Implementation of a digital adaptive filter

by

Thomas Aloysius Sexton

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Co-majors: Biomedical Engineering

Electrical Engineering

Approved:

Iowa State University
Ames, Iowa

1984

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Time invariant filters fail when the spectrum of the noise they are intended to filter changes (nonstationarity). If the noise lies within the signal bandwidth, as white noise would, simple bandpass or bandstop filters cannot eliminate the noise entirely.

Theoretically, an optimal Wiener filter can be applied if the signals are stochastic and stationary. Wiener filters are discussed in (1,2). However, this requires knowledge of the cross-correlation of the noisy signal with the true signal and the autocorrelation of the noisy signal. The filters discussed herein assume almost no previous knowledge of the signal. They approach the optimal response by adapting to the input data. Rather than realizing a specified frequency response, adaptive filters estimate a time-domain version of the noise in a signal and subtract it out. Adaptive filters do not use correlation functions, but they do require a reference noise input. Such data-dependent filters are nonlinear.

The purpose of this research has been to design, build and test an adaptive filter with a bandwidth suitable for biological signals. Since the highest bandwidth needed was only 100 Hz (electrocardiogram or ECG), no bandwidth problems were expected. But, due to the long 64-tap filter needed to cancel white noise, bandwidth was restricted to 30 Hz for this case. A bandwidth of 100 Hz was realizable for filtering an ECG polluted with 60 Hz power line noise.

An adaptive filter was built with the Motorola 68000 microprocessor for its processing unit. The 68000 is part of a microcomputer designed by

Motorola: the Educational Computer Board (ECB). The analog amplifier and filter, analog to digital converter (ADC) and digital to analog converter (DAC) circuits were designed by the author.

The filter was programmed in assembly language and stored on cassette tape. The ECB supports debugging with trace and breakpoint commands. No source code is stored; the firmware assembles each line as it is entered using an interpreter. This precludes use of variable and address names and comments.

This thesis is divided as follows: background, method, results, discussion, and conclusion. The background section covers the two kinds of filters considered, the Adaptive Gradient Lattice (AGL) and the Least Mean Square (LMS) filter. The LMS was ultimately chosen for implementation. Other promising filter structures are also mentioned. The method section gives block and flow diagrams of the different parts of the filter along with a discussion of bandwidth, scaling and noise considerations. The hardware block diagram gives the reader an overview of the data acquisition system used. The results section discusses filter performance for the various signals tested, sinusoidal and broadband. That section contains photographs of filtered signals with corresponding impulse responses. The discussion and conclusion sections review the strengths and weaknesses of the filter.

BACKGROUND

A noise canceler can be used to filter a signal when the source of noise in a signal is known. The heart of a noise canceler is a linear estimator. A least squares estimate of the noise in the signal channel is made by the linear estimator operating on a noise reference.

If the noise is not stationary, then a noise canceler which can adapt is needed. An adaptive algorithm is needed to do adaptive noise cancelling. Two gradient search algorithms are discussed here, the Adaptive Gradient Lattice filter (AGL) and the Least Mean Square filter (LMS). The terms algorithm and filter are synonyms when considering digital filters. Ultimately, the LMS filter was chosen for real-time capability considerations.

Noise Canceler

Two input signals are required for the noise canceler (Figure 1):

1) the noisy information signal, s + n, where
   the desire is to isolate s, and
2) a reference signal, x, which is correlated
   with n but not with s.

The box labeled "W" is properly called the adaptive filter while the entire system is referred to as a noise canceler. W operates on "future" and "past" values of x in order to estimate the present value of n. In filter theory, W is generally known as a linear estimator. The estimate of n is called y. Subtracting y from s + n gives an "error" signal, e. The nomenclature, e, is a pseudonym from control theory. The canceler does not drive the "error" to zero, but rather e approaches the

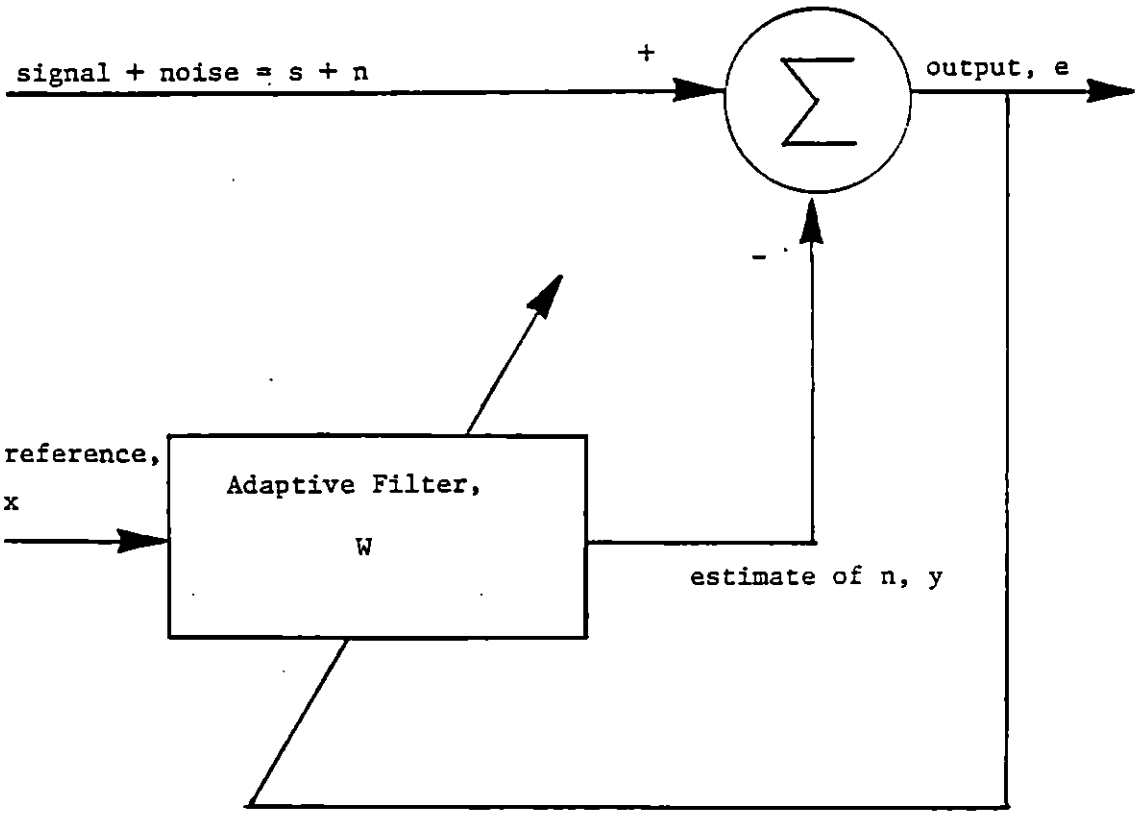


Figure 1. Adaptive noise canceler

information, s.

Some ingenuity is required to obtain the reference signal, x. Obviously, x cannot be set to equal n. If n were known exactly, it could be simply subtracted out. However, the source of n is often known, and if so, the source can be tapped for a signal x. If x arises from the same source as n, then they will be correlated. Care must be taken to acquire x without picking up the information signal, s. If a signal related to s finds its way into x, then the noise canceler will not cancel n exactly, and some compromise will be reached where the error power is a minimum.

Linear estimator

W operates on x to produce a least squares estimate of n. This point is discussed here and its veracity should become apparent in the derivation of the LMS algorithm; the AGL algorithm will not be derived since it was not used.

Following the development in (3) we have:

e = s + n - y. The mean square of e is

E(e**2) = E(s**2) + E((n-y)**2) + 2E(s(n-y))

E denotes the expectation operator and ** denotes "squared". s is orthogonal to both n and y. Hence,

E(e**2) = E(s**2) + E((n-y)**2).

The filter adapts to minimize E(e**2) but s is unaffected since it is an input:

min(E(e**2)) = E(s**2) + min(E((n-y)**2))

Minimizing the left-hand side makes y a least squares match to n in

the right-hand side.

If the initial relationship is rewritten as e - s = n - y, it can be seen that if y is made to match n, then e becomes a least squares estimate of s.

Thus, the noise canceler operates on s + n and x to produce a least squares estimate of s.

Example

An example of LMS filter operation from the results section is included here to add clarity. Figure 2 shows the test set-up. The information signal, s, is a sine wave, while the corrupting noise, n, is bandlimited white noise. Experiments with n not equal to x were also conducted as discussed later. After the filter runs for a few seconds, W converges to the impulse response shown in Figure 3. W weights the future, present and past values of x to estimate the present value of n. The estimate is y. If the autocorrelation of n is low, then W tends to emphasize only those samples of x near in time to the present. Some values of x are in the future compared to s + n because the latter is delayed.

Digital filters are defined by sets of equations. Several different filter structures are possible to realize a given set of equations. Each realization is mathematically equivalent, but each will behave differently under quantization, some doing better than others (3). Either the LMS or AGL filter may be used to implement noise cancelling. Further explanation of the LMS filter is given in the section on the derivation of the LMS algorithm.

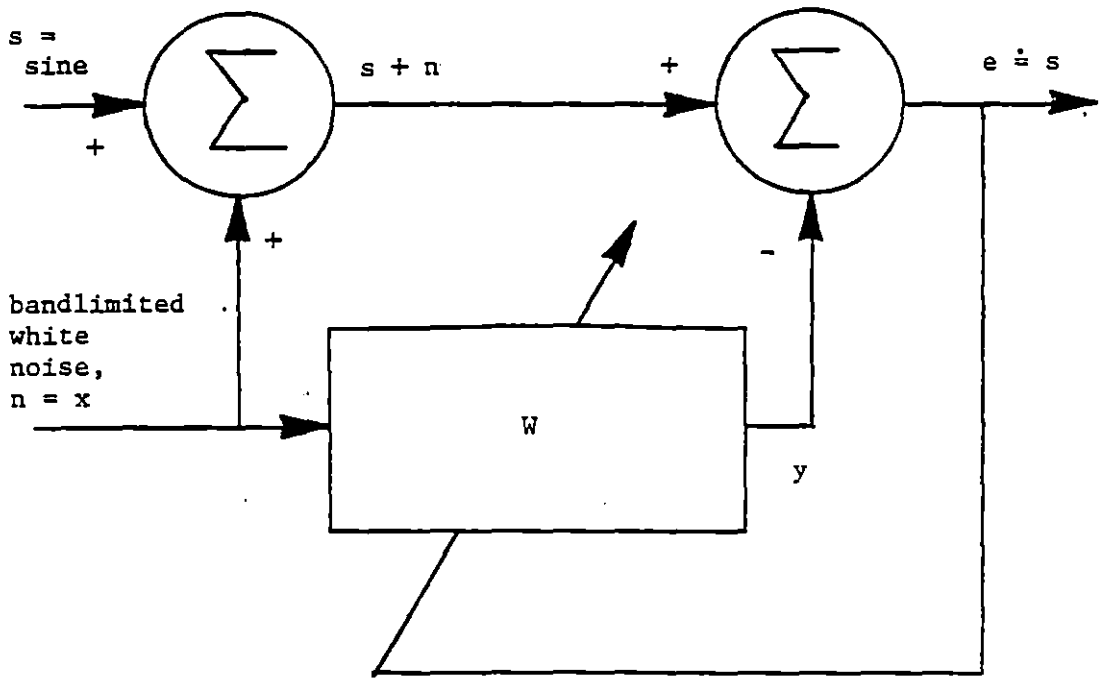


Figure 2. Test setup of example

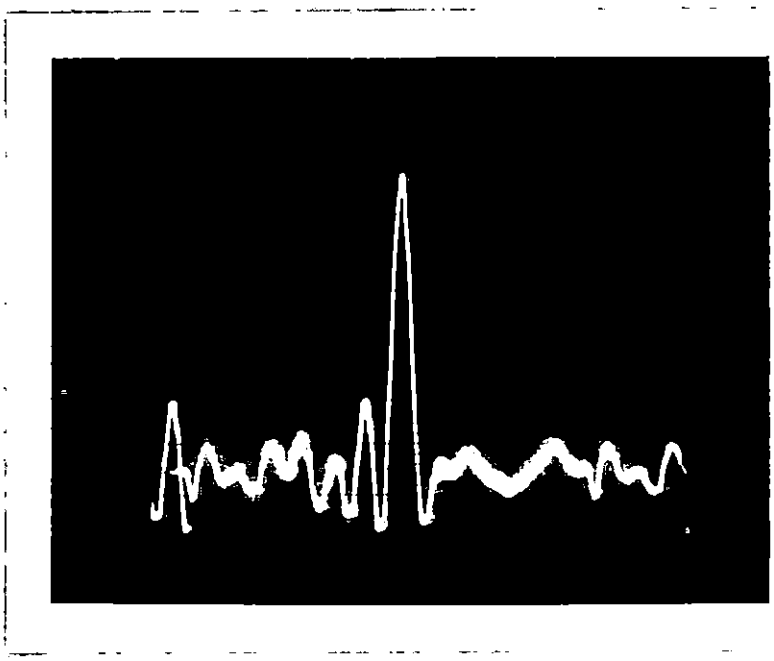


Figure 3. Impulse response of example

Candidate Filters

The salient points of the LMS and AGL filters with respect to implementation are now considered.

Least Mean Square filter

The LMS filter structure is given in Figure 4. It is a realization of the Widrow LMS algorithm (4). Note the familiar tapped delay line or transversal filter structure. The multiply and sum configuration is the direct implementation of the vector inner product taken in the LMS algorithm. The creation of the vector X used in the forthcoming derivation from the scalar sampled x is evident in the figure. A given value of the sampled x is multiplied successively by each element of W. If X(transpose) = (1 0 0 ... 0), y is the impulse response, and W is that response. The elements of X multiplied by w1, w2, ... w(N/2)-1 occur in time before the value of s + n used at the summer to produce e. Thus, they are "future" values of X.

Adaptive Gradient Lattice filter

The lattice filter structure is a signal flow graph of the Levinson recursion (2,5,6). Figure 5 shows one portion of a lattice filter.

A given reflection coefficient, k(i), is equal to the autoregressive parameter which is the ith coefficient in an ith order linear predictor. "Linear predictor" refers to the set of coefficients a(j) used to represent an autoregressive process, viz.

$$x'(t) = \sum_{j=1}^{N} a(j)x(t-j)$$

where prime refers to estimate and t is the time index.

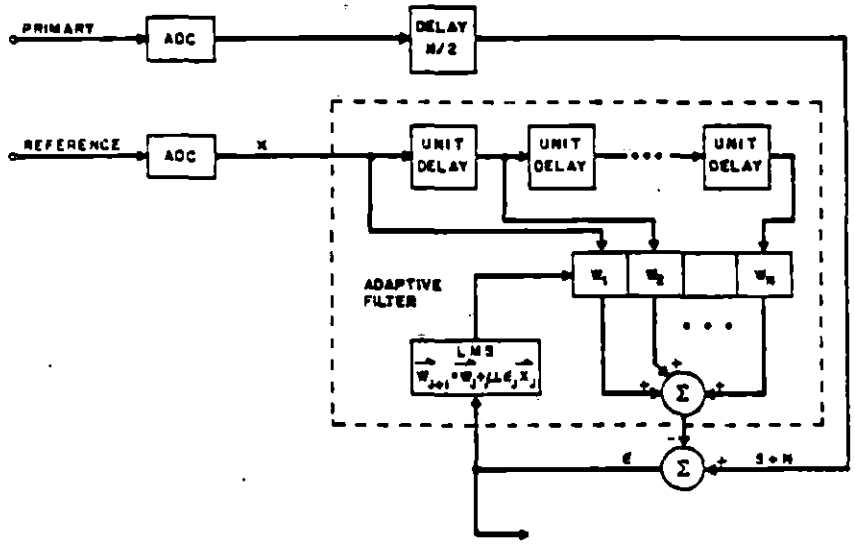


Figure 4. LMS signal flow graph (Taken from (3))

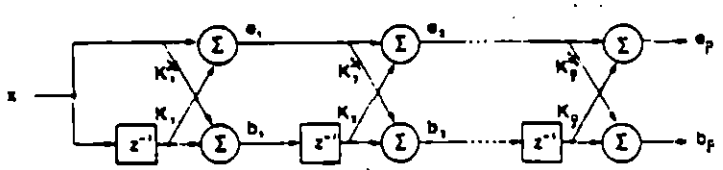


Figure 5. Lattice structure (This is a portion of a noise canceler using the Adaptive Gradient Lattice (taken from (6)).)

The forward error is

$$e(t) = x(t) - x'(t).$$

The a(j) can be reversed in time to "predict" a past value:

$$x'(t-N) = \sum_{j=1}^{N} a(j)x(t-N+j)$$

The backward error is

$$b(t) = x(t-N) - x'(t-N).$$

See (7) for more details on linear prediction. The lattice transforms x into a set of orthogonal variables, b. The portion of b which is correlated with s + n is subtracted from the latter giving an estimate of s (8). Lattice theory will not be delved into further due to its complexity and the fact that it wasn't used.

During filter operation, s + n and x are measured at discrete intervals. The filter transfer function is updated, or adapted, between measurements. The update takes a certain amount of time. The number of multiplications needed per update by the AGL filter is 7N, while it is only 2N for the LMS filter (8,9). Long update times limit the sampling rate. By the well-known Nyquist criterion, bandwidth is one-half of the sampling rate. Given a certain amount of computational speed therefore, the AGL filter will have only about one-third the bandwidth of the LMS filter. This is the weakest point of the AGL filter and the strongest point of the LMS filter.

The AGL filter can be designed with no assumptions to guarantee stability. A worst-case design of the LMS filter requires some knowledge of the eigenvalues of the noise correlation matrix.

Decision to use the LMS algorithm

The crucial point favoring the Widrow LMS algorithm is that it is approximately 3 1/2 times faster than the AGL algorithm (9). One of the primary purposes of this work has been to create a real-time filter using a microcomputer. The AGL filter requires too many operations to do in real time and still have any useful bandwidth. If an array processor had been used, the multiply and sum bottleneck inherent in digital filtering would not have been such an important consideration. However, this research was restricted to using a readily available microcomputer.

The transversal filter structure used in implementing the LMS filter is easy to adapt. However, it suffers greatly from the inaccuracys caused by finite word length arithmetic and coefficient quantization (10). Reference (11) presents filter structures which are less sensitive to coefficient quantization. However, the structures presented in (11) may not be useful as adaptive filters. The difficulty lies in finding a filter structure the performance of which is not too sensitive to small inaccuracies in the filter coefficients and whose coefficients can be rapidly updated.

Most recently Cioffi and Kailath (12) have introduced a fast transversal filter (FTF). The LMS is considered suboptimal since it resorts to a gradient technique to approach the optimal solution, but the FTF is a "true least squares" or optimal algorithm.

Derivation of the LMS Algorithm

The LMS algorithm is derived here starting with the statement of the best linear estimator problem (13).

The data vector x (see Figure 6) is to be weighted by a filter vector W and summed to produce an ouput y. W should give a y which is a minimum square error estimate of a desired signal, d.

Note that W is a row vector: W = (w0 w1 w2 ... wN). X is a column vector (the transpose is shown here): X(transpose) = (x0 x1 x2 ... xN).

In the following development, e**2 denotes "e squared",and 2*e denotes "2 times e", and E is the expectation operator.

From the figure:

e = y - d and y = WX.

We seek to minimize E(e**2).

e**2 = (y - d)**2 = (WX - d)**2

= (WX)**2 - 2dWX + d**2

Taking expectations, the function to be minimized is

F = E(e**2) = E((WX)**2 - 2dWX + d**2).

Exact determination of the W needed to minimize F is done using a calculus of variations approach (1,13). The resulting W is called the optimal Wiener filter. Adaptive processing diverges at this point from the exact solution. Avoiding computation of the correlation functions needed for the exact solution is necessary so that the filter can run in real time.

The function F forms a quadratic surface in N + 1 space which has the shape of a bowl. The terms shape and bowl are used loosely here since they usually refer to a space with three or fewer dimensions. Figure 7 is an example in 3-space taken from the tutorial discussion in (4). The method of steepest descent is used to determine the coordinates of the

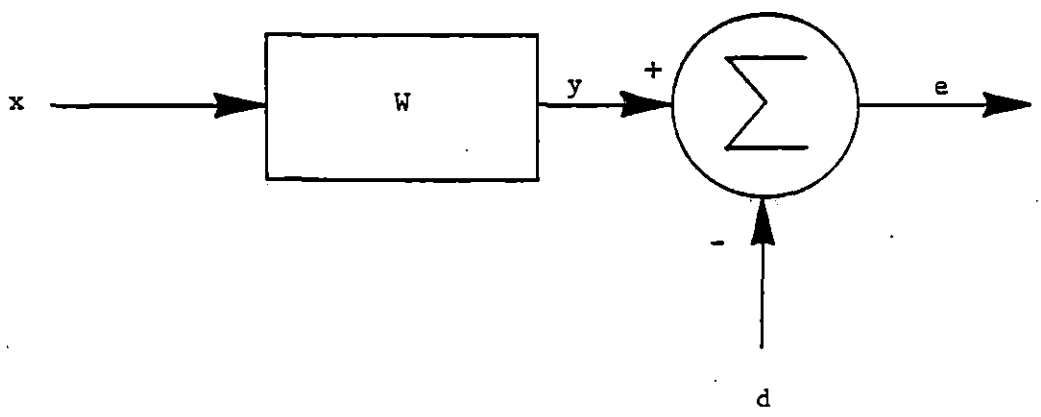
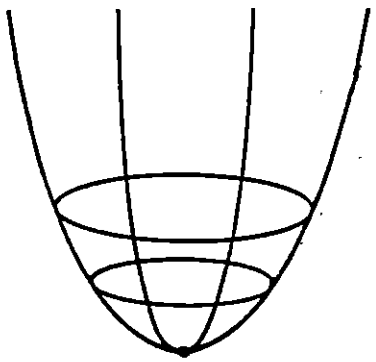


Figure 6. Linear estimator



The error surface can be viewed as an (N+1) dimensional paraboloid where N is the number of taps in the filter. The gradient always points opposite to the bottom. The steepest descent method goes opposite to the gradient direction to find the minimum error point (Taken from (8)).

Figure 7. Error surface

bottom of the bowl (4,8). This point corresponds to the minimum value of F and hence the optimum value of W.

The optimum value of W can be attained by iteration:

1) Any initial guess is made for W like (0 0 0 ... 0).

2) The slope (gradient), G, at W on the F surface is computed by:

$$G(j) = \begin{bmatrix} \frac{\partial(F(j))}{\partial(w0)} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial(F(j))}{\partial(wN)} \end{bmatrix}$$

where j is the time index.

3) W is updated with the gradient multiplied by a step size factor, u, which controls stability and rate of convergence.

$W(j+1) = W(j) - uG(j)$

4) Return to 2) until G(j) = 0, or, in other words, the minimum value of F has been reached.

Computing an expectation or even an estimate of an expectation doesn't lend itself to real-time applications. The expectation of a variable is found by integrating the variable times its probability density function over the range of the variable (1). This implies considerable knowledge of the variable. To make the problem workable, the expectation operator is dropped from the expression for F. Then, an approximate gradient can be

found, G'(j).

$$G'(j) = \begin{bmatrix} \frac{\partial(e^{**}2(j))}{\partial(w0)} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial(e^{**}2(j))}{\partial(wN)} \end{bmatrix} = -2e(j) \begin{bmatrix} \frac{\partial(e(j))}{\partial(w0)} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial(e(j))}{\partial(wN)} \end{bmatrix}$$

The noise canceler is configured so that

$e(j) = s(j) - W(j)X(j)$.

Hence,

$G'(j) = -2e(j)X(j)$, and the iteration becomes

$W(j+1) = W(j) + 2ue(j)X(j)$, which is the LMS algorithm.

Dependence on eigenvalue W must have at least as many elements as there are nonzero eigenvalues in the autocorrelation matrix of x (4). If a given eigenvalue is small, it will cause F to have a shallow surface along one dimension of W. It takes longer for the LMS algorithm to descend along a shallow surface than a steep one. Therefore, small eigenvalues extend the time needed for the LMS algorithm to converge. On the other extreme, large eigenvalues contribute steep slopes to F and limit stability. For the LMS algorithm to be stable (4), the step size, u, must be less than or equal to the reciprocal of the largest eigenvalue. Eigenvalue calculations were not done for this research.

METHOD

The following sections give an overview of the filter implementation.

Hardware

Figure 8 is a block diagram of the data acquisition and analog output circuits with the microcomputer given as a single block.

Block organization

ADC protection Standard voltage limiting zener diode circuits are used to protect the ADC inputs. In order to avoid damage, the ADC inputs are not allowed to exceed the DC reference voltages supplied.

Decoding The ADC and DAC are memory mapped. This means that control lines and registers were assigned to specific addresses. The particular assignments are discussed in the Program section. The decoding circuitry is a simple collection of two-input AND gates.

Reference voltages Reference voltages are obtained by placing resistors and zener diodes in series. Making the references as noise free as possible is necessary to avoid measurement error.

The other circuits can all be found in a text like Milman (14).

Automatic Gain Control (AGC)

The instrumentation amplifiers are equipped with AGC in order to maintain maximum use of the finite word length available. The dynamic characteristic of drain to source resistance with respect to gate voltage of a field effect transistor (FET) was used. The gate voltage is obtained by the author's circuit as shown in Figure 9. The peak value of Vout is fed to the amplifier via the follower. The zero and span of the AGC ouput signal, Vc, are controlled by potentimeters P1 and P2 respectively.

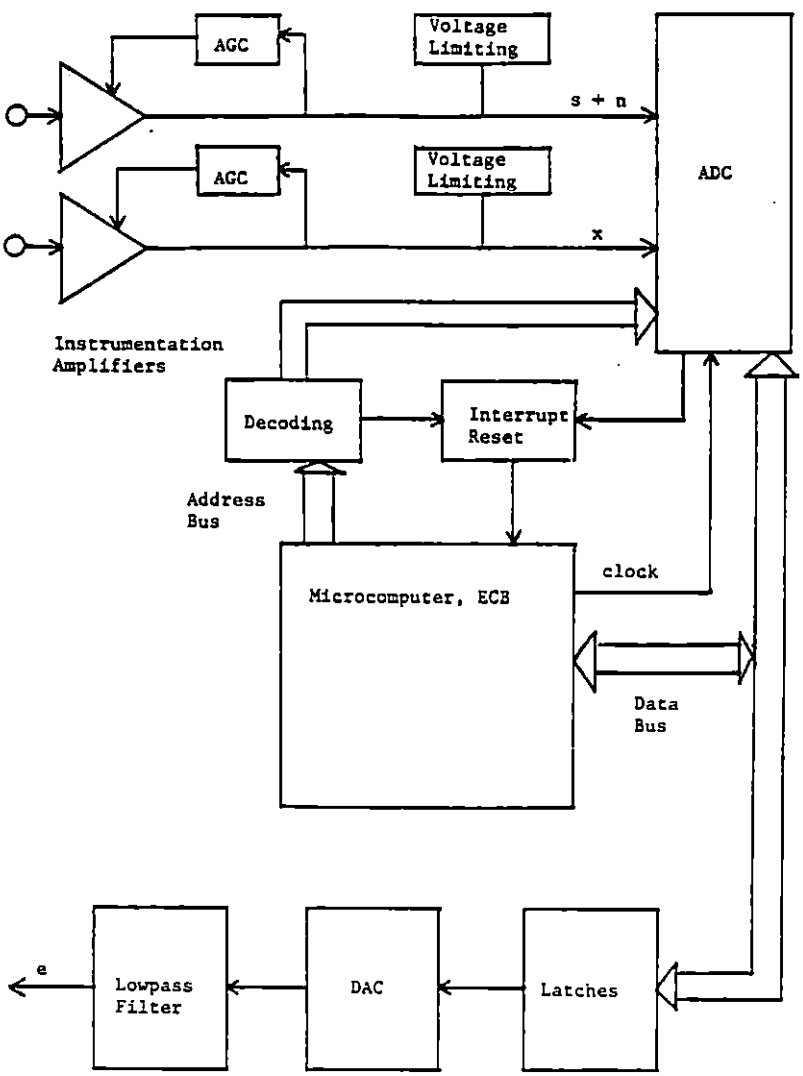


Figure 8. System block diagram

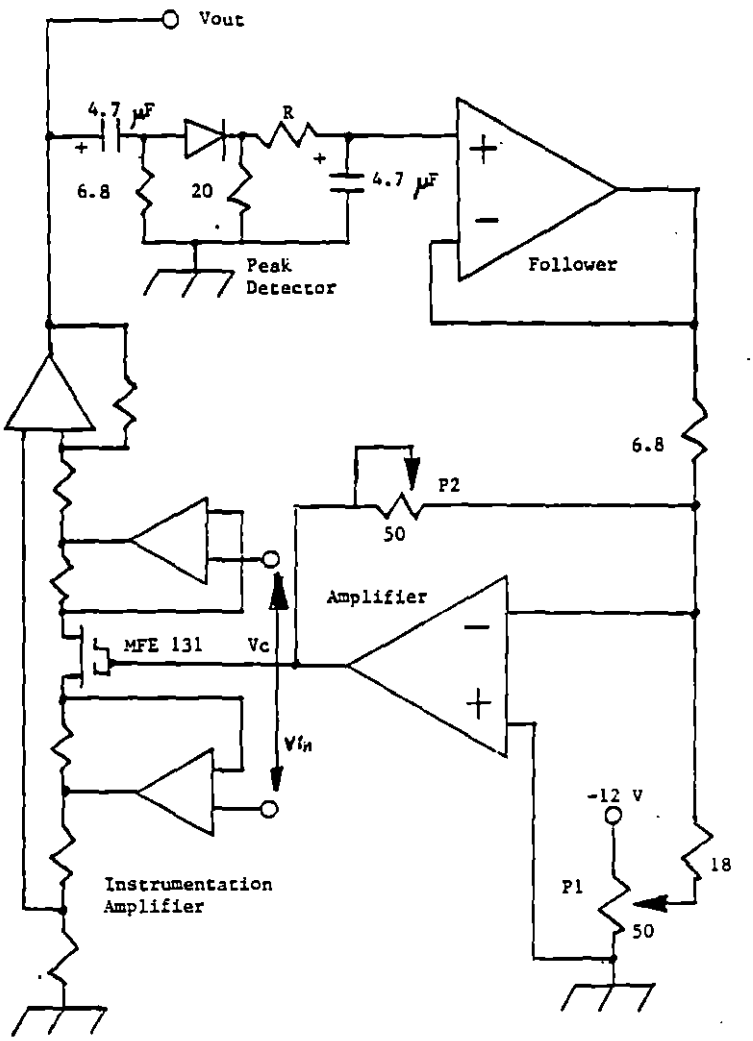


Figure 9. AGC circuit

Program

The following paragraphs cover the details of the program.

Flow graphs

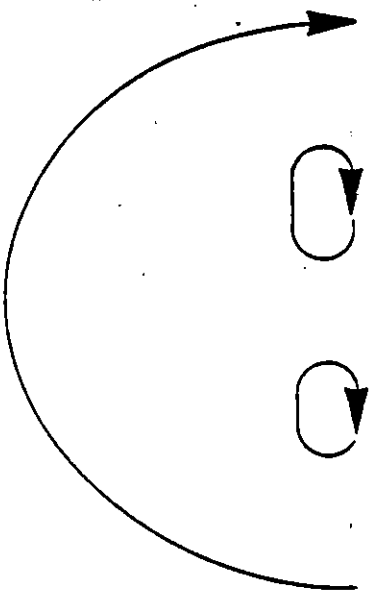
Figure 10 gives the step by step arithmetic done in the program to implement the LMS algorithm. The program consists of two loops. The first loop updates the W vector; the second loop computes the estimate, y.

Careful scaling is needed in fixed word length arithmetic. In other words, keep all the bits you can as long as you can. Figure 11 illustrates scaling in the program flow. The word lengths used are optimal, given that s and x are only 8 bits long. A limitation on this system is the use of an 8 bit ADC. Finite precision problems could be mitigated by sampling at faster rates as this would give a finer representation of the signal to the filter. However, this would require more computing power.

Code, overhead

Figure 12 is the disassembled object code used for the 64 point filter. The code is broken into four sections: initialization, measurement of s + n and x, W update, and computation of e.

Significant overhead in the program lies in keeping track of the circular queues of s + n, x and W. Each time a variable is recalled the queue pointer must be tested to see if it should be reset to the begining of the queue. However, the multiplication executed in each loop consumes the most time of any given single instruction. A 16 by 16 bit multiplication takes about 23 ms.



Operation	Program Step
Measure s and x	1
Update W by summing the vectors W and eX; for i = 1 to N:	
Compute $e*x_i$	2
Compute $u*(ex_i)$	3
Compute $w_i + uex_i$	4
Find vector inner product of W and X; for i = 1 to N:	
Compute $w_i x_i$	5
Compute $y = y + w_i x_i$	6
Find output:	
Compute $e = s - y$	7
Output e	8

Figure 10. Program flow graph (Notation: X - vector, xi - vector element)

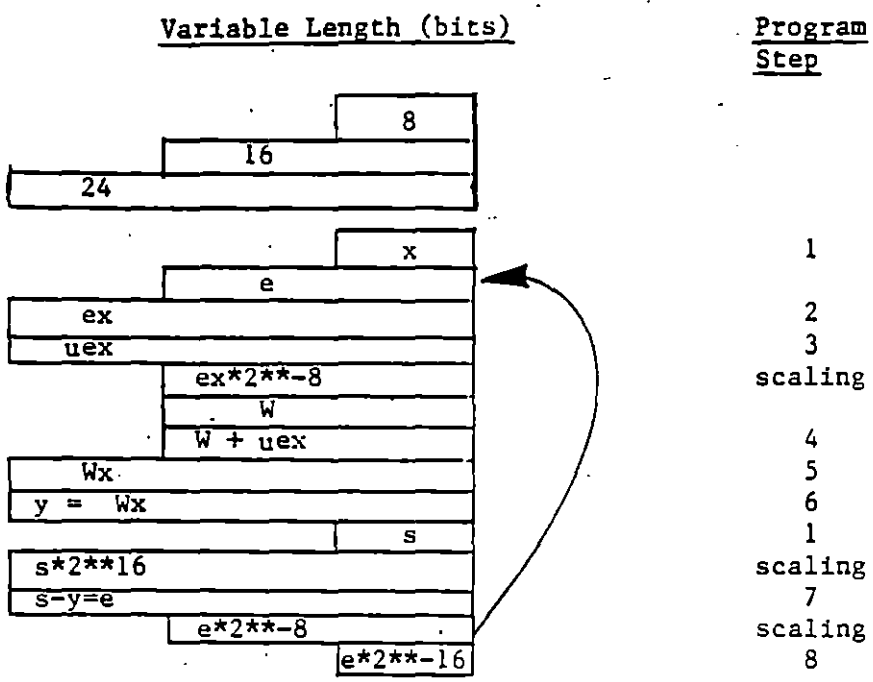


Figure 11. Scaling flow graph

```
001140    1011                  MOVE.B  (A1),D0
001142    42780B5C              CLR.W   $00000B5C
001146    0A7800FF0B5C          EOR.W   #255,$00000B5C
00114C    4E73                  RTE
00114E    0000                  DC.W    $0000
001150    247C00039001          MOVE.L  #233473,A2
001156    207C0003A001          MOVE.L  #237569,A0
00115C    227C0003C001          MOVE.L  #245761,A1
001162    267C00036001          MOVE.L  #221185,A3
001168    203C00000100          MOVE.L  #256,D0
00116E    287C00000A00          MOVE.L  #2560,A4
001174    4254                  CLR.W   (A4)
001176    D9FC00000002          ADD.L   #2,A4
00117C    51C8FFF6              DBF.L   D0,$001174
001180    16BC0002              MOVE.B  #2,(A3)
001184    1210                  MOVE.B  (A0),D1
001186    42780B5C              CLR.W   $00000B5C
00118A    0C7800000B5C          CMP.W   #0,$00000B5C
001190    67F8                  BEQ.S   $00118A
001192    223C00000007          MOVE.L  #7,D1
001198    0300                  BTST    D1,D0
00119A    56C9FFFC              DBNE.L  D1,$001198
00119E    06010001              ADD.B   #1,D1
0011A2    E341                  ASL.W   #1,D1
0011A4    11C10B58              MOVE.B  D1,$00000B58
0011A8    287C00000A00          MOVE.L  #2560,A4
0011AE    2A7C00000A20          MOVE.L  #2592,A5
0011B4    2C7C00000A60          MOVE.L  #2656,A6
```

The format of the disassembly is: memory location on the left, followed by object code and finally the assembly code. The above section of code includes the brief interrupt routine (lines $1140-114C), the begining of the program ($1150), initialization of address registers (A0-A6), and zeroing of the data buffers (via the code of lines $1168-117C). Lines $1180-11A4 were initially used to read a potentiometer to set the step size, u. A data register (D7) is now loaded by the operator with the desired value of u before program execution.

Figure 12. Program disassembly

```
0011BA    16BC0000          MOVE.B  #0,(A3)
0011BE    1210              MOVE.B  (A0),D1
0011C0    42780B5C          CLR.W   $00000B5C
0011C4    0C7800000B5C      CMP.W   #0,$00000B5C
0011CA    67F8              BEQ.S   $0011C4
0011CC    04000080          SUB.B   #128,D0
0011D0    B9FC00000A20      CMP.L   #2592,A4
0011D6    6606              BNE.S   $0011DE
0011D8    287C00000A00      MOVE.L  #2560,A4
0011DE    18C0              MOVE.B  D0,(A4)+
0011E0    16BC0001          MOVE.B  #1,(A3)
0011E4    1210              MOVE.B  (A0),D1
0011E6    42780B5C          CLR.W   $00000B5C
0011EA    0C7800000B5C      CMP.W   #0,$00000B5C
0011F0    67F8              BEQ.S   $0011EA
0011F2    04000080          SUB.B   #128,D0
0011F6    B8FC00000A60      CMP.L   #2656,A5
0011FC    6606              BNE.S   $001204
0011FE    2A7C00000A20      MOVE.L  #2592,A5
001204    1AC0              MOVE.B  D0,(A5)+
001206    B8FC00000A60      CMP.L   #2656,A5
00120C    6606              BNE.S   $001214
00120E    2A7C00000A20      MOVE.L  #2592,A5
001214    1A86              MOVE.B  D6,(A5)
```

The above code corresponds to step 1 of Figure 10. Measurement of s begins with enabling the ADC (lines $11BA and 11BE). Then the program waits for the interrupt routine ($1140-114C) to set flag $B5C ($11C4, 11CA). The measured value is placed in data register D0 by the interrupt routine. This value is scaled for the proper sign bit at line $11CC. Before storing the value in the s queue (the s queue is pointed to by A4), a check is made to see if the end of the queue has been reached (lines $11D0-11D8); the value value is finally stored by execution of line $11DE.
A similar procedure is done to measure and store x (lines $11E0-1204, the x queue is pointed to by A5). Lines $1206-1214 provide for insertion of a constant into the sequence of measured x values. The constant is placed in D6 by the operator before execution.

Figure 12 (continued)

```
001216    263C0000003F          MOVE.L   #63,D3
00121C    32380B5A              MOVE.W   $00000B5A,D1
001220    B8FC00000A60          CMP.L    #2656,A5
001226    6606                  BNE.S    $00122E
001228    2A7C00000A20          MOVE.L   #2592,A5
00122E    101D                  MOVE.B   (A5)+,D0
001230    B0FC00000AE0          CMP.L    #2784,A6
001236    6606                  BNE.S    $00123E
001238    2C7C00000A60          MOVE.L   #2656,A6
00123E    4880                  EXT.W    D0
001240    C1C1                  MULS.W   D1,D0
001242    E080                  ASR.L    #8,D0
001244    6B06                  BMI.S    $00124C
001246    EE60                  ASR.W    D7,D0
001248    D15E                  ADD.W    D0,(A6)+
00124A    6010                  BRA.S    $00125C
00124C    EE60                  ASR.W    D7,D0
00124E    640A                  BCC.S    $00125A
001250    0C40FFFF              CMP.W    #-1,D0
001254    6604                  BNE.S    $00125A
001256    303C0000              MOVE.W   #0,D0
00125A    D15E                  ADD.W    D0,(A6)+
00125C    51CBFFBE              DBF.L    D3,$00121C
001260    4283                  CLR.L    D3
```

This section of code carrys out the adaption of W. D3 is the loop counter and $B5A holds the previous value of e. Program step 2 (Figure 10) corresponds to lines $1220-1240, step 3 to $1246 or $124C, and step 4 to $1248 or $125A. The test for a minus result ($1244) was necessary, as without it, 1/2 rounds off to 0, but -1/2 rounds off to -1 in twos-complement arithmetic.

Figure 12 (continued)

```
001262    283C0000003F        MOVE.L   #63,D4
001268    B8FC00000A60        CMP.L    #2656,A5
00126E    6606                BNE.S    $001276
001270    2A7C00000A20        MOVE.L   #2592,A5
001276    101D                MOVE.B   (A5)+,D0
001278    B0FC00000AE0        CMP.L    #2784,A6
00127E    6606                BNE.S    $001286
001280    2C7C00000A60        MOVE.L   #2656,A6
001286    321E                MOVE.W   (A6)+,D1
001288    4880                EXT.W    D0
00128A    C1C1                MULS.W   D1,D0
00128C    D680                ADD.L    D0,D3
00128E    51CCFFD8            DBF.L    D4,$001268
001292    B9FC00000A20        CMP.L    #2592,A4
001298    6606                BNE.S    $0012A0
00129A    287C00000A00        MOVE.L   #2560,A4
0012A0    1214                MOVE.B   (A4),D1
0012A2    48C1                EXT.L    D1
0012A4    E181                ASL.L    #8,D1
0012A6    E181                ASL.L    #8,D1
0012A8    9283                SUB.L    D3,D1
0012AA    E081                ASR.L    #8,D1
0012AC    31C10B5A            MOVE.W   D1,$00000B5A
0012B0    E081                ASR.L    #8,D1
0012B2    3601                MOVE.W   D1,D3
0012B4    11C30B59            MOVE.B   D3,$00000B59
0012B8    06030080            ADD.B    #128,D3
0012BC    1483                MOVE.B   D3,(A2)
0012BE    4EF811BA            JMP.S    $000011BA
```

The ouput, e, is formed by the above instructions. Program step 5 corresponds to $1268-1284, and 6 to $128C. After some scaling maneuvers ($12A2-12A6), step 7 is carried out in line $12A8. The 16 bit value of e to be fed back is stored in memory (line $12AC). An 8 bit representation of e is sent to the DAC (step 8: lines $12B0- 12BC). Control is then passed back to the point at which new values of s + n and x are measured.

Figure 12 (continued)

Using the ADC and DAC

To obtain an input from an ADC input channel, 3 steps occur:

1) select the channel by writing a number 0-7 to $36001,
2) signal the ADC to begin conversion by putting $3A001 on the address bus (read or write), and
3) after the ADC signals end of conversion (via the interrupt routine setting ($B5C)=$FF), read $3C001 to obtain the 8 bit value.

The DAC is written to by writing the digital ouput value to $39001.

Bandwidth and Filter Length

Bandwidth is determined by filter length N. The number of multiplications needed per update is 2N. The update time is inversely related to the highest frequency which can be sampled without aliasing.

The kind of noise to be cancelled determines N. This follows in that the reference is filtered in order to duplicate the signal noise. A predictive filter works weakly on signals which are not orthogonal over the window it looks at (N is the size of the window). If the signal is a single sine, only a short window is needed. If the signal is a sum of sines, a longer window is needed.

Noise Variables, H(z)

Figure 13 gives a view of the most general noise cancelling situation possible. This figure is found in (4). m1 and m2 are uncorrelated with each other and s and n. They are additional noise signals which limit filter performance. This research is not completely general in that m1 and m2 were not purposely added in as signals. However, they are

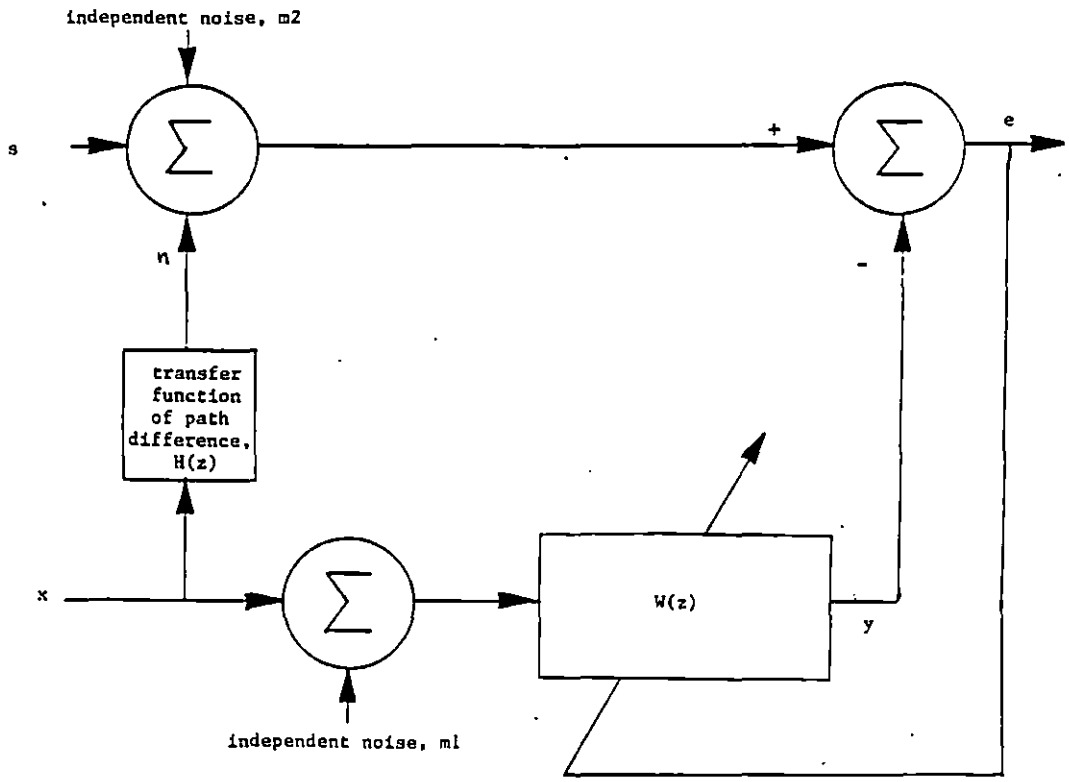


Figure 13. General noise canceler

unfortunately present anyway as quantization error.

Almost all of the results herein correspond to H(z)=1 since this is the simplest case. Elaborate programming is needed to complicate H(z). This was done just once; H(z) = .5z**-8 + .5z**+8 was tested. The signal to be cancelled was a single sine or white noise. About 20 dB improvement in signal to noise ratio was achieved giving hope for use of the filter in nonideal situations where H(z) is not unity.

RESULTS

The following sections discuss and exhibit results of the experimental work.

Signals Tested

Three general waveforms were used to generate test signals: sinusoids, ECGs and broadband noise. The noisy and filtered signals are shown in Figures 14, 15 and 16. Signal to noise ratio (SNR) improvement is given in Table 1.

SNR measurement for the random noise rejection case (figure 16) was not done accurately. The problem extends back to the limited processing speed of the general purpose microcomputer used. Because the noise was white a long filter was needed: N=64. Because bandwidth is inversely related to N, the bandwidth was cut down to .1 - 30 Hz. This causes a conflict with the available voltmeters as they are not suitable for this low band. The mean in the rms measurement done by the voltmeter used is taken over too short a time for this band. If a faster processing system was used, the filter could operate on a frequency range with a higher upper limit.

Amplifier Performance on the ECG

Figure 15 gives filter performance on the ECG. An implicit point this picture shows is that the amplifier hardware works well. Most of the test signals were generated on the lab bench. Considerable testing, however, was done using strap electrodes from the author directly to the amplifiers. The transient nature of the R wave in the ECG caused unacceptable transient AGC behavior initially. Increasing the time

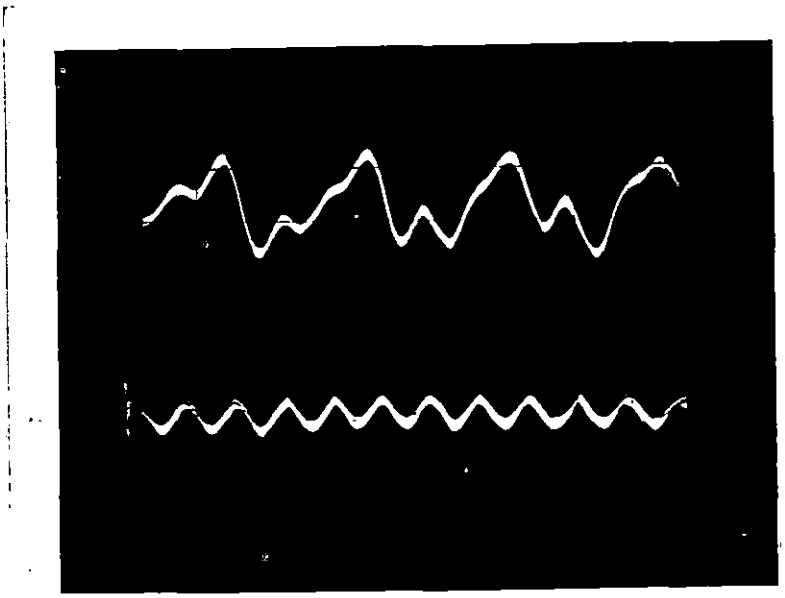


Figure 14. Sum of sines (The upper trace is a sum of three equiamplitude sines. The lower trace is the canceler output.)

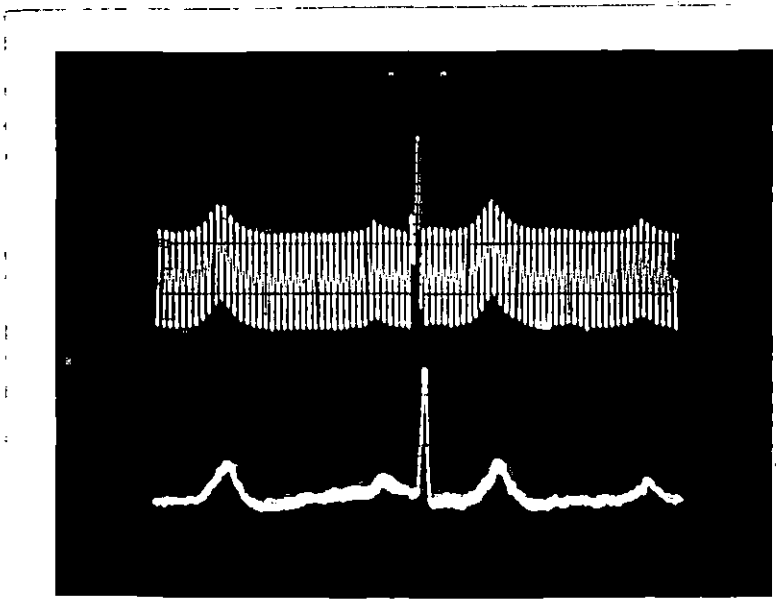


Figure 15. ECG + 60 Hz sine (The upper trace is a noisy ECG. The lower trace is the canceler ouput.)

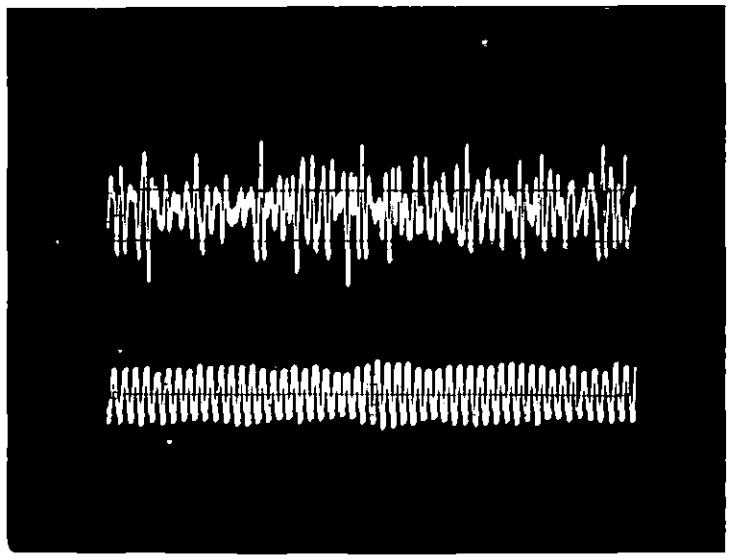


Figure 16. 30 Hz sine + white noise (The upper trace is a corrupted sine. The lower trace is the canceler output.)

Table 1. SNR improvement[a]

signal + noise components (s + n)	output (e) uncancelled	cancelled
sin(6.28*25t)	5 dBm	-15.5 dBm
sin(6.28*25t) + sin(6.28*17t) (where s is the 25 Hz signal)	SNR(power) = 1.16	SNR(power) = 176.6
sin(6.28*25t) + sin(6.28*17t) + sin(6.28*7t) (where s is the 25 Hz signal)	SNR(power) = .73	SNR(power) = 122.45

[a]The SNR improvement of the 25 Hz sine is about 41 dB for the latter two cases. However, low frequency components around 2 Hz make these values vary by +/- 1 dB. The 2 Hz variation is in itself an unaccounted-for noise. For that reason, the data in the first entry is used to give the nominal performance of the filter: 20.5 dB(power) cancelation of the noise signal.

constant of the AGC peak detecting circuit to several seconds solved the problem. This was done by increasing R of figure 8. The algorithm itself had no trouble with the ECG. As long as there is no ECG-correlated signal in the reference, the ECG will pass through the noise canceler.

Example Impulse Responses

Figures 17, 18, 19 and 20 give example impulse responses (IRs) of the filter after convergence. Figures 17 and 18 look very much like autocorrelation functions of the reference signals they operated on. The autocorrelation function of a sine is a cosine and of bandlimited white noise is sin(x)/x. The output of the filter after convergance is the convolution of its IR with the reference signal. In the case of Figures 17 and 18, H(z) is unity. Hence, the filter needs simply to reproduce its input. Convolving the autocorrelation function with the reference gives the most likely present value of the noise. This value is then subtracted from the noisy signal giving optimal noise reduction.

Figures 19 and 20 are for the case H(z) = .5z**-8 + .5z**+8. Figure 20 shows that the filter mimics this transfer function to produce useful cancellation. The IR of Figure 19 also produced noise reduction of about 20 dB. But, a transfer function like that of Figure 20 would also have cancelled a sine. Because the filter is data dependent, it only needs to "create" the frequency response which passes the portion of x corresponding to n. If n is a simple sinusoid of frequency f, then the the number of W's having the needed magnitude and phase response is infinite. It can be shown in the time domain that convolving a function like that of Figure 20 with a sine of frequency f gives another sine

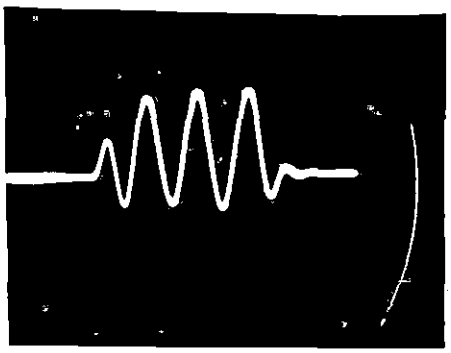


Figure 17. IR when cancelling sine

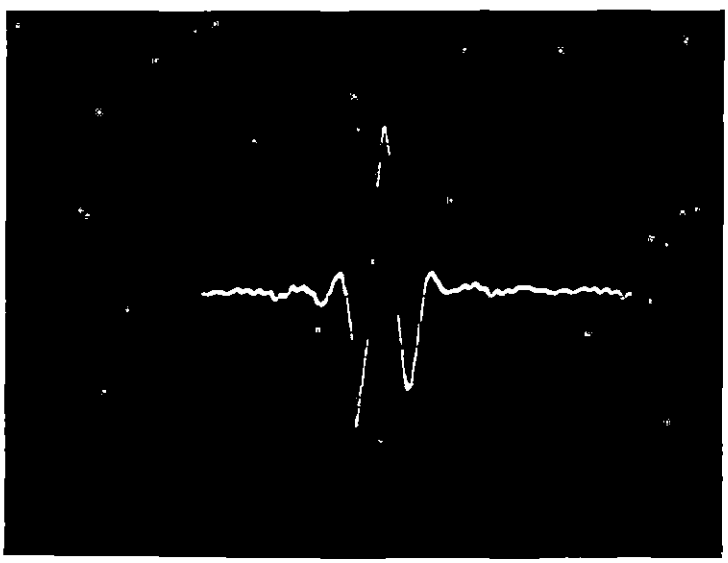


Figure 18. IR when cancelling white noise

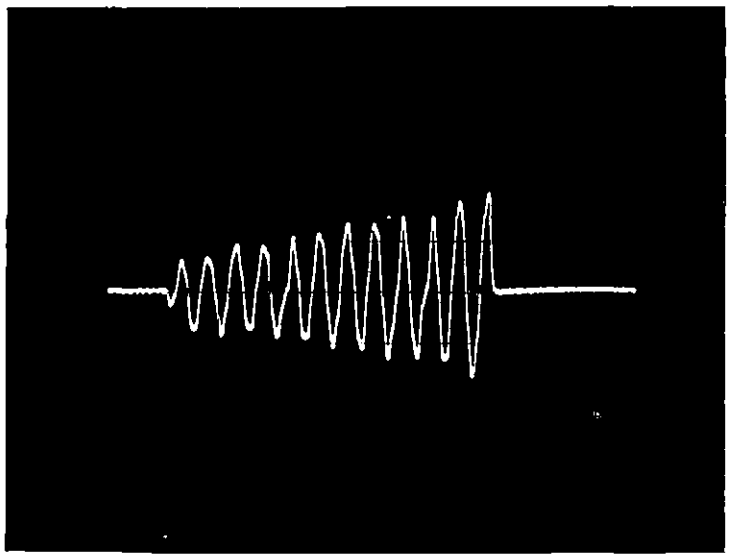


Figure 19. IR when cancelling sine with H(z) unequal to unity

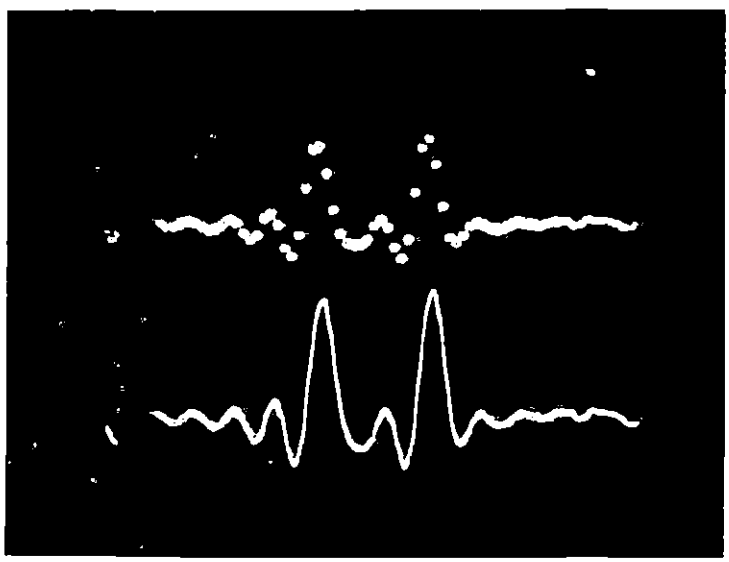


Figure 20. IR when cancelling white noise with H(z) unequal to unity (The upper trace shows the IR before low pass filtering.)

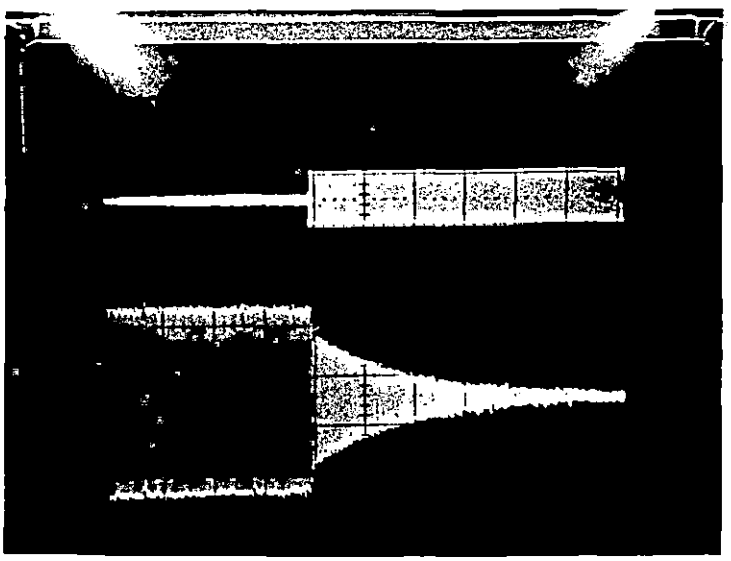


Figure 21. Transient behavior (The upper trace is the reference; the lower trace is the canceler output.)

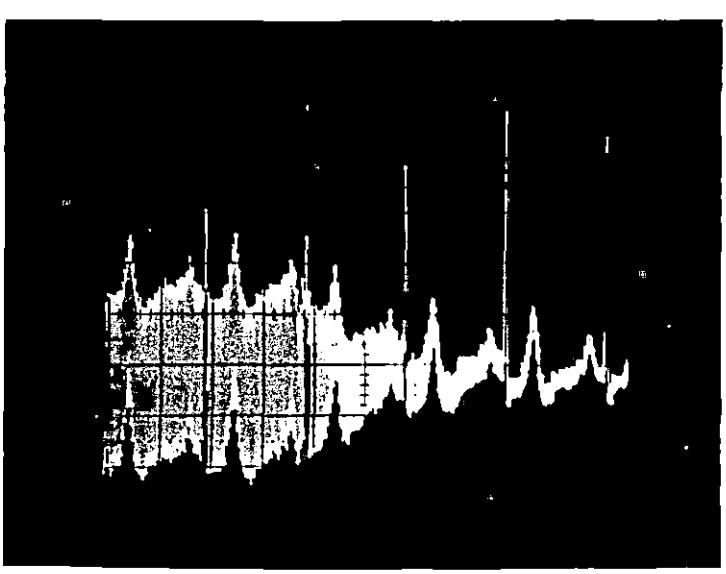


Figure 22. Transient behavior (The filter converges to cancel the 60 Hz sine from the ECG.)

of frequency f with a change in amplitude. With a good deal more effort, it can be shown that the function in Figure 19 will do the same, along with creating some sines of other frequencys roughly 37 dB down from the sine at frequency f. Thus, either impulse response could do the same filtering job.

Figures 21 and 22 show typical transient behavior of the filter. The upper trace in 21 is the reference signal; the lower is the filter ouput. Convergence time is dependent on step size and reference spectrum. For a step size of 2**-5, convergence times ranged from 2 to 10s for simple sines to white noise. Times of less than a second are possible with a step size of 1/2, but stability may then be in doubt.

Table 1 gives data on SNR improvement. The nominal noise reduction of 20 dB is mediocre. The inverse relationship of useful bandwidth to filter length is indicated in Table 2. The 16 tap filter worked well on single sines up to a frequency of 120 Hz. This was the filter used to cancel a 60 Hz sine from an ECG. A longer filter, 64 taps, was needed to cancel white noise.

Stability

As a good test of stability a step size of 2**-4 was chosen to filter white noise. The filter was turned on and let run for 1 hour. In that time, the filter transfer function was updated 36.4 million times. Occasional perturbations occurred, but the filter would reconverge within 10 s. It is considered stable.

Table 2. Bandwidth

Filter length, N	Sampling Rate (Hz)	Bandwidth (Hz)
16	500	120
32	300	50
64	140	30

DISCUSSION

The results of this research are good. A general purpose microcomputer was outfitted with moderately sophisticated analog input channels and an analog output channel. A digital filter program was written and optimized.

The filter achieves a conservative figure of 20 dB improvement in signal to noise ratio for the sinusoidal and wideband signals tested. 20 dB is certainly not the theoretical limit. For the case in which m1 = m2 (Figure 13), and infinite precision, infinite rejection of noise is the limit. However, finite precision arithmetic is limited to finite noise rejection. The SNR improvement achieved is good considering the available equipment. Computation of the precise rejection limit obtainable requires correlation measurements (15). This was not done.

Noise rejection would be improved by taking steps to overcome the problems of finite word length. These steps should include, foremost, use of a microcomputer designed for signal processing applications. Such a system would allow for use of a more sophisticated algorithm like the adaptive gradient lattice and still give useful bandwidth.

Also, a faster system would allow sampling at far above the Nyquist rate. This would effectively interpolate between the points seen at a lower sampling rate.

An improvement would be the use of a 12 bit ADC.

Among all these hardware considerations one also needs to use the filter in a useful way. Good use of this kind of filter can be made wherever the input x (Figure 1) is available. This does not require that

the noise n be known exactly but only that its source be known. Examples of chronic, known biomedical noise sources are power line 60 Hz, electrocautery interference, and electromyograms masking other biopotentials.

CONCLUSION

The Adaptive Gradient Lattice and Least Mean Square filters were considered for use in real-time noise cancelling situations. The LMS was chosen for implementation due to its computational simplicity. Even though the algorithm was simple, bandwidth was still low for the case of white noise. The filter converged for the signals tested and remained stable. Noise reduction was limited to a conservative figure of 20 dB due to the algorithms sensitivity to errors caused by limited word length arithmetic. A working digital adaptive filter has been demonstrated.

REFERENCES

(1) R. Brown. Introduction to Random Signal Analysis and Kalman Filtering. New York: Wiley, 1983.

(2) T. Kailath. "A View of Three Decades of Linear Filtering Theory." IEEE Trans. Information Theory IT-20 (March 1974):146-180.

(3) M. Yelderman, B. Widrow, J. Cioffi, E. Hesler, and J. Leddy. "ECG Enhancement by Adaptive Cancellation of Electrosurgical Interference." IEEE Trans. Biomedical Engineering BME-30 (April 1984):30-38.

(4) B. Widrow, J. Glover, J. McCool, J. Kaunitz, C. Williams, R. Hearn, J. Zeidler, E. Dong and R. Goodlin. "Adaptive Noise Cancelling: Principles and Applications." Proceedings of the IEEE 63 (March 1975):1692-1716.

(5) J. Makhoul. "Stable and Efficient Lattice Methods for Linear Prediction." IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-25 (Oct. 1977):423-428.

(6) S. Kay and S. Marple. "Spectrum Analysis - A Modern Perspective." Proceedings of the IEEE 69 (Nov. 1981):1380-1419.

(7) J. Makhoul. "Linear Prediction: A Tutorial Review." Proceedings of the IEEE 70 (April 1975):561-580.

(8) C. Gritton and D. Lin. "Echo Cancellation Algorithms." IEEE Trans. Acoustics, Speech and Signal Processing ASSP-23 (April 1984):30-38.

(9) B. Friedlander. "Lattice Filters for Adaptive Processing." Proceedings of the IEEE 70 (Aug. 1982):829-867.

(10) C. Mullis and R. Roberts. "Synthesis of Minimum Roundoff Noise Fixed Point Digital Filters." IEEE Trans. Circuits and Systems CAS-23 (Sept. 1976):551-562.

(11) P. Vaidyanathan, and S. Mitra. "Low Passband Sensitivity Digital Filters: A Generalized Viewpoint and Synthesis Procedures." Proceedings of the IEEE 72 (April 1984):404-423.

(12) J. Cioffi and T. Kailath. "Fast, Recursive-Least-Squares Transversal Filters for Adaptive Filtering." IEEE Trans. Acoustics, Speech and Signal Processing ASSP-32 (April 1984):304-337.

(13) A. Whalen. Detection of Signals in Noise. New York: Wiley, 1971.

(14) J. Milman. Microelectronics. New York: McGraw Hill, 1979.

(15) C. Caraiscos and B. Liu. "A Roundoff Error Analysis of the LMS Adaptive Algorithm." IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-32 (Feb. 1984):34-41.