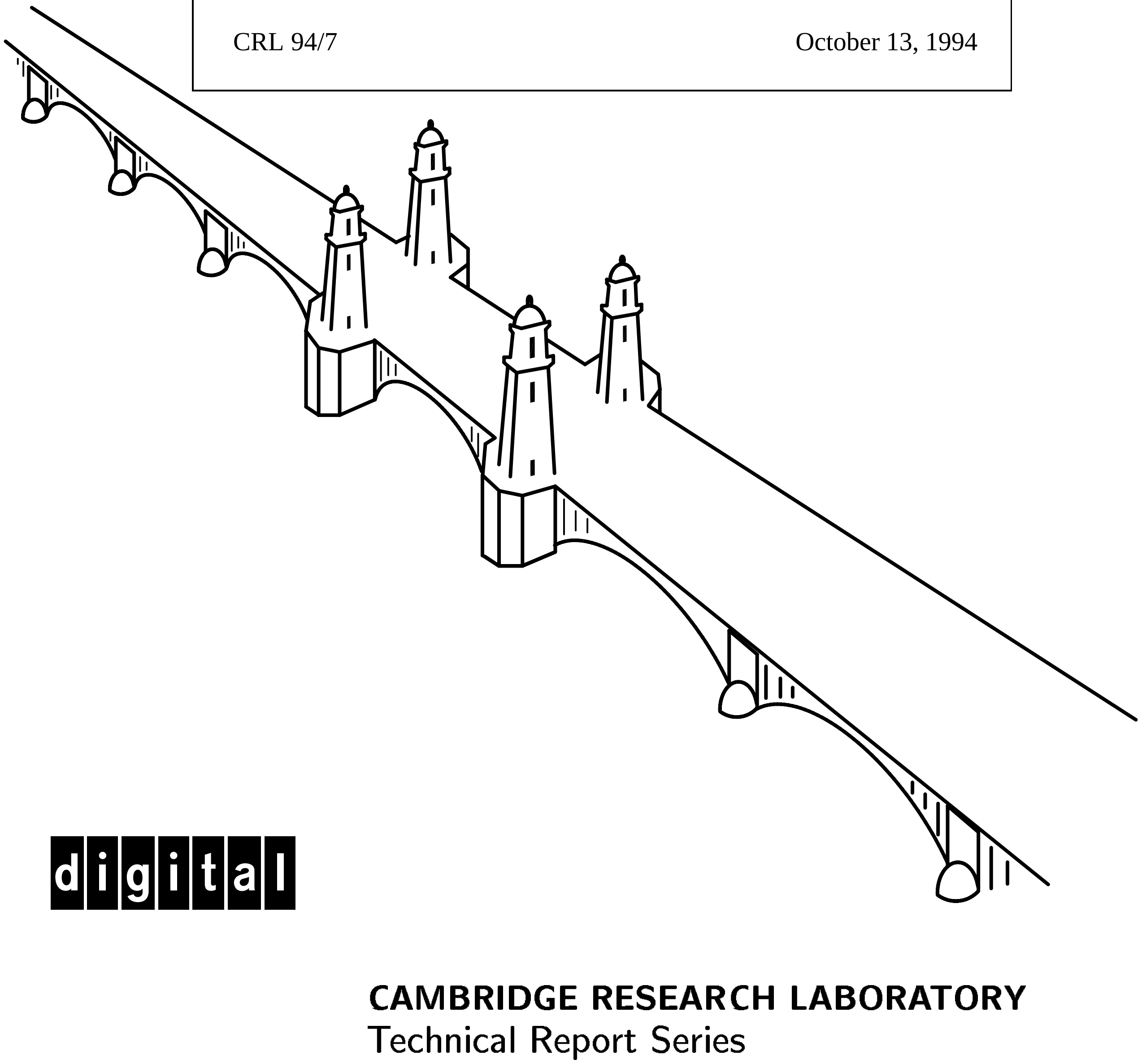
Echo Cancelation

Mathieu C. Hans and Thomas M. Levergood

Digital Equipment Corporation
Cambridge Research Lab

CRL 94/7 October 13, 1994



Echo Cancelation

Mathieu C. Hans[1] and Thomas M. Levergood

Digital Equipment Corporation
Cambridge Research Lab

CRL 94/7 October 13, 1994

Abstract

Typically, users of teleconferencing desktop applications must use headphones to eliminate the direct-path echo introduced by having a loudspeaker and a microphone in the same room. Unfortunately, many users prefer not to use headphones. Implementing echo cancelation would be one way for desktop audio conferencing systems to overcome this limitation.

One technique used to remove the echo is to introduce an autoregressive filter to identify the echo path impulse response defined by the system {Speaker + Room + Microphone}. The signal output to the speaker is then filtered by the estimated echo path impulse response, and the result is subtracted from the degraded input speech to obtain an estimate of the input speech without echoes.

In this paper, we describe the identification process which leads to the least mean square (LMS) algorithms and the recursive least square (RLS) algorithms. We also present an "Echo-Cancelation Software Lab" which was implemented and optimized to allow real-time testing of the LMS, normalized LMS, homogeneous adaptation, and individual adaptation algorithms.

Keywords: audio, echo cancelation, least mean square, adaptive filtering, gradient algorithm, conferencing

[1]Summer intern and PhD student from Georgia Institute of Technology.

Contents

1 Introduction

A typical desktop audio conferencing system uses a speaker and a microphone at each end of the transmission link. An audible problem occurs, however, when the output of the speaker is simultaneously recorded by the microphone and then transmitted digitally, hence creating a direct-path echo which interferes with spoken words. Also, reverberations of the output speaker signal on walls, tables and human bodies located in the conference room add to this direct echo and constitute indirect echoes.

To overcome this problem, a replica of the echoes can be synthesized by modeling the echo path and then subtracting the result from the outgoing signal. Adaptive filtering methods can be used to implement this model, and the system $\{$ Speaker + Room + Microphone $\}$ can be modelled with an autoregressive filter with time-varying coefficients. The least mean square (LMS) and recursive least square (RLS) algorithm classes are most commonly used to update the filter coefficients.

In this paper, Section 2 relates the identification process which leads to the LMS and the RLS algorithms. The LMS algorithms are then described in Section 3 in more detail. Section 4 presents an “Echo-Cancelation Software Lab” which was implemented to allow real-time testing of LMS algorithms such as the LMS, the normalized LMS, the homogeneous adaptation, and the individual adaptation.

2 Identification, Model and Criterion

2.1 Identification

In parametric spectrum analysis, the power spectrum of a process is estimated by assuming a model $H(z)$ for this process[1]. In the case of the echo path impulse response, the unknown system $\{$Speaker + Room + Microphone$\}$ is identified as the output of a linear adaptive filter that is excited by a white-noise process[2].

Figure 1 defines the identification procedure used to evaluate the model. The unknown system and the model are driven by the same input $u(n)$ which is the incoming speech signal from the far-end of a conference. A criterion on the difference $e(n)$ of the output is optimized, and a feedback is given to the model for adaptation. $e(n)$ also defines the outgoing signal and is played back at the far-end speaker.

[1] A good reference for adaptive filter theory is [1].

[2] By definition, a white-noise process has a constant power spectrum.

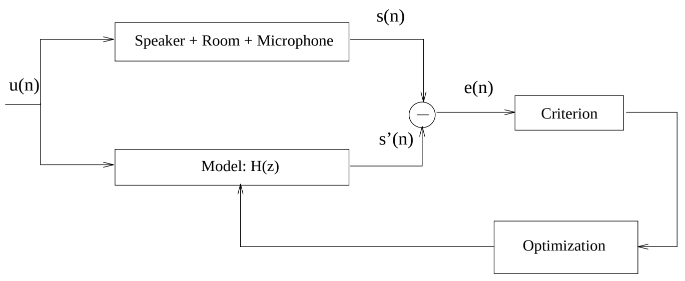


Figure 1: Identification of unknown system {Speaker + Room + Microphone}

2.2 Model

A model that is of practical utility because of its simple implementation is the autoregressive (AR) model. Indeed, the filter's transfer function is assumed to consist only of poles. Let this transfer function be denoted by

$$H(z) = \frac{G}{1 + \sum_{i=1}^{p} a_i z^{-i}}$$

where the a_i are called the autoregressive parameters or filter coefficients, p the model order and G, the system's scale factor.

Another way to understand the AR model is to see the output $s(n)$ of the unknown system as a linear prediction of the delayed output values and an unknown input signal $u(n)$ which is defined as white noise:

$$s(n) = \sum_{i=1}^{p} a_i s(n-i) + Gu(n)$$

Therefore, we will approximate the output signal recorded by the microphone as a linear prediction of the past output signal:

$$s'(n) = \sum_{i=1}^{p} a_i s(n-i)$$

When using this model, we have the following error:

$$e(n) = s(n) - s'(n) = s(n) - \sum_{i=1}^{p} a_i s(n-i)$$

For the optimal case where the unknown system is autoregressive, the variance of the error is the square of the scale factor:

$$E[e(n)^2] = G^2$$

where $E[x^2]$ is the expected value of x^2.

2.3 Criterion

Since the statistical properties of the room change slowly in time, we define the model to be time varying.

To estimate the time dependent filter coefficients, we can choose among adaptive methods that are differentiated by their criterion. There are two criteria commonly used:

- If the criterion is defined as minimizing the mean squared error at time n, $E[e(n)^2]$, compared to the autoregressive parameters, then the algorithm is a least mean square based algorithm.
- If the criterion is defined as minimizing a weighted sum of squared errors such as

  $$\sum_{k=1}^{n} w(k) e_p^2(k)$$

  where $w(0), w(1), ...$ is a sequence of weights, then the algorithm is called a recursive least square (RLS) algorithm.

The computational complexity of the LMS and RLS algorithms are $O(p)$ and $O(p^2)$, (with p being the order of the model). The use of the LMS algorithms is widespread due to its computational simplicity.

Note that fast RLS algorithms have been introduced that circumvent the computational burden of the conventional RLS algorithms. There are two families of such fast algorithms, corresponding to two possible filter structures: the fast lattice (FLA) and the fast transversal filter (FTF) algorithms. While the LMS has a computational complexity of $2p$ multiplications, the FLA and the FTF need $8p$ to $30p$ multiplications. Although less burdensome computationally than RLS, the FTF algorithm can be subject to numerical instability problems. A solution to this problem is proposed in [2].

3 The Mean-Squared-Error Criterion

3.1 The Gradient or Steepest-Descent Method

The criterion of the LMS algorithms is defined as minimizing the Mean-Square-Error $E[e(n)^2]$ compared to the filter coefficient vector:

$$\underline{a}_p(n) = (a_1(n), a_2(n), \cdots, a_p(n))^T$$

The idea is to search for the point $\underline{a}_{p_{min}}$ on the surface $f(\underline{a}_p(n))$ defined by

$$f(\underline{a}_p(n)) = E[(s(n) - \sum_{i=1}^{p} a_i(n)s(n-i))^2]$$

To do this, the steepest-descent method is used by successive approximations, converging to the vector $\underline{a}_{p_{min}}$. The mean-squared-error surface can be viewed as a $(p+1)$-dimensional paraboloid where p is the number of coefficients (or taps) of the impulse response. The optimum impulse response then corresponds to the minimum of the paraboloid.

The gradient method has three steps:

- initialization:
  - chose initial vector $\underline{a}_p(0)$
- propagation:
  - compute gradient at point $\underline{a}_p(n)$, defined as $\nabla f(\underline{a}_p(n))$
  - move on the surface by going in the opposite direction of the gradient, modifying the vector $\underline{a}_p(n)$ as follows:

$$\underline{a}_p(n+1) = \underline{a}_p(n) - \mu \nabla f(\underline{a}_p(n))$$

- termination:
  - stop when $|\nabla f(\underline{a}_p(n))| < threshold$

where μ is the displacement step. As we will show, μ controls the speed and condition of convergence.

3.2 The Gradient Algorithm

The following notations are used to define the gradient.

$$\begin{aligned}
\underline{s}_p(n) &= (s(n-1), s(n-2), \cdots, s(n-p))^T \\
\underline{R}_p &= E[\underline{s}_p(n)\underline{s}_p(n)^T] \\
\underline{c}_p &= E[s(n)\underline{s}_p(n)]
\end{aligned}$$

The gradient is defined by equation 1 and the propagation step of the gradient algorithm is given by equation 2.

$$\nabla f(\underline{a}_p(n)) = -2E[e_p(n)\underline{s}_p(n)] = 2\underline{R}_p\underline{a}_p(n) - 2\underline{c}_p \quad (1)$$
$$\underline{a}_p(n+1) = \underline{a}_p(n) + 2\mu E[e_p(n)\underline{s}_p(n)] \quad (2)$$

Using equations 1 and 2 and assuming that the gradient is 0 at the minimum, the estimation error of vector $\underline{a}_p(n)$ is defined by

$$(\underline{a}_p(n+1) - \underline{a}_{p_{min}}) = [\underline{I} - 2\mu\underline{R}_p](\underline{a}_p(n) - \underline{a}_{p_{min}})$$

From this expression, we have information about the convergence of the gradient algorithm: the algorithm converges if at each step the norm of vector $(\underline{a}_p(n+1) - \underline{a}_{p_{min}})$ is smaller than the norm of vector $(\underline{a}_p(n) - \underline{a}_{p_{min}})$. This is the case if all the eigenvalues λ_k of the correlation matrix $\underline{R}_p$ satisfy the following condition[3]:

$$0 < \mu < \frac{1}{\lambda_k}$$

[3] Pointers to initial proofs are given in [3].

So, by restricting μ to

$$0 < \mu < \frac{1}{\lambda_{max}}$$

we are sure that the algorithm converges.

Finding an estimate of λ_{max} is not easy. To bypass the estimation, we use the fact that

$$Tr(\underline{R}_p) = \sum_{k=1}^{p} \lambda_k \geq \lambda_{max}$$

and a sufficient condition (which is restrictive) is to take

$$0 < \mu < \frac{1}{Tr(\underline{R}_p)} = \frac{1}{pE[s(n)^2]}$$

Hence, μ can be controlled with an estimate of $E[s(n)^2]$ which is the average signal power.

3.3 The LMS Algorithms

In practice, the gradient's value is not known, but we can estimate it using the input signal. The LMS algorithms estimate the gradient as the instantaneous expected value:

$$\nabla f(\underline{a}_p(n)) = -2E[e_p(n)\underline{s}_p(n)] = -2e_p(n)\underline{s}_p(n)$$

With this estimation, the LMS algorithms comprise these steps:

- Initialization:

$$\underline{a}_k(0) = \underline{0}$$

- Filter coefficient update:

$$\underline{a}_k(n+1) = \underline{a}_k(n) + 2\mu e_p(n)\underline{s}(n)$$

- Output of system:

$$e_p(n) = s(n) - \sum_{k=1}^{p} a_k(n)s(n-k)$$

As shown earlier, the convergence factor μ can be controlled with an estimation of the average signal power. The following four algorithms differ only in the definition of this estimation:

- The *LMS* defines the average signal power as a constant P_s. Therefore, the convergence factor is

$$\mu = \frac{1}{2pP_s} \tag{3}$$

- The *Normalized LMS algorithm* (NLMS) defines the input power as

$$P_s(n+1) = \beta P_s(n) + (1-\beta)s^2(n) \tag{4}$$

  with β usually in the range[4] of 0.99 to 0.9999. The convergence factor is then

$$\mu(n) = \frac{1}{2pP_s(n)} \tag{5}$$

- The *Homogeneous Adaptation algorithm* (HA) [4] estimates the input power as

$$P_s(n) = \frac{1}{p}\sum_{i=0}^{p-1} s^2(n-i) \tag{6}$$

  The convergence factor is defined as

$$\mu(n) = \frac{1}{2\sum_{i=0}^{p-1} s^2(n-i)} \tag{7}$$

[4]Note if β is zero, no averaging takes place. As β approaches one, the effective length of the average becomes longer. For β equal to one, the NLMS reduces to the LMS algorithm.

- Finally, instead of defining μ as a scalar, the *Individual Adaptation algorithm* (IA) [4] defines it as a vector, hence controlling each coefficient separately.

$$\mu_j(n) = \frac{|s(n-j)|}{2\sum_{i=0}^{p-1} |s(n-i)|^3} \tag{8}$$

3.4 Computational Complexity

Assuming the filter coefficients are updated at the same time as the output of the filter, the number of multiplications and additions for the different LMS algorithms are shown in Table 1. The last column gives the total number of operations. This table shows that each of the LMS, NLMS, HA and IA algorithms has a computational complexity which is linear in the number of filter taps.

Algorithm	*	$+$	Total number of operations
LMS	$2p+1$	$2p$	$4p+1$
NLMS	$2p+5$	$2p+1$	$4p+6$
HA	$2p+4$	$2p+2$	$4p+6$
IA	$3p+6$	$2p+2$	$5p+8$

Table 1: Computational complexity (p is the number of filter taps)

If no coefficient update is made, the number of operations at each step is algorithm independent and is $p-1$ multiplications and p additions.

Note that in the case of HA, we do not compute the sum of p squared terms each time, instead we update the sum as follows:

$$S(n) = \sum_{i=0}^{p-1} s^2(n-i) = s^2(n) + S(n-1) - s^2(n-p)$$

A similar decomposition was used for IA.

4 Experiments

4.1 Evaluation: ERLE Measurements

In order to evaluate the performance of an echo canceling system, the ratio of the expected value of the microphone output squared $E[s^2(n)]$ divided by the expected value of the error signal squared $E[e^2(n)]$ is monitored. This quantity, in dB, is called the Echo Return Loss Enhancement, or ERLE:

$$ERLE = 10 \log \frac{E[s^2(n)]}{E[e^2(n)]}$$

The expected value is estimated as follows:

$$E[x^2] = \frac{1}{N} \sum_{k=1}^{N} x^2$$

4.2 The "Echo-Cancelation Software Lab"

The "Echo-Cancelation Software Lab" is used to perform real-time testing of adaptive filtering algorithms. For this report, we have implemented the following algorithms: LMS, normalized LMS, homogeneous adaptation, and individual adaptation.

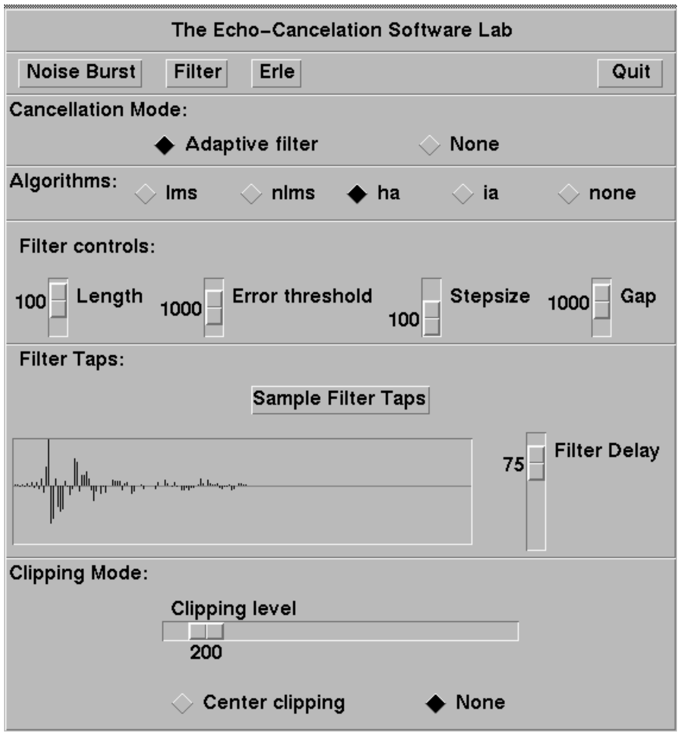
This testbed is written in the 'C' programming language, with a graphical user interface written in Tcl/Tk [5], and uses the AF System [6] for audio support. Figure 2 shows this graphical user interface

As shown in Figure 3, this lab establishes a full-duplex audio connection between two offices. The echo cancelation algorithms were implemented and tested on one side of the conference where a speaker and a microphone were used. In the other office, the user could chose between a headset or a speaker/microphone set.

The user can control the following algorithm parameters:

- The length p of the filter
- The error threshold setting the termination step of the gradient algorithm[5]

[5]To save computation, the termination step is not controlled by the norm of the gradient, but by


The Echo-Cancelation Software Lab

Noise Burst | Filter | Erle | Quit

Cancellation Mode: [✓ Adaptive filter] None

Algorithms: lms nlms [✓ ha] ia none

Filter controls:
100 Length
1000 Error threshold
100 Stepsize
1000 Gap

Filter Taps:
Sample Filter Taps
75 Filter Delay

Clipping Mode:
Clipping level 200
Center clipping [✓ None]

Figure 2: The “Echo-Cancelation Software Lab”

- The stepsize c coefficient controlling the convergence factor[6]. Indeed, to allow a linear control, the convergence factor in the testbed was defined as $\frac{c\mu}{100}$ with c varying from 0 (no filter coefficient update) to 100.
- The gap in sample periods between record and play time-stamps, simulating a transmission time delay between two offices
- The filter delay introduced by the Analog to Digital and Digital to Analog Converter

This test-bed also supports center-clipping echo suppression which, although not described in this paper, can be used to suppress the small amplitude tails of relatively long-delay echos.

the norm of the error which is the output of the system. Therefore, the gradient need not be computed at each output update if the error is small enough.

[6] This coefficient c was not introduced before so as not to overload the notations.

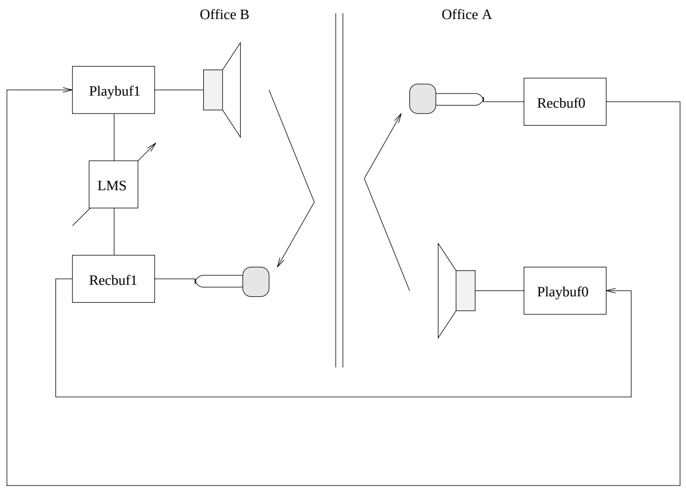


Figure 3: Testbed system diagram

4.3 Results from the "Echo-Cancelation Software Lab"

This experiment was done between two neighboring offices (call them A and B). In both offices, we used MD518 SENNHEISER microphones with a SHURE FP11 microphone pre-amplifier set to the same gain (+54 dB). In office A, a J-Video AF audio server and Realistic Minimus -7 speakers were used, and the microphone was oriented towards the user (i.e. not toward the speaker). In office B, a J300 Sound and Motion board [7] was used with Labtec CS-180 speakers. The microphone in office B was oriented towards the speaker 12 inches away.

4.3.1 Filter Calibration

Starting with arbitrary initial filter coefficients, the algorithm will converge faster if a white noise is played at the input of the adaptive system[7]. During this calibration time, the filter adapts to the current echo path impulse response.

The ERLE measurements for each algorithm were collected during the calibration and are shown in Figure 4. For this experiment, the same parameters were used for all algorithms. As we can see, the algorithms HA and IA converge faster than the NLMS, which converges faster than the LMS. This graph shows the relative performance between the different algorithms for a given set of parameters. Note that

[7] [8] gives a good explanation for why white noise is better during calibration.

this graph does not show the *best* performance of each algorithm. By changing the parameters, one can find the best set for each algorithm and for the studied acoustic environment.

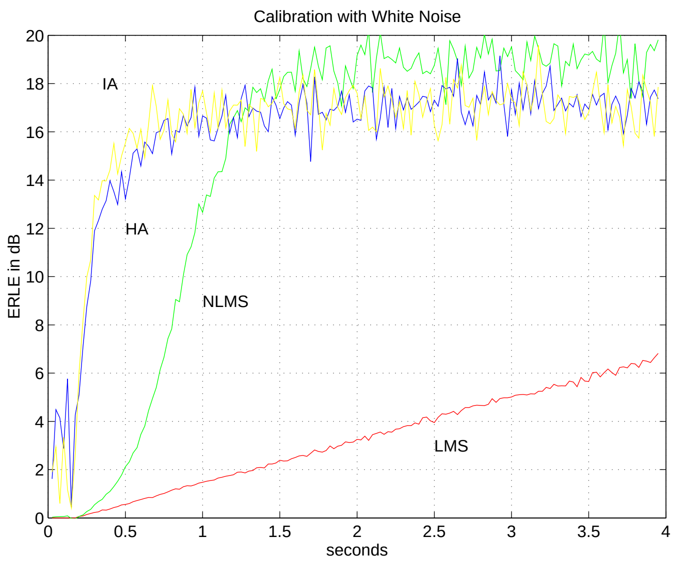


Figure 4: Calibration of the algorithms LMS, NLMS, HA and IA. Parameters used were as follows: filter length of 200, stepsize of 100, an error threshold of 1000, a gap of 1000 samples, and a delay of 280 samples.

4.3.2 The HA Algorithm

We now use the homogeneous adaptation algorithm as an example to demonstrate the effects of varying the filter length, step size, and delay. Comparing the computational complexity of the HA algorithm given in Table 1 and the convergence speed suggested by Figure 4 with the other algorithms, we believe that this algorithm is best from a complexity/performance perspective.

- *Varying the filter length*

  Figure 5 shows two interesting properties as the number of the adaptive filter taps increases:

  - The Echo Return Loss Enhancement also increases. With 50 taps, the ERLE value is 12 dB at steady state, whereas with 250 taps it is 18 dB.

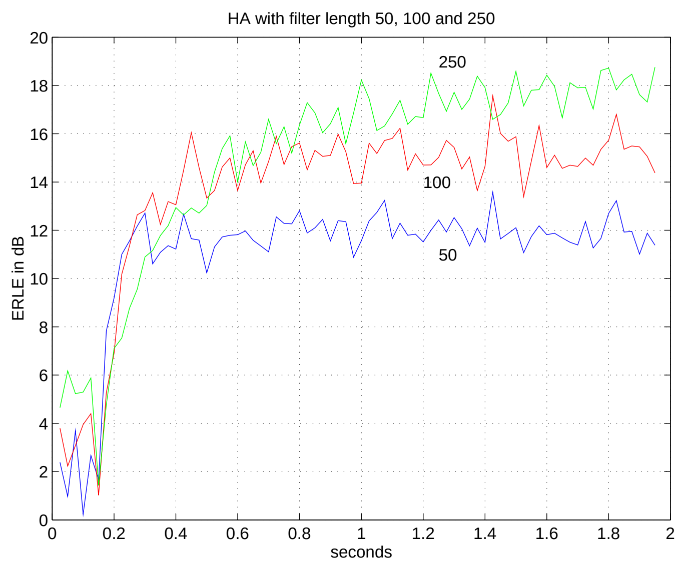


Figure 5: HA algorithm with filter length 50, 100 and 250

– The convergence time also increases. With 50 taps, the steady state is reached after 300 milliseconds and after 2 seconds for 250 taps.

Note that the ERLE value does not start at zero. This is because the HA algorithm started to converge, with the ambient noise from office B, before initiating the noise burst.

- *Varying the stepsize*

  Figure 6 shows that as the stepsize increases, the convergence time decreases. (Recall that for the HA algorithm to be stable, the stepsize $c/100$ may not be greater than 1.0). We note also that the smaller the stepsize is, the closer the algorithm converges to the optimal point (increasing the ERLE value).

- *Varying the delay*

  Figure 7 represents the filter coefficients for the HA algorithm with two different delays (250 and 280 samples). The offset in the filter coefficients is due to the delay introduced by some components in the system (the Analog to Digital Converter and Digital to Analog Converter filter implementations). If this delay is not estimated correctly, most of the leading filter coefficients will be zeroes and CPU power will be wasted.

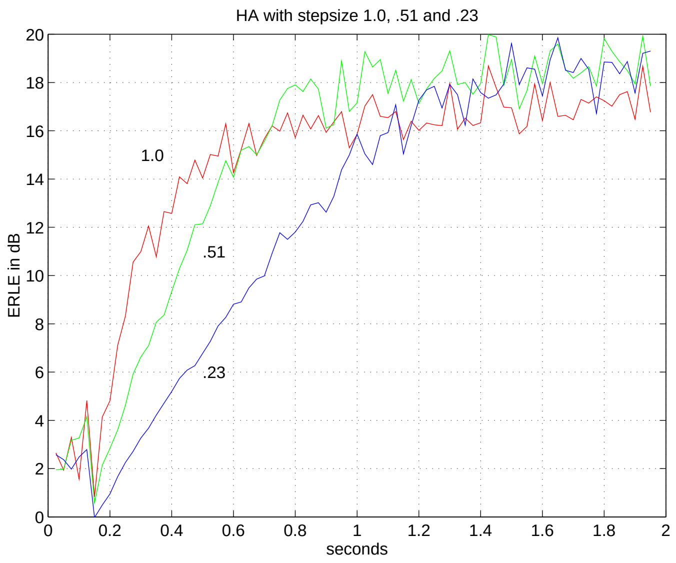


Figure 6: HA algorithm with $\frac{c}{100}$ equal to 1.0, .51 and .23

4.4 Performance Analysis

One major goal for an echo canceler algorithm is to provide real-time operation. Therefore, we implemented the four LMS algorithms, then analyzed the performance and measured the number of cycles per filter tap for each algorithm.

4.4.1 Implementation and Analysis

The four algorithms were implemented in C and compiled on OSF/1 for Alpha V3.0 using the native C compiler. The compilation arguments used were
`-O2 -non_shared -float -float_const`.

The following tools were used to analyze and profile the programs:

- The `omdiag` utility from the ATOM [9] kit was used to diagram how instructions issued.
- The Alpha process cycle counter was used to measure the number of cycles spent updating the output and the filter coefficients. The process cycle counter provides CPU clock cycle resolution, and can be used to gather real-time or process virtual time data.

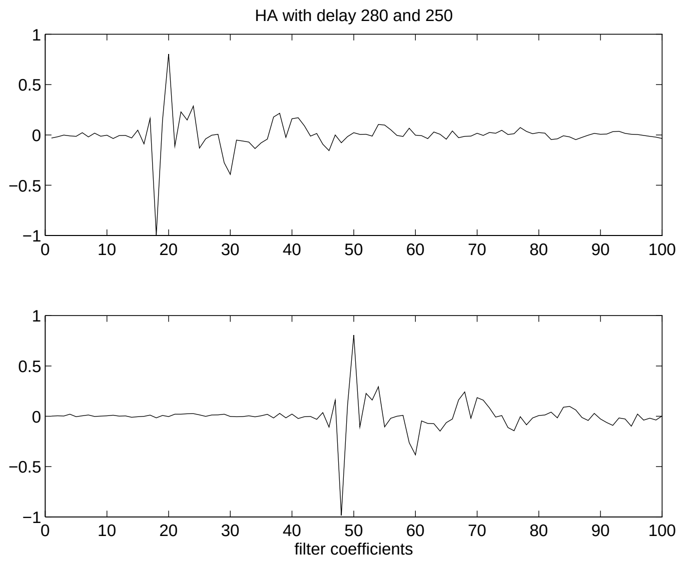


Figure 7: HA algorithm with delay 280 (top) and 250 (bottom)

- The `prof -pixie` command gave us profiling information.

We found, as expected, that most of the CPU cycles were used to execute two `for` loops, one computing the output of the system, the other updating the filter coefficients. These two loops are shown below. The C comments give the number of cycles executed in each line over the total number of cycles executed by the program.

```
/* update output of system */
for (hi=0,n=start;hi<filterlen;hi++, n++) {
  if (n==MAXLEN) n=0;                        /*    2.67%   */
  y += h[hi] * x[n];                         /*   44.10%   */
}
  .
  .
  .
/* update the filter coefficients */
for (hi=0,n=start;hi<filterlen;hi++,n++) {
  if (n==MAXLEN)n=0;                         /*    2.67%   */
  h[hi] += u * x[n];                         /*   49.43%   */
}
```

In the case of the HA algorithm, we observed that these two loops took 98.87% of the total number of cycles executed.

4.4.2 Bench Mark Results

The performance test program counted the number of CPU cycles executed to compute the output of the system and update the filter coefficients. To do this, we computed an average number of cycles per tap for filters of length 17, 33, 49,...,1025 and the average of these results was defined as the output of the bench mark.

We ran the test on three systems:

- A DEC 3000 Model 400 workstation, 21064 processor running at 133 MHz
- A DEC 3000 Model 700 workstation, 21064A processor running at 225 MHz
- A DEC 3000 Model 900 workstation, 21064A processor running at 275 MHz

	LMS	NLMS	HA	IA
DEC 3000 Model 400	19	20	20	49
DEC 3000 Model 700	19	19	19	48
DEC 3000 Model 900	19	19	19	48

Table 2: Cycles per tap

Table 2 gives the results of the bench mark in number of cycles per tap while Table 3 gives the computation time in microseconds for a 500 tap filter[8].

	LMS	NLMS	HA	IA
DEC 3000 Model 400	72	75	75	184
DEC 3000 Model 700	42	42	42	107
DEC 3000 Model 900	35	35	35	87

Table 3: Computation time for a 500 tap filter (in microseconds)

As depicted in Table 2, the IA algorithm costs 2.5 times more in executed cycles than the other algorithms. The extra computation cost is incurred because the compiler did not unroll the `for` loop updating the filter coefficients. If the compiler

[8]Note: if x is the number of cycles per tap and S the CPU clock speed in Hz, a 500 tap filter takes $500xS^{-1}$ seconds to compute. For a sampling rate of 8000 Hz, the filter update and output computation must not exceed 125 microseconds.

unrolled the loop below, or if it were unrolled by hand, then the computation time for IA would be closer to HA.

```
/* update the coefficients */
for (hi=0,n=start;hi<filterlen;hi++,n++) {
  if (n==MAXLEN)n=0;                         /*   7.68%   */
  tmp = x[n]; /* make it a local variable */ /*  11.52%   */
  if ((bufx1 = tmp) < 0) bufx1 = -bufx1;     /*   7.68%   */
  h[hi] += u * bufx1 *  tmp;                 /*  38.42%   */
}
```

5 Conclusion

In this paper, we described the identification process which leads to the LMS algorithms and the recursive least square algorithms. We presented an optimized real-time Echo-Cancelation Software Lab which allows testing of the LMS, normalized least mean square, homogeneous adaptation, and individual adaptation algorithms. We also presented the results of a bench mark measuring the number of cycles per tap for each of these four algorithms on three different DEC 3000 Model workstations. With this Software Lab, we found the homogeneous adaptation algorithm to produce the best results.

Acknowledegements

We would like to acknowledge Andrew Payne for implementing the LMS algorithm in a first version of the software testbed, as well as for writing the prototype Tk graphical user interface.

We also would like to thank Dick Beane, Larry Stewart, and Dave Wecker for reviewing drafts of this paper, and providing us with valuable comments.

References

[1] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall, second edition edition, 1991.

[2] D.T.M. Slock and T. Kailath. Numerically stable fast transversal filters for recursive least squares adaptive filtering. *IEEE Transactions on Signal Processing*, 39(1):92–114, January 1991.

[3] B. Widrow, J.R. Glover, J.M. McCool, J. Kaunitz, C.S. Williams, R.H. Hearn, J.R. Zeidler, E. Dong, and R.C. Goodlin. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63(12), December 1975.

[4] W.B. Mikhael, F.H. Wu, L.G. Kazovsky, G.S. Kang, and L.J. Fransen. Adaptive filters with individual adaptation of parameters. *IEEE Transactions on circuits and systems*, cas-33(7), July 1986.

[5] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Professional Computer series, 1994.

[6] Thomas M. Levergood, Andrew C. Payne, James Gettys, G. Winfield Treese, and Lawrence C. Stewart. AudioFile: A network-transparent system for distributed audio applications. In *Proceedings of the USENIX Summer Conference*, June 1993.

[7] Sound and Motion J300 option. Digital Equipment Corporation Systems and Options Catalog. Available on the WWW at http://www.digital.com/info/Customer-Update/930816003.txt.html.

[8] C.W.K. Gritton and D.W. Lin. Echo cancelation algorithms. *IEEE ASSP Magazine*, pages 30–37, April 1984.

[9] A. Srivastava and A. Eustace. ATOM: A system for building customized program analysis tools. Technical Report 94/2, WRL, Western Research Laboratory 250 University Avenue Palo Alto, California 94301 USA, 1994.