was a recipient of the NSF CAREER Award in 1995. He is currently an Associate Editor for the IEEE Transactions on Signal Processing, the IEEE Signal Processing Letters, and the Journal of VLSI Signal Processing Systems. He is a member of both the Neural Networks for Signal Processing Technical Committee and the Education Technical Committee of the IEEE Signal Processing Society. He served on the Technical Program Committees of the 1995 IEEE Symposium on Circuits and Systems, Seattle, WA, and the 1998 IEEE Digital Signal Processing Workshop, Bryce Canyon, UT. He is the Proceedings Co-chair of the 1998 Workshop on Neural Networks for Signal Processing, Madison, WI, and is the Proceedings Editor of the 1999 International Symposium on Active Control of Sound and Vibration, Ft. Lauderdale, FL. He is the exhibits co-chair of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing, Salt Lake City. He is a member of Phi Beta Kappa and Tau Beta Pi.