THE UNIVERSITY OF QUEENSLAND

ACOUSTIC ECHO CANCELLATION USING DIGITAL SIGNAL PROCESSING

By

Michael Hutson

The School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of Bachelor of Engineering (Honours) in the division of Electrical and Electronic Engineering.

November 2003.

28th October 2003

Professor Simon Kaplan
School of Information Technology and Electrical Engineering,
The University of Queensland,
St. Lucia, QLD
4072

Dear Professor Kaplan,

In accordance with the requirements of the Degree of Bachelor of Engineering (Honours) in the division of Electrical and Electronic Engineering, I present the Following thesis entitled, "Acoustic Echo Cancellation using Digital Signal Processing". This work was performed under the supervision of Dr Vaughan Clarkson.

I declare that the work submitted is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at the University of Queensland or any other institution.

Yours Sincerely,

MICHAEL HUTSON

ACKNOWLEDGEMENTS.

Thanks go to Dr. Vaughan Clarkson, my thesis supervisor. Without him this work would never have been completed. His advice, assistance and patience are greatly appreciated.

Also, thank you to my family and friends for tolerating my regular madness during this project.

Hats off to you all.

TABLE OF CONTENTS.

Thesis Abstract

Acoustic echo cancellation is a common occurrence in todays telecommunication systems. It occurs when an audio source and sink operate in full duplex mode, an example of this is a hands-free loudspeaker telephone. In this situation the received signal is output through the telephone loudspeaker (audio source), this audio signal is then reverberated through the physical environment and picked up by the systems microphone (audio sink). The effect is the return to the distant user of time delayed and attenuated images of their original speech signal.

The signal interference caused by acoustic echo is distracting to both users and causes a reduction in the quality of the communication. This thesis focuses on the use of adaptive filtering techniques to reduce this unwanted echo, thus increasing communication quality.

Adaptive filters are a class of filters that iteratively alter their parameters in order to minimise a function of the difference between a desired target output and their output. In the case of acoustic echo in telecommunications, the optimal output is an echoed signal that accurately emulates the unwanted echo signal. This is then used to negate the echo in the return signal. The better the adaptive filter emulates this echo, the more successful the cancellation will be.

This thesis examines various techniques and algorithms of adaptive filtering, employing discrete signal processing in MATLAB. Also a real-time implementation of an adaptive echo cancellation system has been developed using the Texas Instruments TMS320C6711 DSP development kit.

1. Introduction.

The topic for this thesis is “Acoustic Echo Cancellation using Digital Signal Processing”. Acoustic echo occurs when an audio signal is reverberated in a real environment, resulting in the original intended signal plus attenuated, time delayed images of this signal.

This thesis will focus on the occurrence of acoustic echo in telecommunication systems. Such a system consists of coupled acoustic input and output devices, both of which are active concurrently. An example of this is a hands-free telephony system. In this scenario the system has both an active loudspeaker and microphone input operating simultaneously. The system then acts as both a receiver and transmitter in full duplex mode. When a signal is received by the system, it is output through the loudspeaker into an acoustic environment. This signal is reverberated within the environment and returned to the system via the microphone input. These reverberated signals contain time delayed images of the original signal, which are then returned to the original sender (Figure 1.1, a_k is the attenuation, t_k is time delay). The occurrence of acoustic echo in speech transmission causes signal interference and reduced quality of communication.

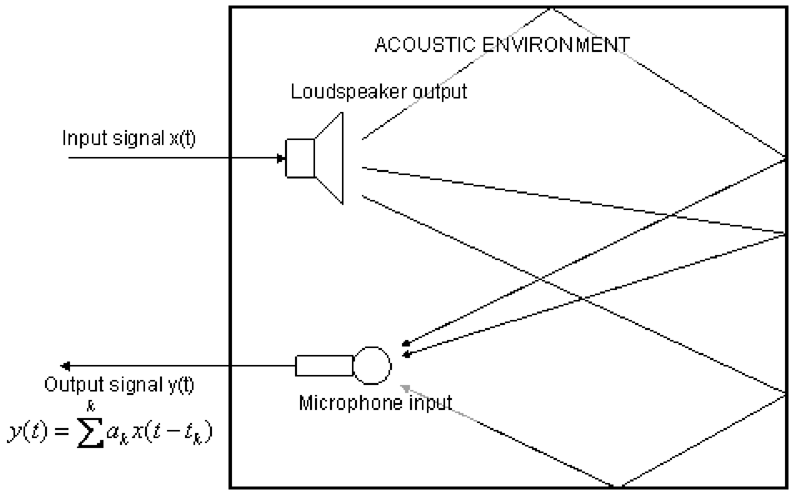


Figure 1.1: Origins of acoustic echo.

The method used to cancel the echo signal is known as adaptive filtering. Adaptive filters are dynamic filters which iteratively alter their characteristics in order to achieve an optimal desired output. An adaptive filter algorithmically alters its parameters in order to minimise a function of the difference between the desired output d(n) and its actual output y(n). This function is known as the cost function of the adaptive algorithm. Figure 1.2 shows a block diagram of the adaptive echo cancellation system implemented throughout this thesis. Here the filter H(n) represents the impulse response of the acoustic environment, W(n) represents the adaptive filter used to cancel the echo signal. The adaptive filter aims to equate its output y(n) to the desired output d(n) (the signal reverberated within the acoustic environment). At each iteration the error signal, e(n)=d(n)-y(n), is fed back into the filter, where the filter characteristics are altered accordingly.

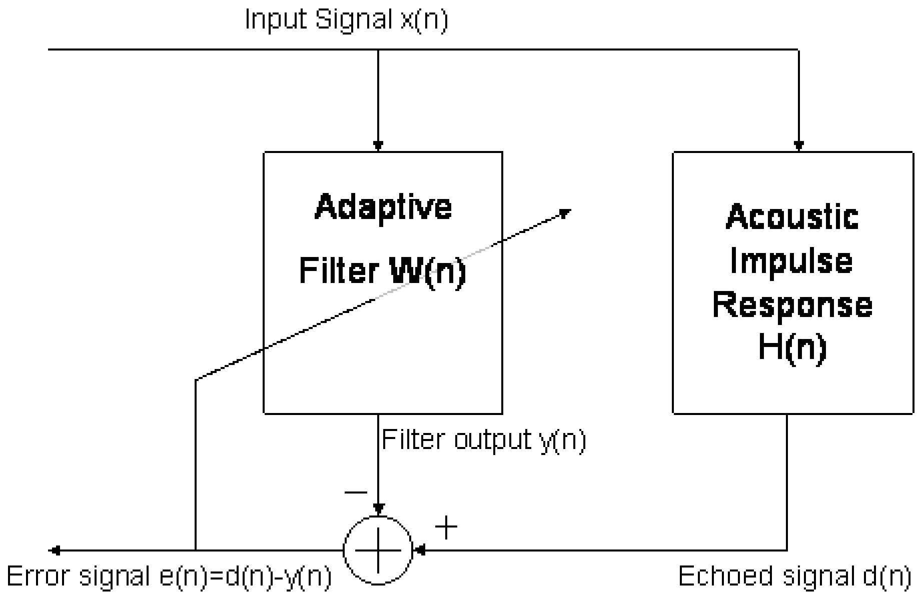


Figure 1.2: Block diagram of an adaptive echo cancellation system.

This thesis deals with acoustic echo as applies to telecommunication systems, although the techniques will be applicable to a variety of other disciplines.

This goals of this thesis are as follows:

- To examine adaptive filtering techniques as they apply to acoustic echo cancellation.
- To simulate these adaptive filtering algorithms using Matlab.
- To develop a real time acoustic echo cancellation system using the Texas Instruments TMS320C6711 digital signal processing development kit.

This thesis is divided into a number of sections:

- Section 2 deals with background signal processing theory as it pertains to the application of adaptive filters.
- Section 3 studies the basis of adaptive filtering techniques as well as the development and derivation of algorithms used within this thesis.
- Section 4 details the simulations of adaptive filtering techniques as developed in Matlab. This section shows the results of these simulations as well as discussing the advantages and disadvantages of each technique.
- Section 5 outlines the real time implementation of an adaptive echo cancellation system. This system uses the Texas Instruments TMS320C6711 DSP development kit, the details of which are also examined in this section.
- Section 6 summarises the work and provides recommendations for future research and development.
- Appendices A and B list the source code used in the Matlab and real time implementations respectively. These files are also included on the CD accompanying this thesis.

2. BACKGROUND THEORY

In order to understand the content presented in this thesis it is first necessary to provide some background information regarding digital signal theory. It will start out rather elementary and then progress to more complicated matters. Later chapters will build upon this theoretical basis in the derivation and implementation of the adaptive filtering techniques used in acoustic echo cancellation.

2.1 DISCRETE TIME SIGNALS.

Real world signals, such as speech are analog and continuous. An audio signal, as heard by our ears is a continuous waveform which derives from air pressure variations fluctuating at frequencies which we interpret as sound. However, in modern day communication systems these signals are represented electronically by discrete numeric sequences. In these sequences, each value represents an instantaneous value of the continuous signal. These values are taken at regular time periods, known as the sampling period, T_s.

For example, consider a continuous waveform given by x(t). In order to process this waveform digitally we first must convert this into a discrete time vector. Each value in the vector represents the instantaneous value of this waveform at integer multiples of the sampling period. The values of the sequence, x(t) corresponding to the value at n times the sampling period is denoted as x(n).

$$x(n) = x(nT_s) \quad \text{(eq 2.1)}$$

2.2 TRANSVERSAL FIR FILTERS

A filter can be defined as a piece of software or hardware that takes an input signal and processes it so as to extract and output certain desired elements of that signal (Diniz 1997, p.1). There are numerous filtering methods, both analog and digital which are widely used. However, this thesis shall be contained to adaptive filtering using a particular method known as transversal finite impulse response (FIR) filters.

The characteristics of a transversal FIR filter can be expressed as a vector consisting of values known as tap weights. It is these tap weights which determine the performance of the filter. These values are expressed in column vector form as, $\mathbf{w}(n) = [w_0(n)\ w_1(n)\ w_2(n) \ldots\ w_{N-1}(n)]^T$. This vector represents the impulse response of the FIR filter. The number of elements on this impulse response vector corresponds to the order of the filter, denoted in this thesis by the character N.

The utilisation of an FIR filter is simple, the output of the FIR filter at time n is determined by the sum of the products between the tap weight vector, $\mathbf{w}(n)$ and N time delayed input values. If these time delayed inputs are expressed in vector form by the column vector $\mathbf{x}(n) = [x(n)\ x(n-1)\ x(n-2) \ldots\ x(n-N+1)]^T$, the output of the filter at time n is expressed by equation 2.2. Throughout this thesis the vector containing the time delayed input values at time n is referred to as the input vector, $\mathbf{x}(n)$. In adaptive filtering the tap weight values are time varying so for each at each time interval a new FIR tap weight vector must be calculated, this is denoted as the column vector $\mathbf{w}(n) = [w_0(n)\ w_1(n)\ w_2(n) \ldots\ w_{N-1}(n)]^T$.

$$y(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i) \quad \text{(eq 2.2)}$$

It can be seen that this is also equivalent to the dot product between the impulse response vector and the input vector, and alternatively the product of the transpose of the filter tap weight vector and the input vector, see equation 2.3. Both of these equalities will be used throughout the thesis.

$$y(n) = \mathbf{w}(n) \cdot \mathbf{x}(n)$$
$$y(n) = \mathbf{w}^T(n)\mathbf{x}(n) \quad \text{(eq 2.3)}$$

Figure 2.1 shows a block schematic of a real transversal FIR filter, here the input values are denoted by u(n), the filter order is denoted by M, and z^{-1} denotes a delay of one sample period.

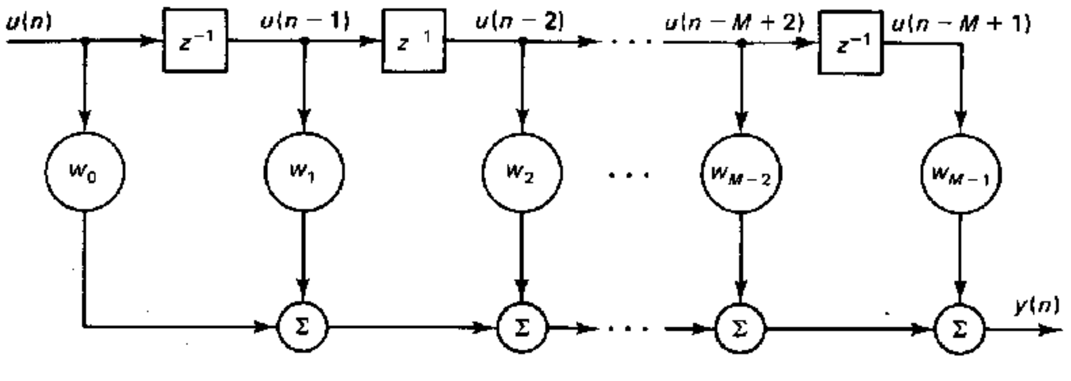


Figure 2.1: Transversal FIR filter (Haykin 1991, p. 5)

Adaptive filters utilise algorithms to iteratively alter the values of the impulse response vector in order to minimise a value known as the cost function. The cost function, ξ(n), is a function of the difference between a desired output and the actual output of the FIR filter. This difference is known as the estimation error of the adaptive filter, e(n) = d(n)-y(n) . Section 3 will examine adaptive filter theory further and derive several algorithms used.

2.3 RANDOM SIGNALS

Like many signals in real world applications, the values of the input vector of the acoustic echo cancellation system are unknown before they arrive. Also as it is difficult to predict these values, they appear to behave randomly. So a brief examination of random signal theory will be treated in this section.

A random signal, expressed by random variable function, x(t), does not have a precise description of its waveform. It may, however, be possible to express these random processes by statistical or probabilistic models (Diniz 1997, p.17). A single occurrence of a random variable appears to behave unpredictably. But if we take several occurrences of the variable, each denoted by n, then the random signal is expressed by two variables, x(t,n).

The main characteristic of a random signal treated in this thesis is known as the expectation of a random signal. It is defined as the mean value across all n occurrences of that random variable, denoted by E[x(t)], where x(t) is the input random variable. It should be noted that the number of input occurrences into the acoustic echo cancellation system is always 1. Throughout this thesis the expectation of an input signal is equal to the actual value of that signal. However, the E[x(n)] notation shall still be used in order to derive the various algorithms used in adaptive filtering.

2.4 CORRELATION FUNCTION

The correlation function is a measure of how statistically similar two functions are. The autocorrelation function of a random signal is defined as the expectation of a signals value at time n multiplied by its complex conjugate value at a different time m. This is shown in equation 2.4, for time arbitrary time instants, n and m.

$$\phi_{xx}(n,m) = E\left[x(n)x^{*}(m)\right] \text{ (eq 2.4)}$$

As this thesis deals only with real signals the above equation becomes.

$$\phi_{xx}(n,m) = E\left[x(n)x(m)\right] \text{ (eq 2.5)}$$

The derivations of adaptive filtering algorithms utilise the autocorrelation matrix, **R**. For real signals this is defined as the matrix of expectations of the product of a vector **x**(n) and its transpose. This is shown in equation 2.6 (Diniz 1997, p27).

$$\mathbf{R} = \begin{bmatrix} E\left[\left|x_0(k)^2\right|\right] & E\left[x_0(k)x_1(k)\right] & \dots & E\left[x_0(k)x_N(k)\right] \\ E\left[x_1(k)x_0(k)\right] & E\left[\left|x_1(k)^2\right|\right] & \dots & E\left[x_1(k)x_N(k)\right] \\ \dots & \dots & \dots & \dots \\ E\left[x_N(k)x_0(k)\right] & E\left[x_N(k)x_1(k)\right] & \dots & E\left[\left|x_N(k)^2\right|\right] \end{bmatrix}$$

$$\mathbf{R} = E\left[\mathbf{x}(k)\mathbf{x}^{T}(k)\right] \text{ (eq 2.6)}$$

The autocorrelation matrix has the additional property that its trace, i.e. the sum of its diagonal elements, is equal to the sum of the powers of the values in the input vector (Farhang-Boroujeny 1999, p. 97).

As we will see later, sometimes a single value replaces one of the vectors in the autocorrelation matrix, in this case the correlation function results in a vector. This vector is given by the expectation of that single value multiplied by the expectation of each of the values in the vector.

Correlation matrices and vectors are based on either cross-correlation or autocorrelation functions. This simply refers to the signals being used in the function. If it is cross correlation, the signals are different, if it is autocorrelation, the two signals used in the function are the same.

2.5 STATIONARY SIGNALS

A signal can be considered stationary in the wide sense, if the two following criteria are met (Farhang-Boroujeny 1999, pp 37-8).

1. The mean values, or expectations, of the signal are constant for any shift in time.

$$m_x(n) = m_x(n+k) \quad \text{(eq 2.7)}$$

2. The autocorrelation function is also constant over an arbitrary time shift.

$$\phi_{xx}(n,m) = \phi_{xx}(n+k,m+k) \quad \text{(eq 2.8)}$$

The above implies that the statistical properties of a stationary signal are constant over time. In the derivation of adaptive filtering algorithms it is often assumed that the signals input to the algorithm are stationary. Speech signals are not stationary in the wide sense, however the do exhibit some temporary stationary behaviour as will be seen in the next section.

2.6 SPEECH SIGNALS

A speech signal consists of three classes of sounds. They are voiced, fricative and plosive sounds. Voiced sounds are caused by excitation of the vocal tract with quasi-periodic pulses of airflow. Fricative sounds are formed by constricting the vocal tract and passing air through it, causing turbulence that results in a noise-like sound. Plosive sounds are created by closing up the vocal tract, building up air behind it then suddenly releasing it, this is heard in the sound made by the letter 'p' (Oppenheim & Schafer 1989, p. 724).

Figure 2.2 shows a discrete time representation of a speech signal. By looking at it as a whole we can tell that it is non-stationary. That is, its mean values vary with time and cannot be predicted using the above mathematical models for random processes. However, a speech signal can be considered as a linear composite of the above three classes of sound, each of these sounds are stationary and remain fairly constant over intervals of the order of 30 to 40 ms (Oppenheim & Schafer 1989, p. 724).

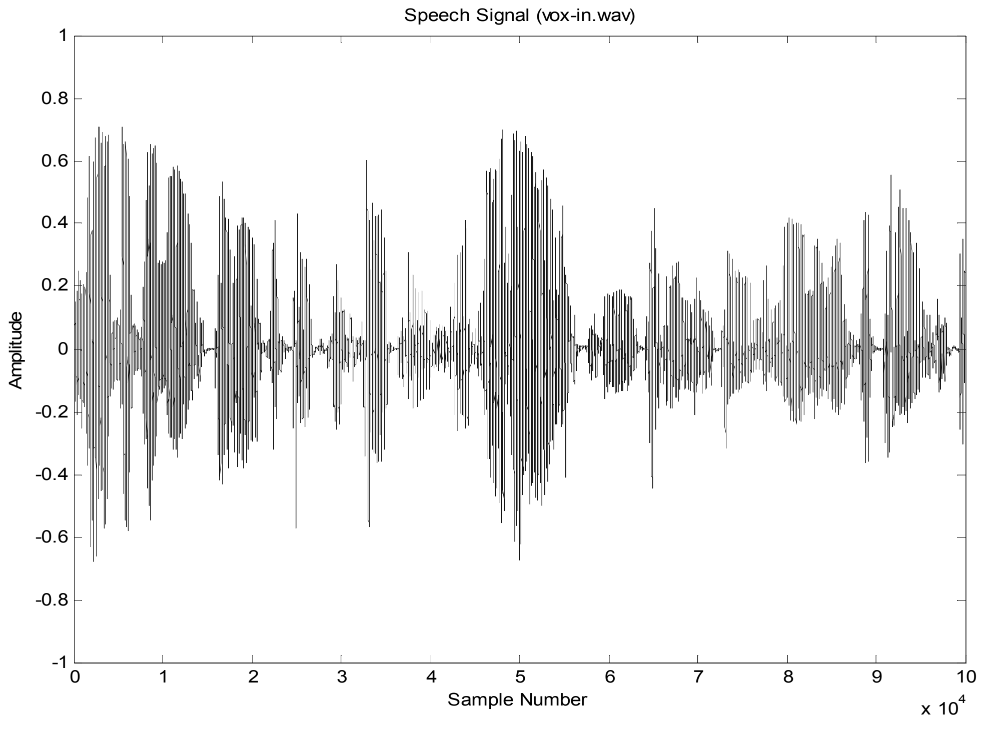


Figure 2.2. Speech signal representation.

The theory behind the derivations of many adaptive filtering algorithms usually requires the input signal to be stationary. Although speech is non-stationary for all time, it is an assumption of this thesis that the short term stationary behaviour outlined above will prove adequate for the adaptive filters to function as desired.

2.7 MATRIX INVERSION LEMMA.

The matrix inversion lemma is an identity of matrix algebra, it is a simple way to determine the inverse of a matrix. The matrix inversion lemma is as follows (Haykin 1991, p. 480). Let **A** and **B** be two positive-definite MxM matrices, **C** be an MxN matrix and **D** is a positive-definite NxN matrix, (superscript H denotes hermitian transposition, which is transposition followed by complex conjugation).

$$\text{If } \mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^{H}$$

$$\text{then } \mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^{H}\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^{H}\mathbf{B} \quad \text{(eq 2.9)}$$

A special form of the matrix inversion lemma is used in the derivation of the recursive least squares (RLS) adaptive filtering algorithm in section 3.4. This special form is stated in equation 2.10, for an arbitrary non-singular NxN matrix **A**, any Nx1 vector **a,** and a scalar α (Farhang-Boroujeny 1999, p.421).

$$(\mathbf{A} + \alpha\mathbf{a}\mathbf{a}^{T})^{-1} = \mathbf{A}^{-1} - \frac{\alpha\mathbf{A}^{-1}\mathbf{a}\mathbf{a}^{T}\mathbf{A}^{-1}}{1 + \alpha\mathbf{a}^{T}\mathbf{A}^{-1}\mathbf{a}} \quad \text{(eq 2.10)}$$

3. ADAPTIVE FILTERS.

3.1 INTRODUCTION

Figure 3.1 shows the block diagram for the adaptive filter method utilised in this thesis.

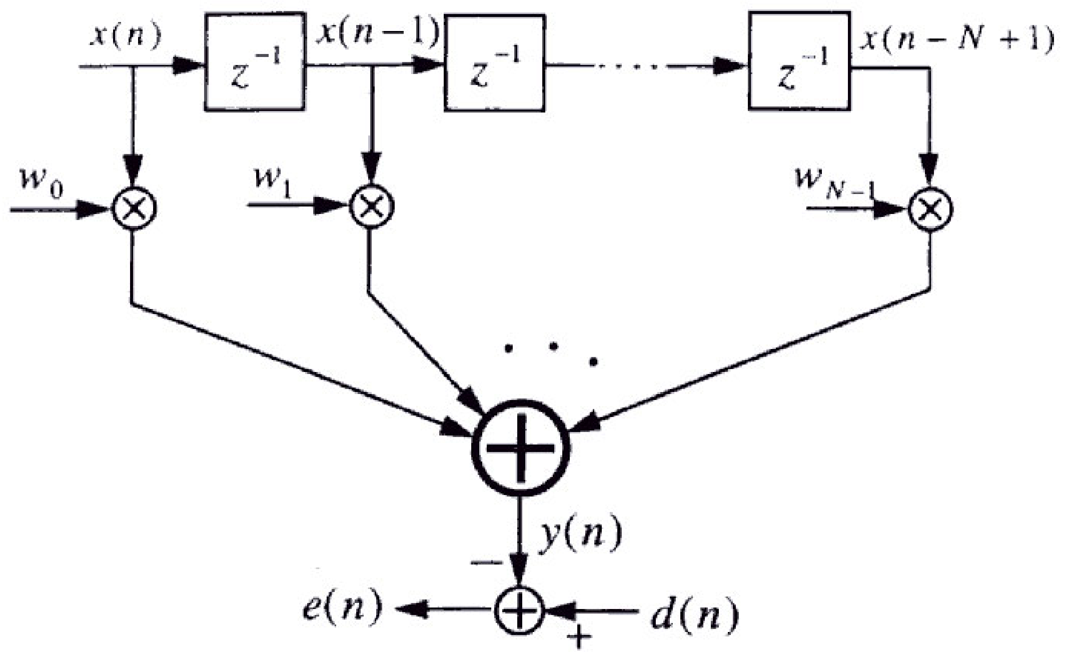


Figure 3.1. Adaptive filter block diagram (Farhang-Boroujeny 1999, p. 120)

Here w represents the coefficients of the FIR filter tap weight vector, x(n) is the input vector samples, z^{-1} is a delay of one sample perios, y(n) is the adaptive filter output, d(n) is the desired echoed signal and e(n) is the estimation error at time n.

The aim of an adaptive filter is to calculate the difference between the desired signal and the adaptive filter output, e(n). This error signal is fed back into the adaptive filter and its coefficients are changed algorithmically in order to minimise a function of this difference, known as the cost function. In the case of acoustic echo cancellation, the optimal output of the adaptive filter is equal in value to the unwanted echoed signal. When the adaptive filter output is equal to desired signal the error signal goes to zero. In this situation the echoed signal would be completely cancelled and the far user would not hear any of their original speech returned to them.

This section examines adaptive filters and various algorithms utilised. The various methods used in this thesis can be divided into two groups based on their cost functions. The first class are known as Mean Square Error (MSE) adaptive filters, they aim to minimise a cost function equal to the expectation of the square of the difference between the desired signal d(n), and the actual output of the adaptive filter y(n) (equation 3.1).

$$\xi(n) = E\left[e^2(n)\right] = E\left[(d(n) - y(n))^2\right] \quad \text{(eq 3.1)}$$

The second class are known as Recursive Least Squares (RLS) adaptive filters and they aim to minimise a cost function equal to the weighted sum of the squares of the difference between the desired and the actual output of the adaptive filter for different time instances. The cost function is recursive in the sense that unlike the MSE cost function, weighted previous values of the estimation error are also considered. The cost function is shown below in equation 3.2, the parameter λ is in the range of 0<λ<1. It is known as the forgetting factor as for λ<1 it causes the previous values to have an increasingly negligible effect on updating of the filter tap weights (Diniz 1997, p. 184). The value of 1/(1- λ) is a measure of the memory of the algorithm (Farhang-Boroujeny 1999, p. 420), this thesis will primarily deal with infinite memory ,i.e. λ =1. The cost function for RLS algorithm, ζ(n), is stated in equation..

$$\zeta(n) = \sum_{k=1}^{n} \rho_n(k) e_n^2(k)$$
$$\rho_n(k) = \lambda^{n-k} \quad \text{(eq 3.2)}$$

Where k=1, 2, 3…n., k=1 corresponds to the time at which the RLS algorithm commences. Later we will see that in practice not all previous values are considered, rather only the previous N (corresponding to the filter order) error signals are considered.

As stated previously, considering that the number of processes in our ensemble averages is equal to one, the expectation of an input or output value is equal to that actual value at a unique time instance. However, for the purposes of deriving these algorithms, the expectation notation shall still be used.

3.2 WIENER FILTERS

Wiener filters are a special class of transversal FIR filters which build upon the mean square error cost function of equation 3.1 to arrive at an optimal filter tap weight vector which reduces the MSE signal to a minimum. They will be used in the derivation of adaptive filtering algorithms in later sections, this theory is based on Diniz 1997, pp. 38 to 42, and Farhang-Boroujeny 1999, pp. 51-54.

Consider the output of the transversal FIR filter as given below, for a filter tap weight vector, **w**(n), and input vector, **x**(n).

$$y(n) = \sum_{i=0}^{N-1} w(n)x(n-i) = \mathbf{w}^T(n)\mathbf{x}(n) \qquad \text{(eq 3.3)}$$

The mean square error cost function can be expressed in terms of the cross-correlation vector between the desired and input signals, **p**(n)=E[**x**(n) d(n)], and the autocorrelation matrix of the input signal, **R**(n)=E[**x**(n)**x**^{T}(n)].

$$\begin{aligned}\xi(n) &= E\left[e^2(n)\right] \\ &= E\left[(d(n)-y(n))^2\right] \\ &= E\left[d^2(n) - 2d(n)\mathbf{w}^T(n)\mathbf{x}(n) + \mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)\right] \\ &= E\left[d^2(n)\right] - 2E\left[\mathbf{w}^T(n)\mathbf{x}(n)\right] + E\left[\mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)\right] \\ &= E\left[d^2(n)\right] - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w} \end{aligned} \qquad \text{(eq 3.4)}$$

When applied to FIR filtering the above cost function is an N-dimensional quadratic function. The minimum value of ξ(n) can be found by calculating its gradient vector related to the filter tap weights and equating it to 0.

$$\begin{aligned} \frac{\partial}{\partial w_i} &= 0 \text{ for } i = 0,1,\ldots,\text{N - 1} \\ \nabla &= \left[\frac{\partial}{\partial w_o}\frac{\partial}{\partial w_1}\cdots\frac{\partial}{\partial w_{N-1}}\right]^T \\ \nabla\xi &= \mathbf{0} \end{aligned} \qquad \text{(eq 3.5)}$$

By finding the gradient of equation 3.4, equating it to zero and rearranging gives us the optimal wiener solution for the filter tap weights, $\mathbf{w}_o$.

$$\nabla\xi = \mathbf{0}$$
$$2\mathbf{R}\mathbf{w}_0 - 2\mathbf{p} = \mathbf{0}$$
$$\mathbf{w}_0 = \mathbf{R}^{-1}\mathbf{p} \quad \text{(eq 3.6)}$$

The optimal wiener solution is the set of filter tap weights which reduce the cost function to zero. This vector can be found as the product of the inverse of the input vector autocorrelation matrix and the cross correlation vector between the desired signal and the input vector. The Least Mean Square algorithm of adaptive filtering attempts to find the optimal wiener solution using estimations based on instantaneous values.

3.3 MEAN SQUARE ERROR (MSE) ADAPTIVE FILTERS

3.3.1 LEAST MEAN SQUARE (LMS) ALGORITHM

The Least Mean Square (LMS) algorithm was first developed by Widrow and Hoff in 1959 through their studies of pattern recognition (Haykin 1991, p. 67). From there it has become one of the most widely used algorithms in adaptive filtering. The LMS algorithm is a type of adaptive filter known as stochastic gradient-based algorithms as it utilises the gradient vector of the filter tap weights to converge on the optimal wiener solution. It is well known and widely used due to its computational simplicity. It is this simplicity that has made it the benchmark against which all other adaptive filtering algorithms are judged (Haykin 1991, p. 299).

With each iteration of the LMS algorithm, the filter tap weights of the adaptive filter are updated according to the following formula (Farhang-Boroujeny 1999, p. 141).

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n) \quad \text{(eq 3.7)}$$

Here $\mathbf{x}(n)$ is the input vector of time delayed input values, $\mathbf{x}(n) = [x(n)\ x(n-1)\ x(n-2)\ \ldots\ x(n-N+1)]^T$. The vector $\mathbf{w}(n) = [w_0(n)\ w_1(n)\ w_2(n)\ \ldots\ w_{N-1}(n)]^T$ represents the coefficients of the adaptive FIR filter tap weight vector at time n. The parameter μ is known as the step size parameter and is a small positive constant. This step size parameter controls the influence of the updating factor. Selection of a suitable value for μ is imperative to the performance of the LMS algorithm, if the value is too small the time the adaptive filter takes to converge on the optimal solution will be too long; if μ is too large the adaptive filter becomes unstable and its output diverges.

Derivation of the LMS algorithm.

The derivation of the LMS algorithm builds upon the theory of the wiener solution for the optimal filter tap weights, $\mathbf{w}_o$, as outlined in section 3.2. It also depends on the steepest-descent algorithm as stated in equation 3.8, this is a formula which updates the filter coefficients using the current tap weight vector and the current gradient of the cost function with respect to the filter tap weight coefficient vector, $\nabla\xi(n)$.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\nabla\xi(n)$$
$$\text{where } \xi(n) = E\left[e^2(n)\right] \qquad \text{(eq 3.8)}$$

As the negative gradient vector points in the direction of steepest descent for the N-dimensional quadratic cost function, each recursion shifts the value of the filter coefficients closer toward their optimum value, which corresponds to the minimum achievable value of the cost function, $\xi(n)$. This derivation is based on Diniz 1997, pp. 71-3 and Farhang-Boroujeny 1999, pp.139–41.

The LMS algorithm is a random process implementation of the steepest descent algorithm, from equation 3.8. Here the expectation for the error signal is not known so the instantaneous value is used as an estimate. The steepest descent algorithm then becomes equation 3.9.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\nabla\xi(n)$$
$$\text{where } \xi(n) = e^2(n) \qquad \text{(eq 3.9)}$$

The gradient of the cost function, $\nabla\xi(n)$, can alternatively be expressed in the following form.

$$\begin{aligned}\nabla\xi(n) &= \nabla(e^2(n)) \\ &= \frac{\partial e^2(n)}{\partial \mathbf{w}} \\ &= 2e(n)\frac{\partial e(n)}{\partial \mathbf{w}} \\ &= 2e(n)\frac{\partial(d(n)-y(n))}{\partial \mathbf{w}} \\ &= -2e(n)\frac{\partial \mathbf{w}^T(n)\mathbf{x}(n)}{\partial \mathbf{w}} \\ &= -2e(n)\mathbf{x}(n) \qquad \text{(eq 3.10)}\end{aligned}$$

Substituting this into the steepest descent algorithm of equation 3.8, we arrive at the recursion for the LMS adaptive algorithm.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n) \quad \text{(eq 3.11)}$$

Implementation of the LMS algorithm.

Each iteration of the LMS algorithm requires 3 distinct steps in this order:

1. The output of the FIR filter, y(n) is calculated using equation 3.12.

$$y(n) = \sum_{i=0}^{N-1} w(n)x(n-i) = \mathbf{w}^T(n)\mathbf{x}(n) \quad \text{(eq 3.12)}$$

2. The value of the error estimation is calculated using equation 3.13.

e(n)=d(n)-y(n) (eq 3.13)

3. The tap weights of the FIR vector are updated in preparation for the next iteration, by equation 3.14.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n) \quad \text{(eq 3.14)}$$

The main reason for the LMS algorithms popularity in adaptive filtering is its computational simplicity, making it easier to implement than all other commonly used adaptive algorithms. For each iteration the LMS algorithm requires 2N additions and 2N+1 multiplications (N for calculating the output, y(n), one for 2μe(n) and an additional N for the scalar by vector multiplication) (Farhang-Boroujeny 1999, p. 141).

3.3.2 NORMALISED LEAST MEAN SQUARE (NLMS) ALGORITHM

One of the primary disadvantages of the LMS algorithm is having a fixed step size parameter for every iteration. This requires an understanding of the statistics of the input signal prior to commencing the adaptive filtering operation. In practice this is rarely achievable. Even if we assume the only signal to be input to the adaptive echo cancellation system is speech, there are still many factors such as signal input power and amplitude which will affect its performance.

The normalised least mean square algorithm (NLMS) is an extension of the LMS algorithm which bypasses this issue by selecting a different step size value, μ(n), for each iteration of the algorithm. This step size is proportional to the inverse of the total expected energy of the instantaneous values of the coefficients of the input vector **x**(n) (Farhang-Boroujeny 1999, p.172). This sum of the expected energies of the input samples is also equivalent to the dot product of the input vector with itself, and the trace of input vectors auto-correlation matrix, **R** (Farhang-Boroujeny 1999, p.173).

$$\begin{aligned} tr[\mathbf{R}] &= \sum_{i=0}^{N-1} E\left[x^2(n-i)\right] \\ &= E\left[\sum_{i=0}^{N-1} x^2(n-i)\right] \end{aligned} \quad \text{(eq 3.15)}$$

The recursion formula for the NLMS algorithm is stated in equation 3.16.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{1}{\mathbf{x}^T(n)\mathbf{x}(n)} e(n)\mathbf{x}(n) \quad \text{(eq 3.16)}$$

Derivation of the NLMS algorithm.

This derivation of the normalised least mean square algorithm is based on Farhang-Boroujeny 1999, pp.172-175, and Diniz 1997, pp 150-3. To derive the NLMS algorithm we consider the standard LMS recursion, for which we select a variable step size parameter, μ(n). This parameter is selected so that the error value , $e^+(n)$, will be minimised using the updated filter tap weights, w(n+1), and the current input vector, **x**(n).

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + 2\mu(n)e(n)\mathbf{x}(n) \\ e^+(n) &= d(n) - \mathbf{w}^T(n+1)\mathbf{x}(n) \\ &= (1 - 2\mu(n)\mathbf{x}^T(n)\mathbf{x}(n))e(n) \end{aligned} \quad \text{(eq 3.17)}$$

Next we minimise $(e^{+}(n))^{2}$, with respect to μ(n). Using this we can then find a value for μ(n) which forces $e^{+}(n)$ to zero.

$$\mu(n) = \frac{1}{2\mathbf{x}^{T}(n)\mathbf{x}(n)} \quad \text{(eq 3.18)}$$

This μ(n) is then substituted into the standard LMS recursion replacing μ, resulting in the following.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu(n)e(n)\mathbf{x}(n)$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{1}{\mathbf{x}^{T}(n)\mathbf{x}(n)}e(n)\mathbf{x}(n) \quad \text{(eq 3.19)}$$

Often the NLMS algorithm is expressed as equation 3.20, this is a slight modification of the standard NLMS algorithm detailed above. Here the value of ψ is a small positive constant in order to avoid division by zero when the values of the input vector are zero. This was not implemented in the real time implementation as in practice the input signal is never allowed to reach zero due to noise from the microphone and from the AD codec on the Texas Instruments DSK. The parameter μ is a constant step size value used to alter the convergence rate of the NLMS algorithm, it is within the range of $0<\mu<2$, usually being equal to 1. One is the value that has been used throughout the Matlab implementations and the real time application of sections 4 and 5.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)\mathbf{x}(n)$$
$$\text{where } \mu(n) = \frac{\mu}{\mathbf{x}^{T}\mathbf{x} + \psi} \quad \text{(eq 3.20)}$$

Implementation of the NLMS algorithm.

The NLMS algorithm has been implemented in Matlab and in a real time application using the Texas Instruments TMS320C6711 Development Kit. As the step size parameter is chosen based on the current input values, the NLMS algorithm shows far greater stability with unknown signals. This combined with good convergence speed and relative computational simplicity make the NLMS algorithm ideal for the real time adaptive echo cancellation system. The code for both the Matlab and TI Code composer studio applications can be found in appendices A and B respectively.

As the NLMS is an extension of the standard LMS algorithm, the NLMS algorithms practical implementation is very similar to that of the LMS algorithm. Each iteration of the NLMS algorithm requires these steps in the following order (Farhang-Boroujeny 1999, p. 175).

1. The output of the adaptive filter is calculated.

$$y(n) = \sum_{i=0}^{N-1} w(n)x(n-i) = \mathbf{w}^T(n)\mathbf{x}(n) \quad \text{(eq 3.21)}$$

2. An error signal is calculated as the difference between the desired signal and the filter output.

$$e(n) = d(n) - y(n) \quad \text{(eq 3.22)}$$

3. The step size value for the input vector is calculated.

$$\mu(n) = \frac{1}{\mathbf{x}^{\mathrm{T}}(n)\mathbf{x}(n)} \quad \text{(eq 3.23)}$$

4. The filter tap weights are updated in preparation for the next iteration.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)e(n)\mathbf{x}(n) \quad \text{(eq 3.24)}$$

Each iteration of the NLMS algorithm requires 3N+1 multiplications, this is only N more than the standard LMS algorithm and as we shall see in section 4.2, this is an acceptable increase considering the gains in stability and echo attenuation achieved.

3.3.3 VARIABLE STEP SIZE LMS ALGORITHM

Both the LMS and the NLMS algorithms have a fixed step size value for every tap weight in each iteration. In the Variable Step Size Least Mean Square (VSLMS) algorithm the step size for each iteration is expressed as a vector, $\boldsymbol{\mu}(n)$. Each element of the vector $\boldsymbol{\mu}(n)$ is a different step size value corresponding to an element of the filter tap weight vector, $\mathbf{w}(n)$.

Derivation of the VSLMS algorithm

The VSLMS algorithm is as follows.

$$w_i(n+1) = w_i + 2\mu_i g_i(n)$$
$$\mathbf{g}(n) = e(n)\mathbf{x}(n) \qquad \text{(eq 3.25)}$$

Where g is a vector comprised of the gradient terms, $g_i(n)=e(n)x(n-i)$, i=0…N-1, the length corresponds to the order of the adaptive filter. The values for $\mu(n)$ can be calculated in either of the methods expressed in equation 3.26. The choice is dependent on the application, if a digital signal processor is used then the second equation is preferred. However, if a custom chip is designed then the first equation is usually utilised (Farhang-Boroujeny, p.176). For both the Matlab and real time applications the first equation is being implemented. Here ρ is a small positive constant optionally used to control the effect of the gradient terms on the update procedure, in the later implementations this is set to 1.

$$\mu_i(n) = \mu_i(n-1) + \rho sign(g_i(n))sign(g_i(n-1))$$
$$\mu_i(n) = \mu_i(n-1) + \rho g_i(n)g_i(n-1) \qquad \text{(eq 3.26)}$$

In order to ensure the step size parameters do not become too large (resulting in instability), or too small (resulting in slow reaction to changes in the desired impulse response), the allowable values for each element in the step size are bounded by upper and lower values (Farhang-Boroujeny, p.176). As with the standard LMS algorithm this implies that a prior knowledge of the signal statistics is necessary to guarantee optimal performance of the adaptive filter. A method to overcome this requirement is proposed in the section 3.3.4, 'Variable Step Size Normalised LMS algorithm'.

Implementation of the VSLMS algorithm.

The VSLMS algorithm is executed by following these steps for each iteration (Farhang-Boroujeny, p.177). With ρ=1, each iteration of the VSLMS algorithm requires 4N+1 multiplication operations.

1. The output of the adaptive filter is calculated.

$$y(n) = \sum_{i=0}^{N-1} w(n)x(n-i) = \mathbf{w}^T(n)\mathbf{x}(n) \qquad \text{(eq 3.27)}$$

2. The error signal is calculated as the difference between the desired output and the filter output.

$$e(n) = d(n) - y(n) \quad \text{(eq 3.28)}$$

3. The gradient, step size and filter tap weight vectors are updated using the following equations in preparation for the next iteration.

$$\begin{aligned}
&\text{For i} = 0,1,\ldots,\text{N - 1} \\
&g_i(n) = e(n)x(n-i) \\
&\mathbf{g}(n) = e(n)\mathbf{x}(n) \\
&\mu_i(n) = \mu_i(n-1) + \rho g_i(n) g_i(n-1) \\
&\text{if } \mu_i(n) > \mu_{\max}, \mu_i(n) = \mu_{\max} \\
&\text{if } \mu_i(n) < \mu_{\min}, \mu_i(n) = \mu_{\min} \\
&w_i(n+1) = w_i(n) + 2\mu_i(n) g_i(n) \qquad \text{(eq 3.29)}
\end{aligned}$$

3.3.4 VARIABLE STEP SIZE NORMALISED LMS ALGORITHM.

The VSLMS algorithm still has the same drawback as the standard LMS algorithm in that to guarantee stability of the algorithm, a statistical knowledge of the input signal is required prior to the algorithms commencement. Also, recall the major benefit of the NLMS algorithm is that it is designed to avoid this requirement by calculating an appropriate step size based upon the instantaneous energy of the input signal vector.

It is a natural progression to incorporate this step size calculation into the variable step size algorithm, in order increase stability for the filter without prior knowledge of the input signal statistics. This is what I have tried to achieve in developing the Variable step size normalised least mean square (VSNLMS) algorithm.

In the VSNLMS algorithm the upper bound available to each element of the step size vector, μ(n), is calculated for each iteration. As with the NLMS algorithm the step size value is inversely proportional to the instantaneous input signal energy.

Implementation of the VSNLMS algorithm.

The VSNLMS algorithm is implemented in Matlab as outlined later in section 4. It is essentially an extension of the implementation of the VSLMS algorithm with the added calculation of a maximum step size parameter for each iteration.

1. The output of the adaptive filter is calculated.

$$y(n) = \sum_{i=0}^{N-1} w(n)x(n-i) = \mathbf{w}^T(n)\mathbf{x}(n)$$ (eq 3.30)

2. The error signal is calculated as the difference between the desired output and the filter output.

$$e(n) = d(n) - y(n)$$ (eq 3.31)

3. The gradient, step size and filter tap weight vectors are updated using the following equations in preparation for the next iteration.

$$\begin{aligned}
&\text{f or } i = 0,1,\ldots,N-1 \\
&g_i(n) = e(n)x(n-i) \\
&\mathbf{g}(n) = e(n)\mathbf{x}(n) \\
&\mu_i(n) = \mu_i(n-1) + \rho g_i(n)g_i(n-1) \\
&\mu_{\max}(n) = \frac{1}{2\mathbf{x}^T(n)\mathbf{x}(n)} \\
&\text{if } \mu_i(n) > \mu_{\max}(n), \mu_i(n) = \mu_{\max}(n) \\
&\text{if } \mu_i(n) < \mu_{\min}(n), \mu_i(n) = \mu_{\min}(n) \\
&w_i(n+1) = w_i(n) + 2\mu_i(n)g_i(n)
\end{aligned}$$ (eq 3.32)

ρ is an optional constant the same as is the VSLMS algorithm. With ρ =1, each iteration of the VSNLMS algorithm requires 5N+1 multiplication operations.

3.4 RECURSIVE LEAST SQUARES (RLS) ADAPTIVE FILTERS.

The other class of adaptive filtering techniques studied in this thesis is known as Recursive Least Squares (RLS) algorithms. These algorithms attempt to minimise the cost function in equation 3.33(FB p.413). Where k=1 is the time at which the RLS algorithm commences and λ is a small positive constant very close to, but smaller than 1. With values of λ<1 more importance is given to the most recent error estimates and thus the more recent input samples, this results in a scheme that places more emphasis on recent samples of observed data and tends to forget the past (Farhang-Boroujeny p.419).

$$\zeta(n) = \sum_{k=1}^{n} \lambda^{n-k} e_n^2(k) \qquad \text{(eq 3.33)}$$

Unlike the LMS algorithm and its derivatives, the RLS algorithm directly considers the values of previous error estimations. RLS algorithms are known for excellent performance when working in time varying environments…these advantages come with the cost of an increased computational complexity and some stability problems (Diniz p.183).

Derivation of the RLS algorithm.

The RLS cost function of equation 3.33 shows that at a time n, all previous values of the estimation error since the commencement of the RLS algorithm are required. Clearly as time progresses the amount of data required to process this algorithm increases. The fact that memory and computation capabilities are limited makes the RLS algorithm a practical impossibility in its purest form. However, the derivation still assumes that all data values are processed. In practice only a finite number of previous values are considered, this number corresponds to the order of the RLS FIR filter, N.

The following derivation of the RLS is summarised from Farhang-Boroujeny pages 419 to 425. First we define $y_n(k)$ as the output of the FIR filter, at n, using the current tap weight vector, and the input vector of a previous time k. The estimation error value $e_n(k)$ is the difference between the desired output value at time k, and the corresponding value of $y_n(k)$. These and other appropriate definitions are expressed in equation 3.34, for k=1, 2, 3,…, n.

$$\begin{aligned}
y_n(k) &= \mathbf{w}^T(n)\mathbf{x}(k) \\
e_n(k) &= d(k) - y_n(k) \\
\mathbf{d}(n) &= [d(1), d(2) \ldots d(n)]^T \\
\mathbf{y}(n) &= [y_n(1), y_n(2) \ldots y_n(n)]^T \\
\mathbf{e}(n) &= [e_n(1), e_n(2) \ldots e_n(n)]^T \\
\mathbf{e}(n) &= \mathbf{d}(n) - \mathbf{y}(n)
\end{aligned} \qquad \text{(eq 3.34)}$$

If we define the **X**(n) as the matrix consisting of the n previous input column vector up to the present time then **y**(n) can also be expressed as equation 3.35.

$$\mathbf{X}(n) = [\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)]$$
$$\mathbf{y}(n) = \mathbf{X}^T(n)\mathbf{w}(n) \qquad \text{(eq 3.35)}$$

The cost function of equation 3.33, can then be expressed in matrix vector form using Λ(n), a diagonal matrix consisting of the weighting factors.

$$\begin{aligned}\zeta(n) &= \sum_{k=1}^{n} \lambda^{n-k} e_n^2(k)\\ &= \mathbf{e}^T(n)\widetilde{\Lambda}(n)\mathbf{e}(n)\end{aligned}$$
$$\text{where } \widetilde{\Lambda}(n) = \begin{bmatrix} \lambda^{n-1} & 0 & 0 & \ldots & 0 \\ 0 & \lambda^{n-2} & 0 & \ldots & 0 \\ 0 & 0 & \lambda^{n-3} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix} \qquad \text{(eq 3.36)}$$

Substituting values from equations 3.34 and 3.35, the cost function can be expanded then reduced as in equation 3.37. (Temporarily dropping (n) notation for clarity).

$$\begin{aligned}\zeta(n) &= \mathbf{e}^T(n)\widetilde{\Lambda}(n)\mathbf{e}(n)\\ &= \mathbf{d}^T\widetilde{\Lambda}\mathbf{d} - \mathbf{d}^T\widetilde{\Lambda}\mathbf{y} - \mathbf{y}^T\widetilde{\Lambda}\mathbf{d} + \mathbf{y}^T\widetilde{\Lambda}\mathbf{y}\\ &= \mathbf{d}^T\widetilde{\Lambda}\mathbf{d} - \mathbf{d}^T\widetilde{\Lambda}(\mathbf{X}^T\mathbf{w}) - (\mathbf{X}^T\mathbf{w})^T\widetilde{\Lambda}\mathbf{d} + (\mathbf{X}^T\mathbf{w})^T\widetilde{\Lambda}(\mathbf{X}^T\mathbf{w})\\ &= \mathbf{d}^T\widetilde{\Lambda}\mathbf{d} - 2\widetilde{\theta}_\lambda^T\mathbf{w} + \mathbf{w}^T\widetilde{\Psi}_\lambda\mathbf{w}\end{aligned}$$
$$\begin{aligned}\text{where } \widetilde{\Psi}_\lambda(n) &= \mathbf{X}(n)\widetilde{\Lambda}(n)\mathbf{X}^T(n)\\ \widetilde{\theta}_\lambda(n) &= \mathbf{X}(n)\widetilde{\Lambda}(n)\mathbf{d}(n)\end{aligned} \qquad \text{(eq 3.37)}$$

We then derive the gradient of the above expression for the cost function with respect to the filter tap weights. By forcing this to zero we then find the coefficients for the filter, $\overline{\mathbf{w}}(n)$, which minimises the cost function.

$$\widetilde{\Psi}_\lambda(n)\overline{\mathbf{w}}(n) = \widetilde{\theta}_\lambda(n)$$
$$\overline{\mathbf{w}}(n) = \widetilde{\Psi}_\lambda^{-1}(n)\widetilde{\theta}_\lambda(n) \qquad \text{(eq 3.38)}$$

The matrix Ψ(n) in the above equation can be expanded and then rearranged in a recursive form, we can then use the special form of the matrix inversion lemma of equation 2.10 to find an inverse for this matrix, which is required to calculate the tap weight vector update. The vector **k**(n) is known as the gain vector and is included in order to simplify the calculation.

$$\begin{aligned}
\widetilde{\Psi}_{\lambda}^{-1}(n) &= \lambda \widetilde{\Psi}_{\lambda}^{-1}(n-1) + \mathbf{x}(n)\mathbf{x}^T(n) \\
&= \lambda^{-1}\widetilde{\Psi}_{\lambda}^{-1}(n-1) - \frac{\lambda^{-2}\widetilde{\Psi}_{\lambda}^{-1}(n-1)\mathbf{x}(n)\mathbf{x}^T(n)\widetilde{\Psi}_{\lambda}^{-1}(n-1)}{1+\lambda^{-1}\mathbf{x}^T(n)\widetilde{\Psi}_{\lambda}^{-1}(n-1)\mathbf{x}(n)} \\
&= \lambda^{-1}(\widetilde{\Psi}_{\lambda}^{-1}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\widetilde{\Psi}_{\lambda}^{-1}(n-1)) \\
\text{where } \mathbf{k}(n) &= \frac{\lambda^{-1}\widetilde{\Psi}_{\lambda}^{-1}(n-1)\mathbf{x}(n)}{1+\lambda^{-1}\mathbf{x}^T(n)\widetilde{\Psi}_{\lambda}^{-1}(n-1)\mathbf{x}(n)} \\
&= \widetilde{\Psi}_{\lambda}^{-1}(n)\mathbf{x}(n)
\end{aligned}$$

(eq 3.39)

The vector θ_λ(n) of equation 3.37 can also be expressed in a recursive form. Using this and substituting Ψ^{-1}(n) from equation 3.39 into equation 3.38 we can finally arrive at the filter weight update vector for the RLS algorithm, as in equation 3.40.

$$\begin{aligned}
\widetilde{\theta}_{\lambda}(n) &= \lambda\widetilde{\theta}_{\lambda}(n-1) + \mathbf{x}(n)\mathbf{d}(n) \\
\overline{\mathbf{w}}(n) &= \widetilde{\Psi}_{\lambda}^{-1}(n)\widetilde{\theta}_{\lambda}(n) \\
&= \widetilde{\Psi}_{\lambda}^{-1}(n-1)\widetilde{\theta}_{\lambda}(n-1) - \mathbf{k}(n)\mathbf{x}^T\widetilde{\Psi}_{\lambda}^{-1}(n-1)\widetilde{\theta}_{\lambda}(n-1) + \mathbf{k}(n)d(n) \\
&= \overline{\mathbf{w}}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\overline{\mathbf{w}}(n-1) + \mathbf{k}(n)d(n) \\
&= \overline{\mathbf{w}}(n-1) + \mathbf{k}(n)(d(n) - \overline{\mathbf{w}}^T(n-1)\mathbf{x}(n)) \\
\overline{\mathbf{w}}(n) &= \overline{\mathbf{w}}(n-1) + \mathbf{k}(n)\overline{e}_{n-1}(n) \\
\text{where } \overline{e}_{n-1}(n) &= d(n) - \overline{\mathbf{w}}^T(n-1)\mathbf{x}(n)
\end{aligned}$$

(eq 3.40)

Implementation of the RLS algorithm.

As stated the previously the memory of the RLS algorithm is confined to a finite number of values, corresponding to the order of the filter tap weight vector. Firstly, two factors of the RLS implementation should be noted: the first is that although matrix inversion is essential to the derivation of the RLS algorithm, no matrix inversion calculations are required for the implementation, thus greatly reducing the amount of computational complexity of the algorithm. Secondly, unlike the LMS based algorithms, current variables are updated within the iteration they are to be used, using values from the previous iteration.

To implement the RLS algorithm, the following steps are executed in the following order.

1. The filter output is calculated using the filter tap weights from the previous iteration and the current input vector.

   $$\overline{y}_{n-1}(n) = \overline{\mathbf{w}}^T(n-1)\mathbf{x}(\mathbf{n}) \quad \text{(eq 3.41)}$$

2. The intermediate gain vector is calculated using equation 3.42.

   $$\mathbf{u}(n) = \tilde{\psi}_{\lambda}^{-1}(n-1)\mathbf{x}(n)$$
   $$\mathbf{k}(n) = \frac{1}{\lambda + \mathbf{x}^T(n)\mathbf{u}(n)}\mathbf{u}(n) \quad \text{(eq 3.42)}$$

3. The estimation error value is calculated using equation 3.43.

   $$\overline{e}_{n-1}(n) = d(n) - \overline{y}_{n-1}(n) \quad \text{(eq 3.43)}$$

4. The filter tap weight vector is updated using equation 3.43 and the gain vector calculated in equation 3.42.

   $$\mathbf{w}(n) = \overline{\mathbf{w}}^T(n-1) + \mathbf{k}(n)\overline{e}_{n-1}(n) \quad \text{(eq 3.44)}$$

5. The inverse matrix is calculated using equation 3.45.

   $$\psi_{\lambda}^{-1}(n) = \lambda^{-1}(\psi_{\lambda}^{-1}(n-1) - \mathbf{k}(n)[\mathbf{x}^{\mathrm{T}}(n)\psi_{\lambda}^{-1}(n-1)]) \quad \text{(eq 3.45)}$$

Each iteration of the RLS algorithm requires $4N^2$ multiplication operations and $3N^2$ additions (Farhang-Boroujeny p. 424). This makes its very costly to implement, thus LMS based algorithms, while they do not perform as well, are more favourable in practical situations.

4. MATLAB SIMULATIONS OF ADAPTIVE FILTERING ALGORITHMS.

Each of the adaptive filtering algorithms outlined in Section 3 were implemented using Matlab. The code for these implementations can be found in Appendix A, copies of the M-files have also been provided on the CD accompanying this thesis. In each simulation the echoed signal was generated by definfing an appropriate impulse response then convolving this with a vocal input wav file.

4.1 LMS Algorithm

Figure 4.1 shows the desired signal, adaptive output signal, estimation error and cost function (MSE) for the LMS algorithm with vocal input, FIR filter order of 1000 and step size of 0.007 (determined empirically). The MSE shows that as the algorithm progresses the average value of the cost function decreases, this corresponds to the LMS filters impulse response converging to the actual impulse response, more accurately emulating the desired signal and thus more effectively cancelling the echoed signal.

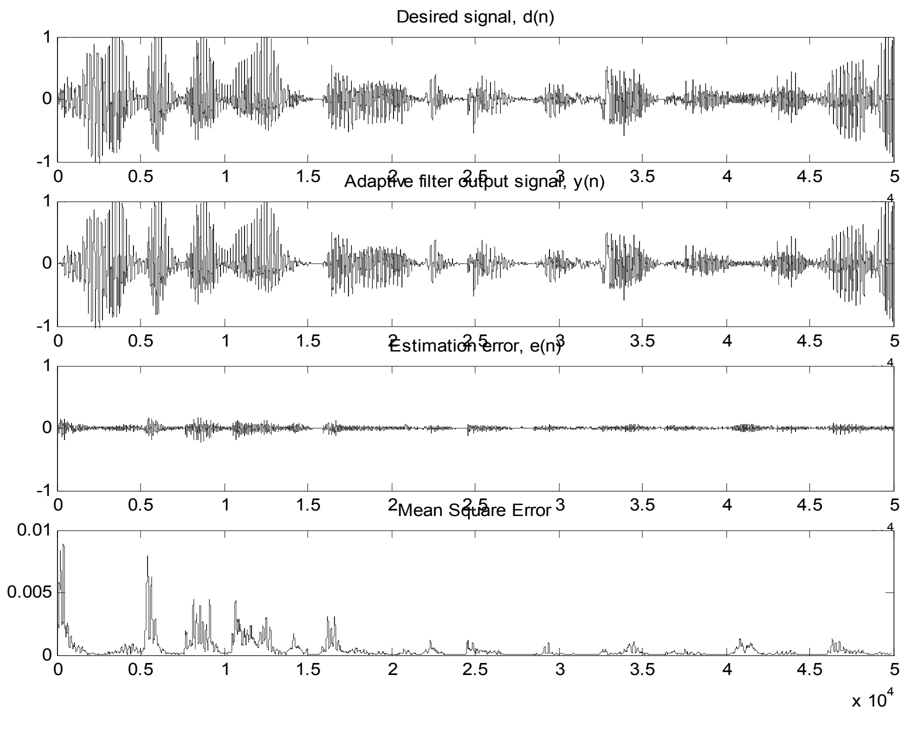


Figure 4.1: LMS algorithm outputs for vocal input, N=1000, μ=0.007.

The success of the echo cancellation can be determined by the ratio of the desired signal and the error signal. Figure 4.2 shows this attenuation as expressed in decibels. The average attenuation for this simulation of the LMS algorithm is –18.2dB.

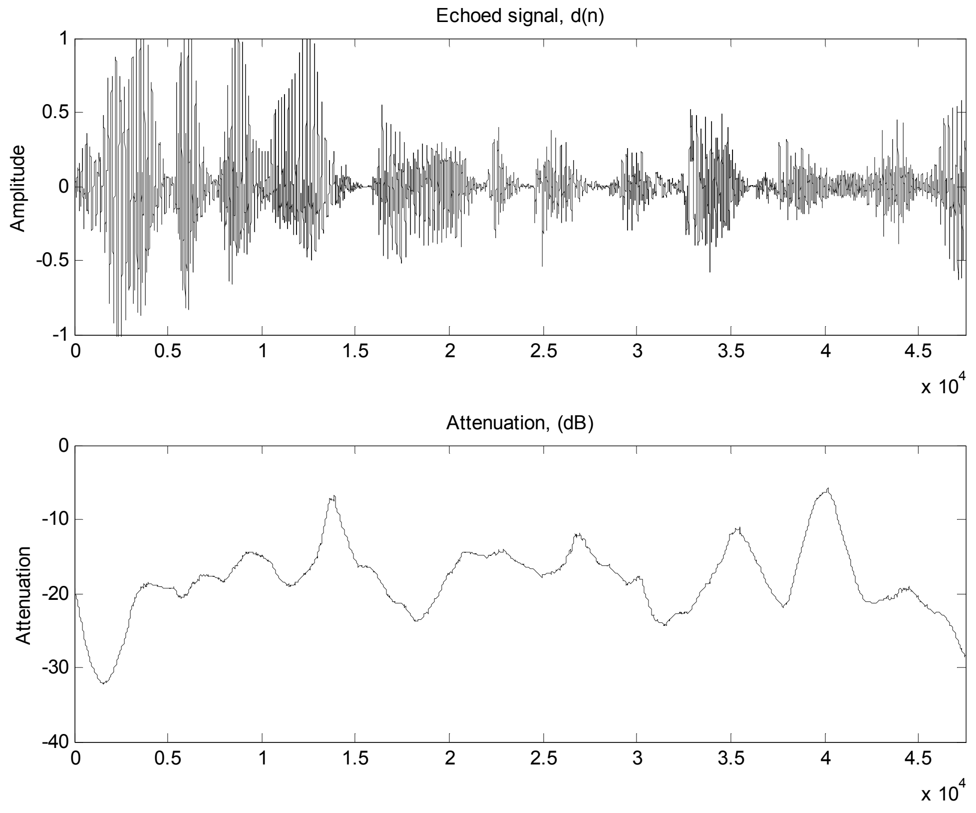


Figure 4.2: LMS attenuation of echoed signal (dB).

Figure 4.3 shows the estimated impulse response of the LMS algorithm at integer multiples of 7500 iterations. It can be seen that the adaptive filter more accurately represents the actual impulse response of the simulated echo as the algorithm progresses.

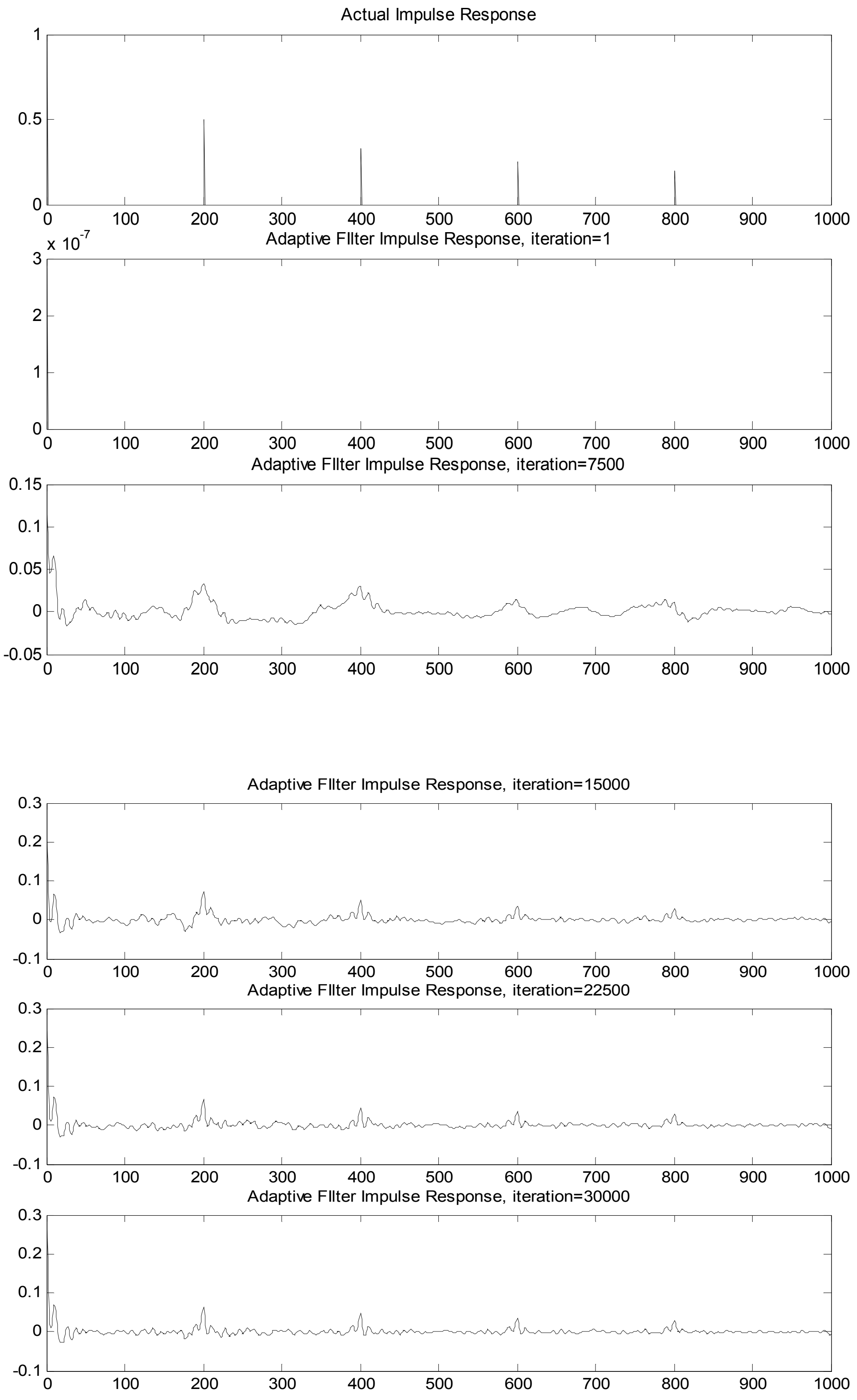


Figure 4.3: LMS adaptive filter impulse response

4.2 NLMS algorithm.

The normalised LMS algorithm was simulated using Matlab, it is also the algorithm used in the real time echo cancellation system. Figure 4.4 shows the results of the NLMS adaptive echo cancellation simulation.

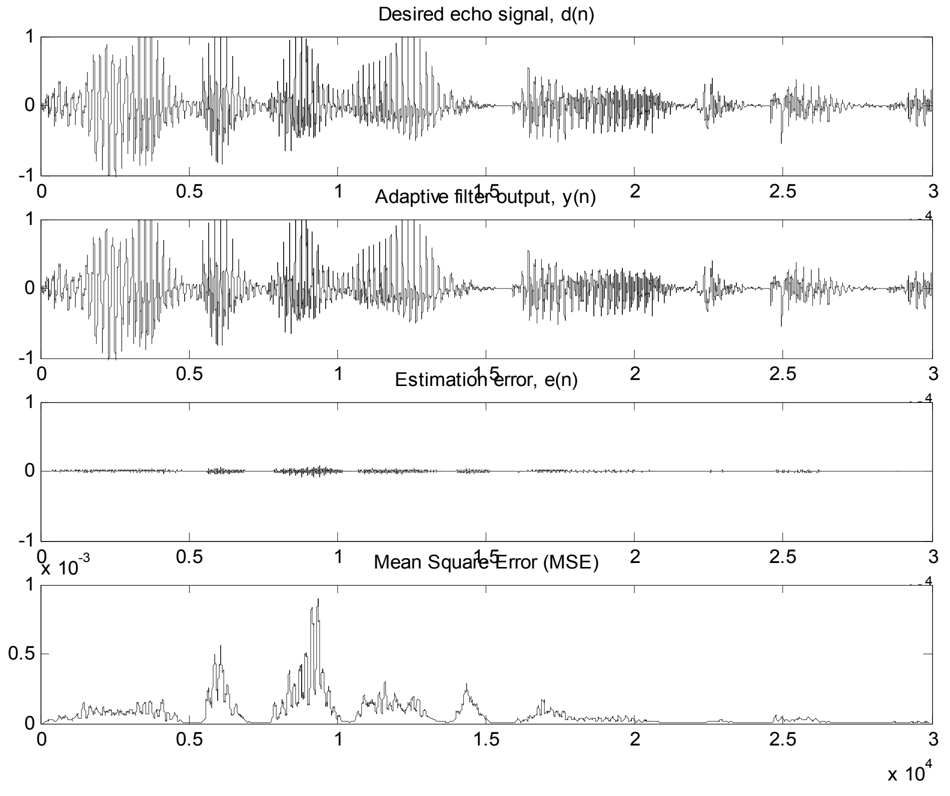


Figure 4.4. NLMS echo cancellation algorithm outputs.

Comparing this with figure 4.1, we can see that the error signal is smaller and the amplitude of the mean square error is in the order of ten times smaller than that of the standard LMS algorithm. This comes with the addition of only N more multiplication operations.

Figure 4.5 shows the dB attenuation of the echoed signal by the NLMS adaptive filter, it has an average attenuation of –27.9 dB, an approximate improvement of 10 dB over the standard LMS algorithm. This increase in performance, for marginal computational increase, as well as the increase in stability make it perfect for the real time echo cancellation system outlined in section 5.

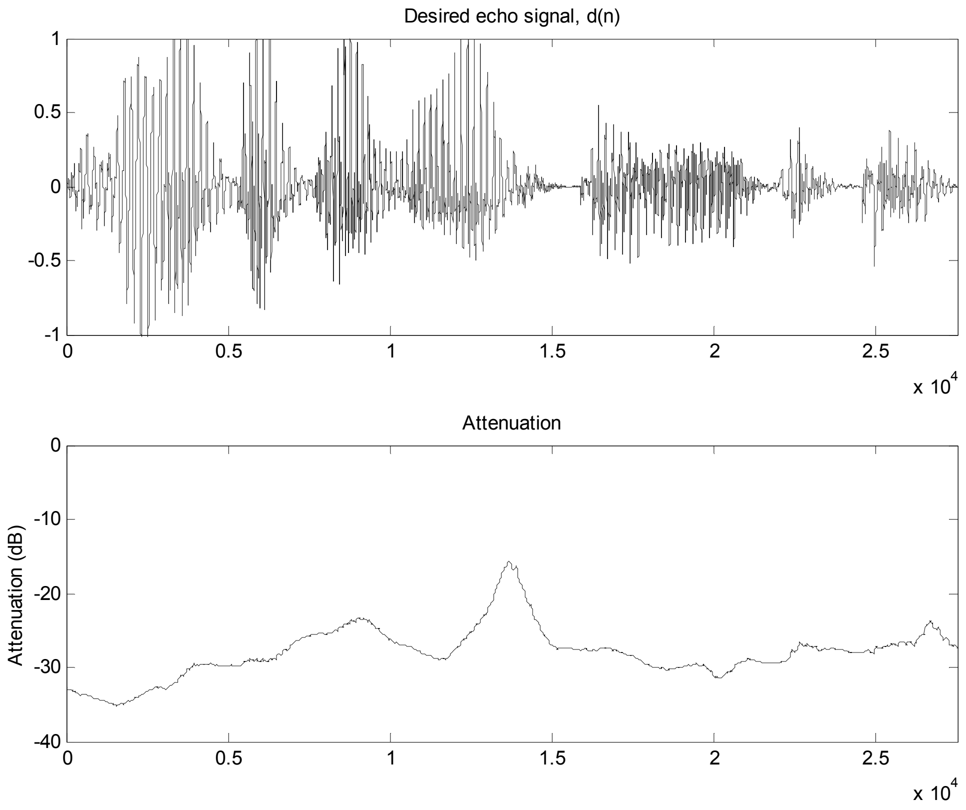


Figure 4.5. dB attenuation of NLMS echo cancellation.

Figure 4.6 shows the variation in the step size parameter (limited to 1 for visual clarity). As the step size is inversely proportional to the input signal energy levels, it peaks at times when the signal levels of d(n) are at a minimum.

The NLMS adaptive filter impulse response after 30000 iterations is shown in figure 4.7. Comparing this to figure 4.3, the NMLS impulse response has peaks of approximately twice the amplitude of the LMS algorithm after the same number of iterations, this shows the convergence rate of the NLMS algorithm is greater than that of the standard LMS algorithm.

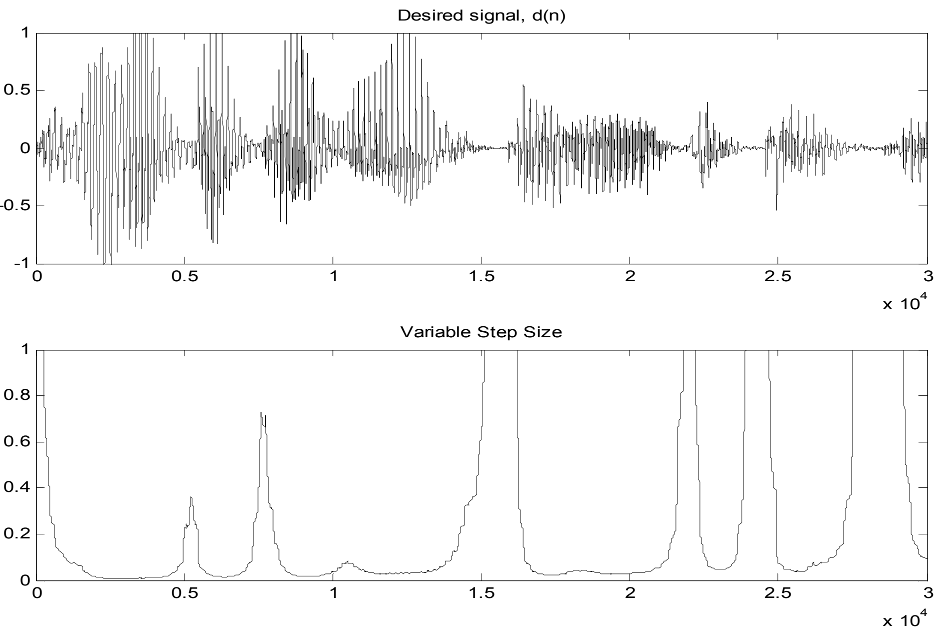


Figure 4.6. Variation in the NLMS step size.

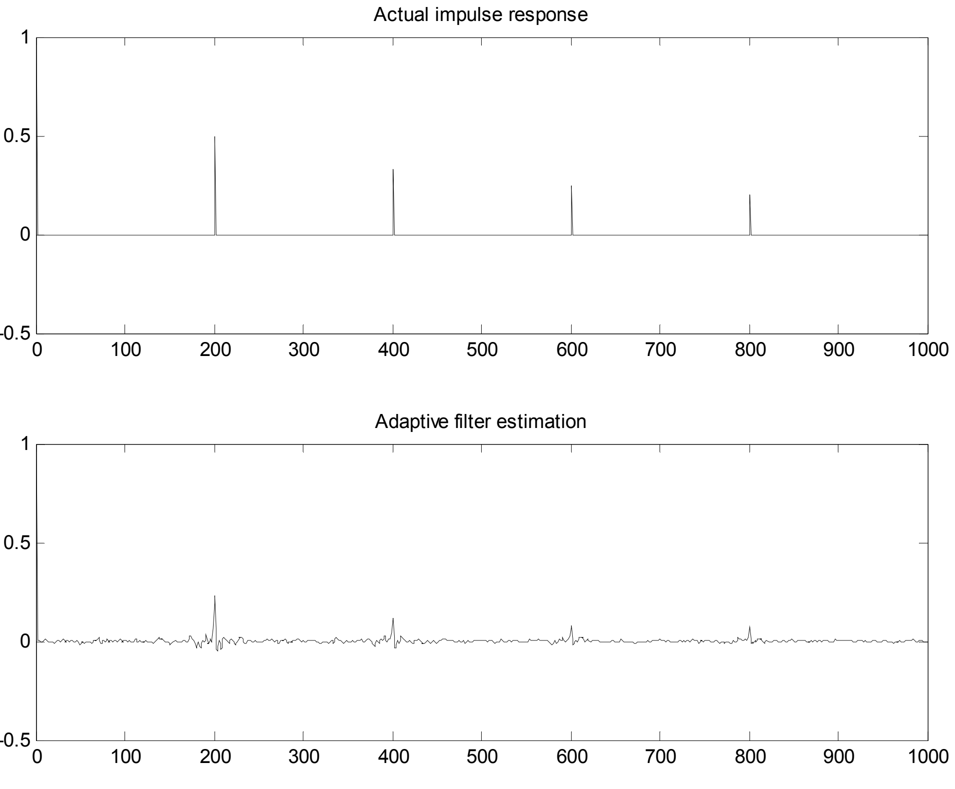


Figure 4.7. Adaptive filter impulse response estimation.

4.3 VSLMS algorithm.

The variable step size LMS algorithm was simulated in Matlab, figure 4.8 shows the results of this simulation. Comparing this with the LMS and NLMS algorithms, the VSLMS algorithm performs poorly having a peak mean square error approximately four times greater than that of the standard LMS algorithm. This is very poor considering each iteration of the VSLMS algorithm has 2N more multiplication operations than the LMS algorithm. This is possibly due to speech signals being non-stationary, regardless the VSLMS aglorithm was not considered for the real time application.

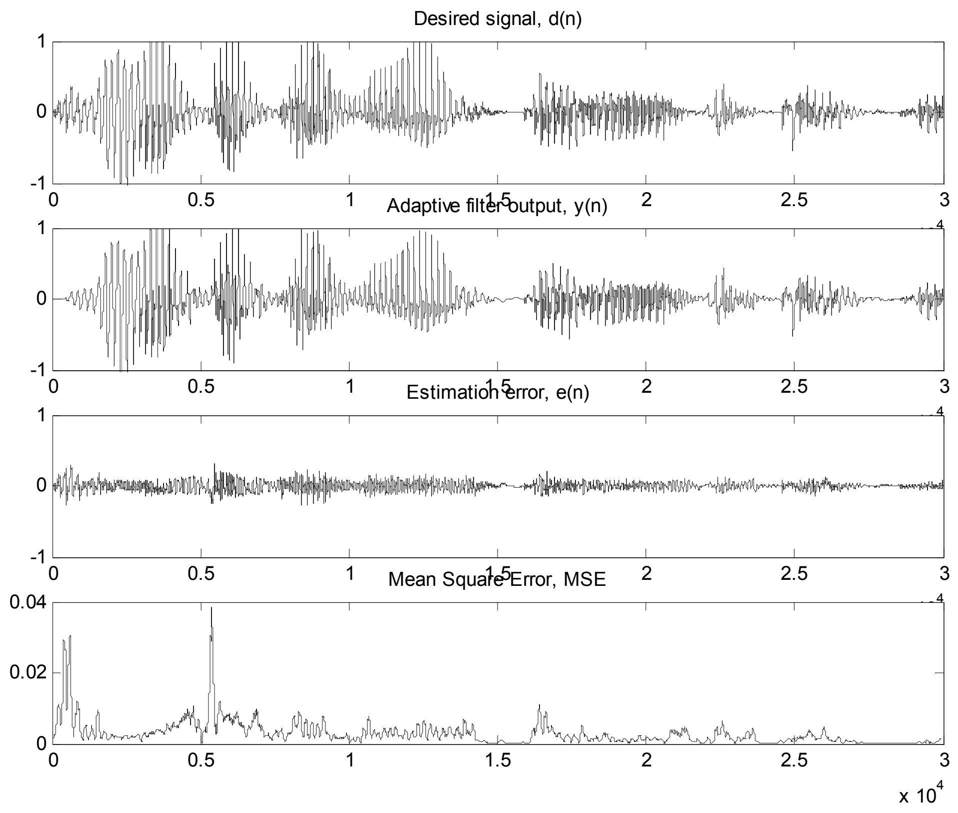


Figure 4.8. VSLMS simulation outputs.

Figure 4.9 shows the attenuation performance of the VSLMS algorithm, again it shows the algorithms poor performance. It has an average attenuation of –9.8 dB almost half that of the standard LMS algorithm.

Figure 4.10 shows the estimation of the impulse response by the VSLMS adaptive filtering algorithm, again demonstrating its poor performance for speech echo cancellation.

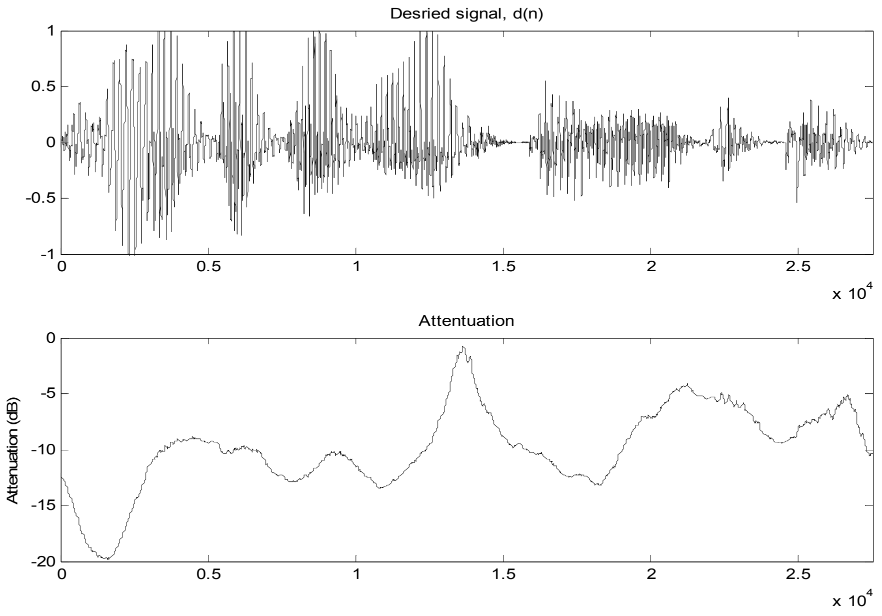


Figure 4.9. VSLMS echoed signal attenuation.

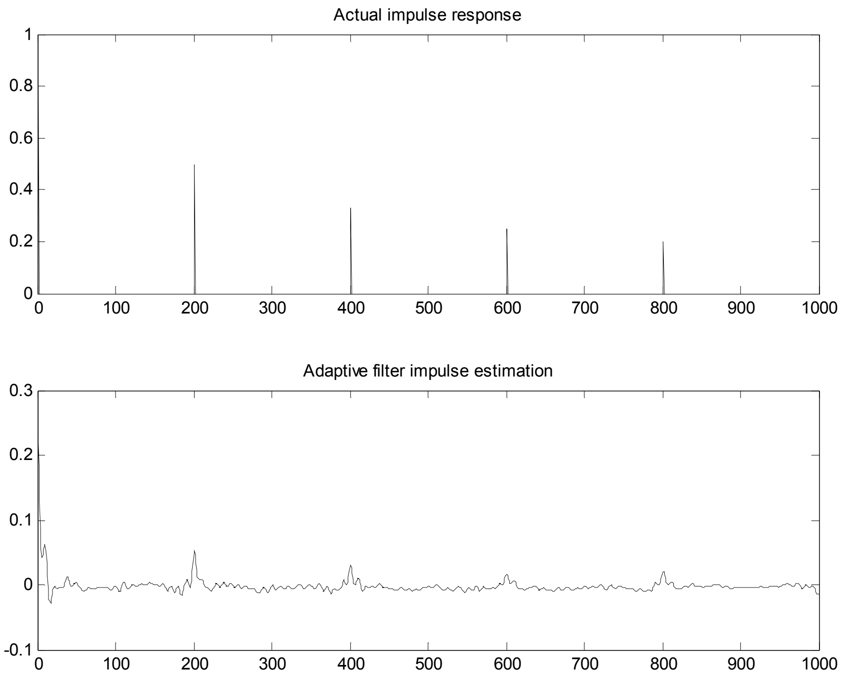


Figure 4.10. VSLMS impulse response estimation.

4.4 VSNLMS algorithm.

Figure 4.11 shows the outputs of the variable step size normalised LMS algorithm proposed in section 3.1.4. It shows 30000 iterations of the algorithm with a minimum step size of 0, and a ρ value of 1. It can be seen that the results are virtually idenitcal to those for the VSLMS algorithm. This is because although the upper step size boundary is calculated for each iteration, this does not calculate the actual step size. The step size is calculated by equation 2.41, this value may never actually reach the limit calculated by the VSNLMS algorithm. This extra calculation leads to no noticeable gain in performance.

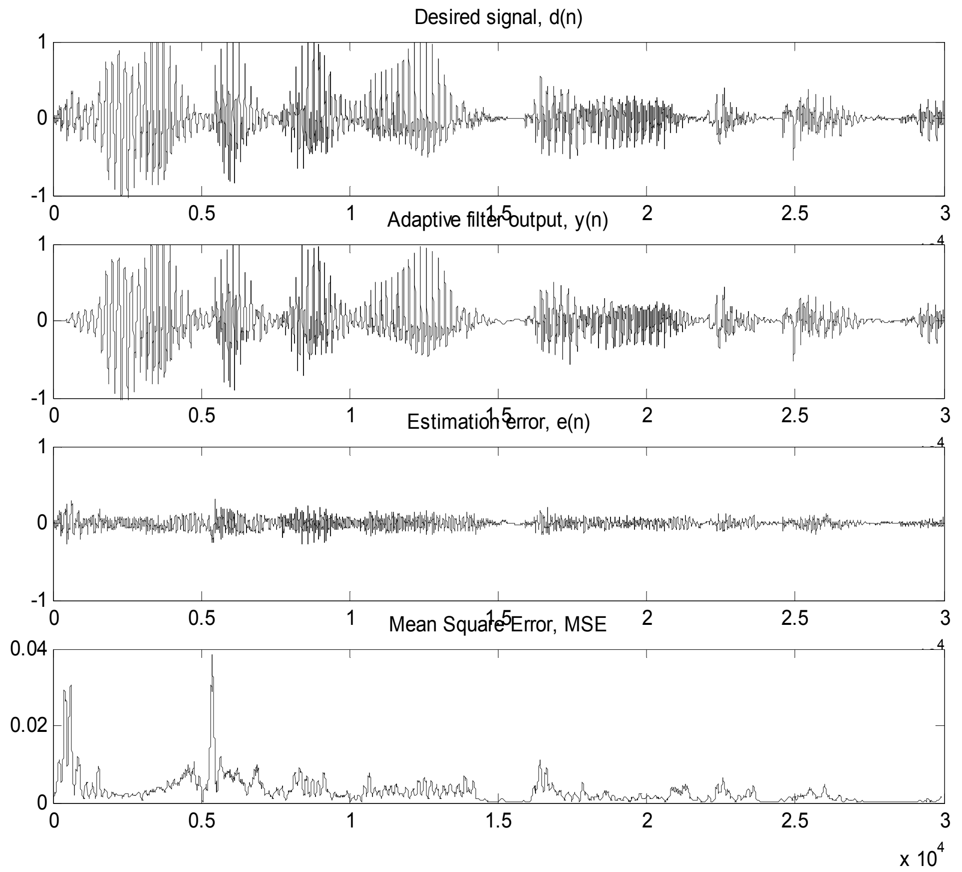


Figure 4.11: Outputs of the VSLMS adaptive filter

Figure 4.12 shows the attenuation of the VSNLMS algorithm, comparing this to figures 4.5 and 4.2 we can see that it, like the VSLMS algorithm, does not perform nearly as well as the NLMS or standard LMS algorithms. The VSNLMS algorithm has an average attenuation of –9.9 dB, this is relatively weak performance for 5N+1 multiplication operations, thus ruling it out of consideration for the echo cancellation system.

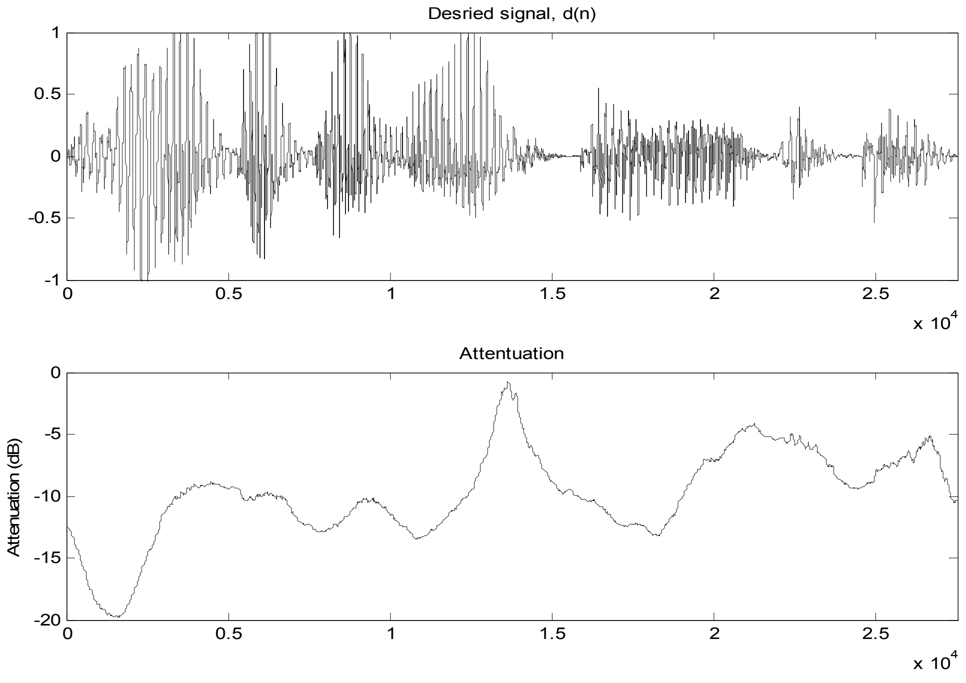


Figure 4.12. Attenuation of the echoed signal by the VSNLMS algorithm.

4.5 RLS algorithm.

The RLS algorithm was simulated using Matlab. This algorithm proved to be very effective and outperformed all other algorithms. Figure 4.13 shows the outputs of the simulation, it can be seen that the error estimation signal is very small and the average of the MSE quickly decays almost to zero. Note that MSE is not the cost function of the RLS algorithm, however it was used for easier comparison with the other algorithms.

Figure 4.14 shows the RLS adaptive filter echo signal attenuation. It is the greatest out of all the algorithms with an average attenuation of –34.3 dB. Unfortunately, each iteration of the RLS algorithm requires $4N^2$ multiplication operations. Far more than any of the other algorithms.

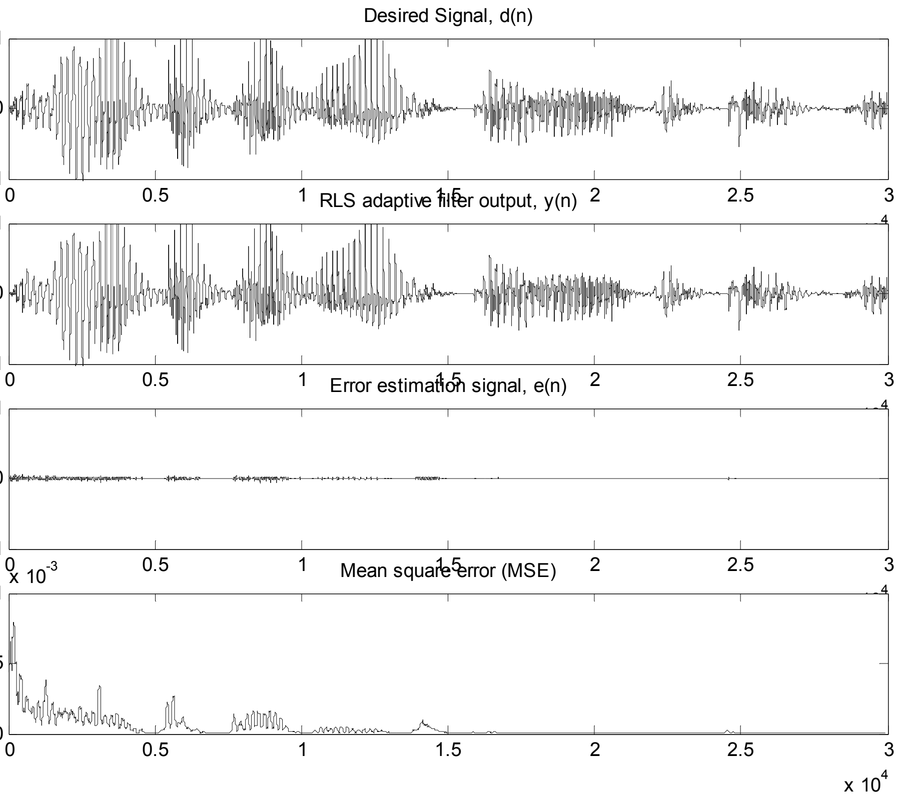


Figure 4.13. Results of the RLS simulation.

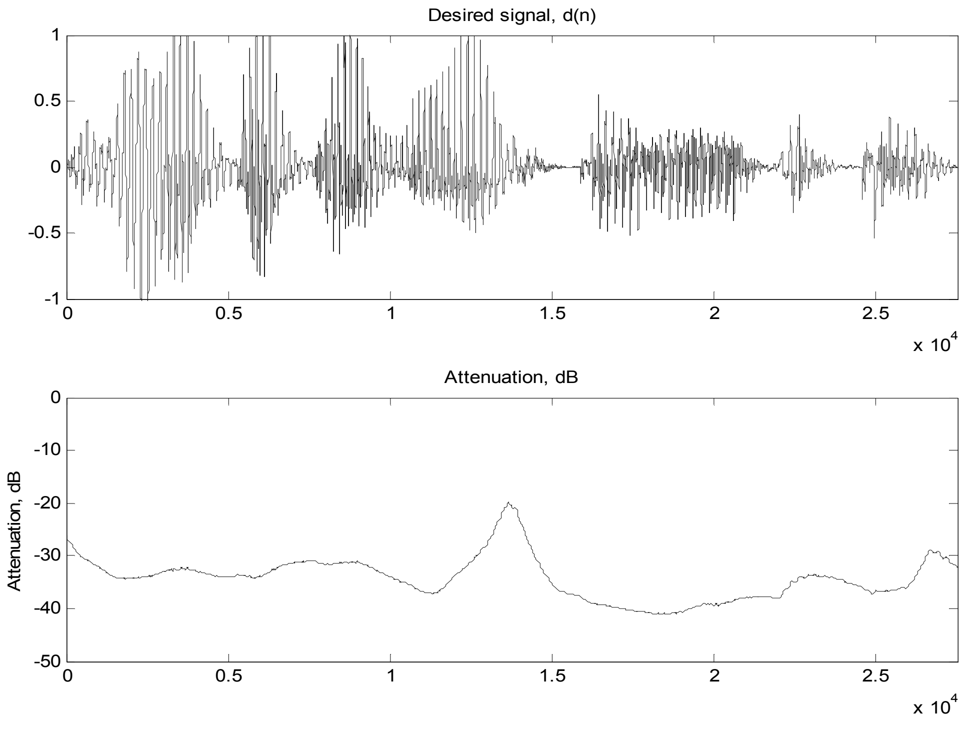


Figure 4.14: RLS algorithm echo signal attenuation.

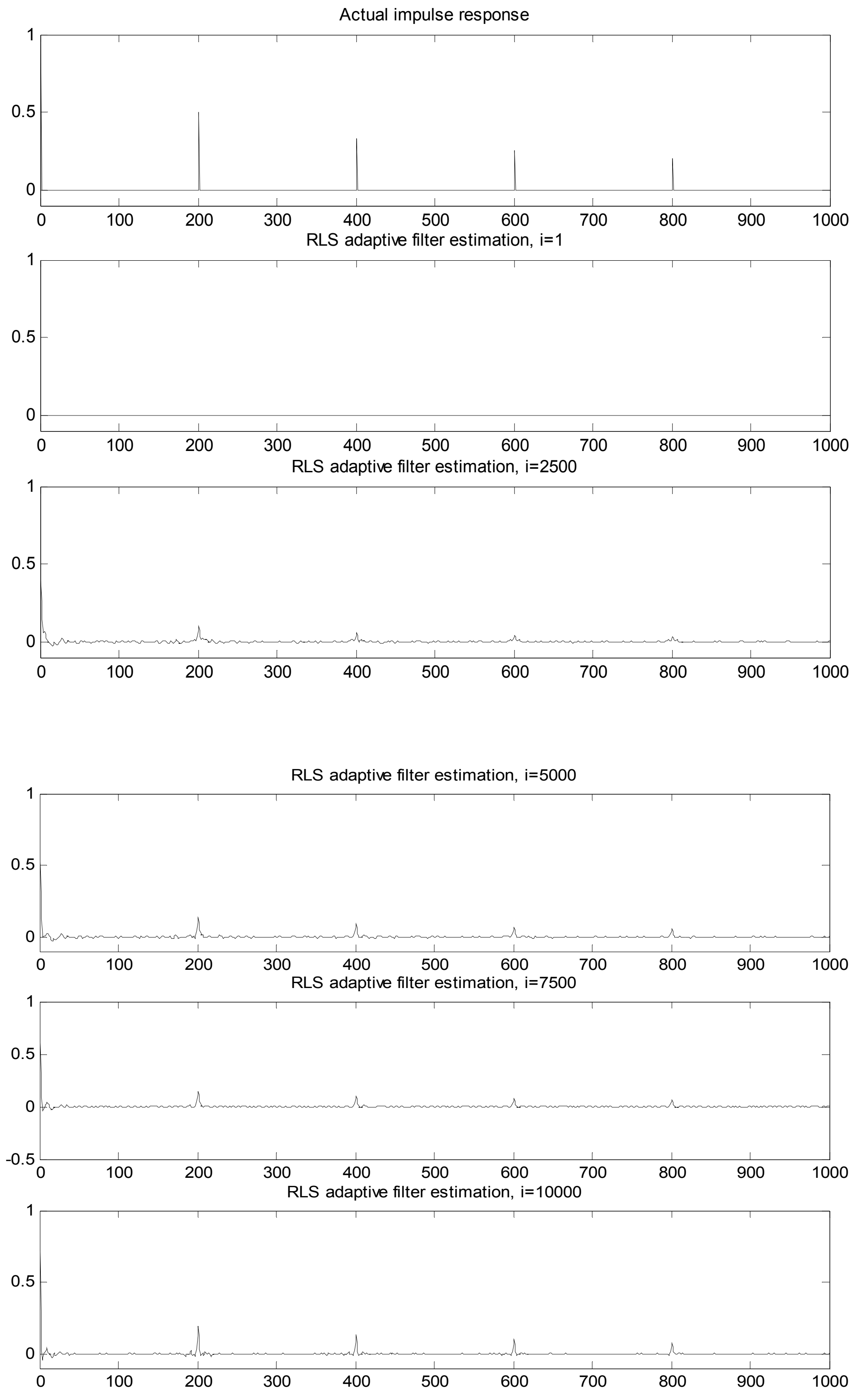


Figure 4.15: Convergence of the RLS adaptive filter to actual impulse response.

Figure 4.15 shows the impulse response of the RLS adaptive filter at integer multiples of 2500 iterations. Comparing this with figure 4.3 it is clear that the RLS algorithm has a far greater convergence rate that the LMS algorithm. However, this performance comes at the cost of computational complexity. As stated above, each iteration requires $4N^2$ multiplications. For echo cancellation systems the FIR filter order is usually in the thousands. With at least 4 million multiplications required per sample period the RLS algorithm is too costly to implement. In practice the LMS based algorithms, although poorer performers, are preferred.

5. REAL TIME IMPLEMENTATION

5.1 Texas Instruments TMS320C6711 development board.

The TI TMS320C6711 DSP starter kit (DSK) is a digital signal processing development tool used to prototype DSP applications designed for the C6711 family of processors. It has a single mono input and output, with analog to digital conversion executed by the onboard AD535 codec. The AD535 codec uses sigma-delta conversions at a fixed sample rate of 8kHz. The DSK also includes 16MB of synchronous dynamic RAM and 128 kB of flash ROM, (Chassaing 2002, p 4).

The DSK board is connected to a PC via a parallel port. The PC uses TI code composer studio to create program files which are then loaded onto the DSK, control over parameters of the echo cancellation system can also be achieved using the CCS.

5.2 C6711 Digital Signal Processor.

The heart of the TI development board is the C6711 floating-point digital signal processor. It is based on the very long instruction word (VLIW) architecture making it suitable for numerically intensive algorithms. The structure of the internal program memory is such that eight instructions can be fetched every cycle. Additional features of the C6711 include 72kB of internal memory, eight functional or execution units consisting of six ALUs and two multiplier units, a 32-bit address bus to address 4 gigabytes and two sets of 32-bit general purpose registers (Chassaing 2002, p 4).

5.3 TI Code Composer Studio

The Texas Instruments Code Composer Studio is an integrated development environment included with the TMS320C6711 development kit. It contains a C compiler, debugger, assembler and linker. The real time echo cancellation algorithm was written in C using CCS. There is also the facility to modify the application in assembly language once they have been compiled.

The CCS has inbuilt code optimisation options which rapidly increase the execution speed of an application without requiring the developer to create or modify assembly code. By using these optimisation options the echo cancellation system was designed to a suitable level of performance without needing to code in assembly. If time had permitted, further optimisation would have been performed at the assembly level.

TI CCS allows the user to produce graphical objects such as sliders using General Extension Language (GEL) format files. GEL files allow the developer to perform functions such as variable modification in real time without needing to recompile the source code.

CCS also has facilities to watch variables for monitoring and debug purposes. Also variables can be visualised using the CCS graphing function. The echo cancellation system uses support files that are included with the CCS package. It also utilises support files from the CD accompanying Chassaing's 'DSP applications using C and the TMS320C6x DSK'.

5.4 Echo Cancellation System

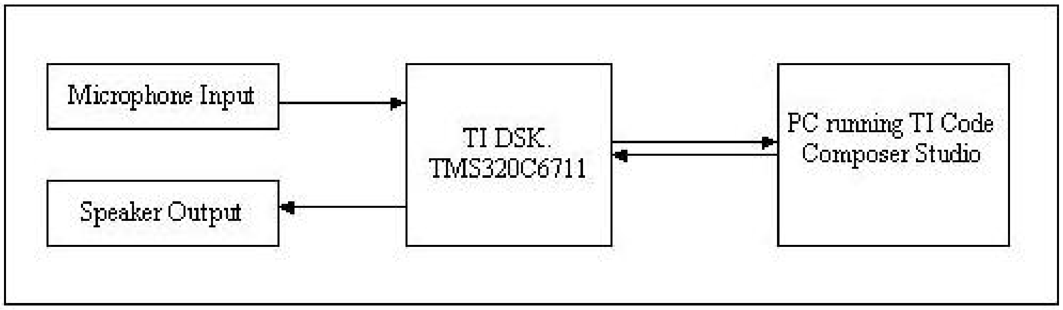


Figure 5.1. Block diagram of echo cancellation system.

The real-time acoustic echo cancellation system has been implemented as shown in figure 5.1. A microphone is used to input a voice signal from the user, this is then fed in to the Texas Instrument TMS320C6711 development board. Here it is sampled by the AD535 codec at a fixed rate of 8 kHz. The first operation performed by the TMS320C6711 is a simulation of an echo response of a real acoustic environment. This is achieved by storing each value of the sampled input in a buffer whose index is incremented with each sample instance. When the buffer index reaches the end of the array, the index is reset. The echoed signal is generated by adding the instantaneous sample value and the stored values in the buffer, this signal is then fed back into the buffer resulting in multiple echoes. The amount of time delay the echoed signal contains is determined by the length of the buffer as given in equation 5.1. The buffer length of the echo cancellation system can be adjusted between 300 and 600, giving a time delay in the range of 37.5 and 75 ms. A round trip time delay of 75ms emulates the effect of sound reflecting off an object approximately 12 metres away. This echoed signal is what the adaptive filter attempts to emulate. The better it does this, the more effective the cancellation will be.

$$\begin{aligned} t_d &= \frac{\text{buffer length}}{\text{sample rate}} \\ &= \frac{\text{buffer length}}{8000} \qquad \text{(eq 5.1)} \end{aligned}$$

The algorithm used on the echo cancellation system is the normalised LMS algorithm detailed in section 3.4. At each new sample time the input is sampled and input to an array representing the input vector **x**(n). The step size value for this iteration of the NLMS algorithm is then calculated by inverting the dot product of the input vector and itself. Equation 3.19, shows that a small constant should be included in the denominator to avoid zero divides, however due to the noise inherent in the microphone input and DSK board this was found to be an unnecessary calculation.

The output of the adaptive filter is then calculated by the dot product of the input vector and the current filter tap weight vector. This output is then subtracted from the desired signal to determine the estimation error value, e(n). This error value, the step size value and the current FIR tap weight vector are then input to the NLMS algorithm to calculate the filters tap weight to be used in the next iteration.

Using the CCS graphing function the impulse response emulated by the echo cancellation system can be viewed, this is shown in figure 5.2 below. As can be seen the system emulates the impulse response of the desired signal. Figure 5.2 shows impulse peaks of decreasing magnitude at integer multiples of 600 samples, corresponding to a delay time of 75 milliseconds.

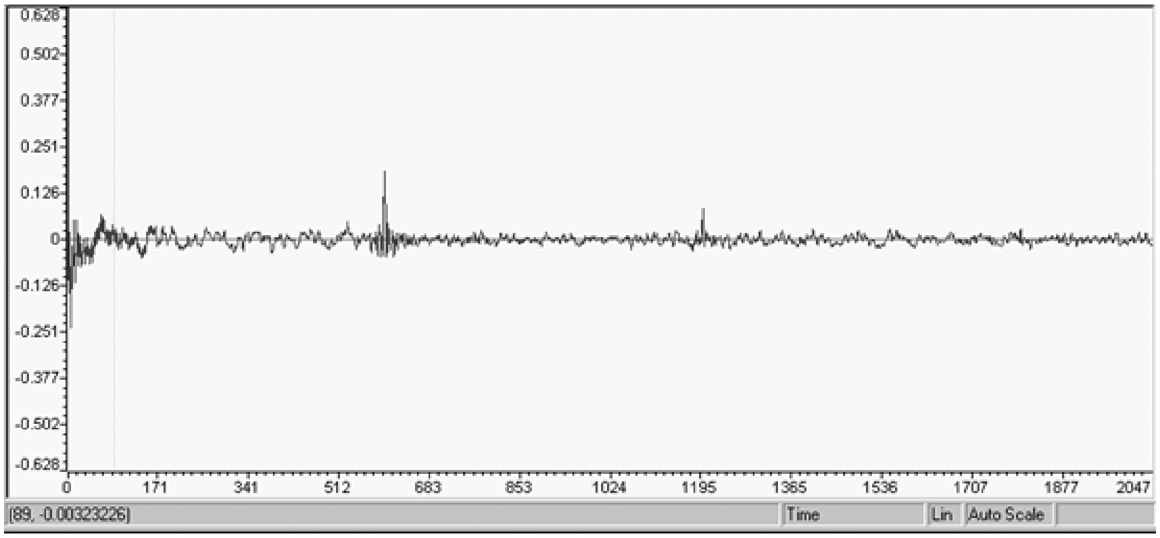
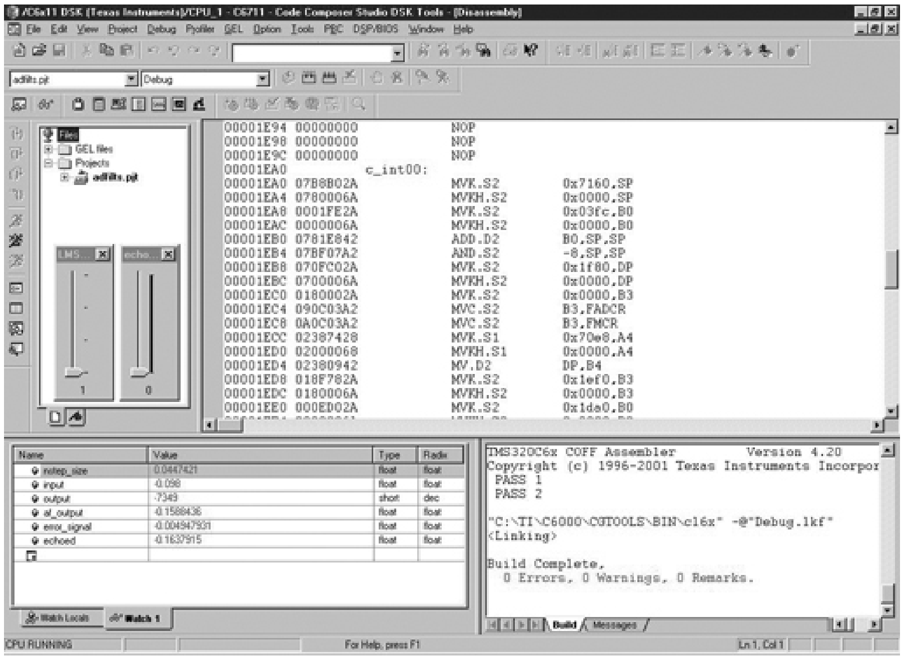


Figure 5.2. CCS plot of adaptive filter impulse response, delay =600 samples.

Using a GEL slider the user can then choose to listen to either the echo cancelled output, the adaptive filter output, the echoed input signal or the clean input signal. Another GEL slider can be used to vary the length of the echo buffer and hence the time delay of the echo. The CCS application window including these GEL files can be seen in figure 5.3.


/C6x11 DSK (Texas Instruments)/CPU_1 - C6711 - Code Composer Studio DSK Tools - [Disassembly]
File Edit View Project Debug Profiler GEL Option Tools PBC DSP/BIOS Window Help
adfilts.pjt
Debug
Files
GEL files
Projects
adfilts.pjt
LMS...
echo...
1
0
Watch Locals
Watch 1
Build
Messages
CPU RUNNING
For Help, press F1
Ln 1, Col 1

```
00001E94 00000000              NOP
00001E98 00000000              NOP
00001E9C 00000000              NOP
00001EA0           c_int00:
00001EA0 07B8B02A              MVK.S2       0x7160,SP
00001EA4 0780006A              MVKH.S2      0x0000,SP
00001EA8 0001FE2A              MVK.S2       0x03fc,B0
00001EAC 0000006A              MVKH.S2      0x0000,B0
00001EB0 0781E842              ADD.D2       B0,SP,SP
00001EB4 07BF07A2              AND.S2       -8,SP,SP
00001EB8 070FC02A              MVK.S2       0x1f80,DP
00001EBC 0700006A              MVKH.S2      0x0000,DP
00001EC0 0180002A              MVK.S2       0x0000,B3
00001EC4 090C03A2              MVC.S2       B3,FADCR
00001EC8 0A0C03A2              MVC.S2       B3,FMCR
00001ECC 02387428              MVK.S1       0x70e8,A4
00001ED0 02000068              MVKH.S1      0x0000,A4
00001ED4 02380942              MV.D2        DP,B4
00001ED8 018F782A              MVK.S2       0x1ef0,B3
00001EDC 0180006A              MVKH.S2      0x0000,B3
00001EE0 000ED02A              MVK.S2       0x1da0,B0
```

Name	Value	Type	Radix
mstep_size	0.0447421	float	float
input	-0.098	float	float
output	-7349	short	dec
af_output	-0.1588436	float	float
error_signal	-0.004947931	float	float
echoed	-0.1637915	float	float

```
TMS320C6x COFF Assembler        Version 4.20
Copyright (c) 1996-2001 Texas Instruments Incorpor
 PASS 1
 PASS 2

"C:\TI\C6000\CGTOOLS\BIN\cl6x" -@"Debug.lkf"
<Linking>

Build Complete,
  0 Errors, 0 Warnings, 0 Remarks.
```

Figure 5.3, CCS application window.

A copy of the source code used in the echo cancellation system can be found on the CD accompanying this thesis.

6. SUMMARY AND RECOMMENDATIONS FOR FUTURE WORK.

A summary of the performance of the adaptive filtering algorithms is expressed in figure 6.1. It can be seen that when considering the attenuation values and the number of multiplication operations for each algorithm, the NLMS algorithm is the obvious choice for the real time acoustic echo cancellation system. Additionally, it does not require a prior knowledge of the signal values to ensure stability.

Algorithm	**Average attenutaion**	**Multiplication operations**	**Comments**
LMS	-18.2 dB	2N+1	Is the simplest to implement and is stable when the step size parameter is selected appropriately. This requires prior knowledge of the input signal which is not feasible for the echo cancellation system.
NLMS	-27.9dB	3N+1	Simple to implement and computationally efficient. Shows very good attenuation, and variable step size allows stable performance with non-stationary signals. This was the obvious choice for real time implementation.
VSLMS	-9.8 dB	4N+1	Displays very poor performance, possibly due to non-stationary nature of speech signals. Only half the attenuation of the standard LMS algorithm. Not considered for real time implementation.
VSNLMS	-9.9 dB	5N+1	Increase in multiplications gives negligible improvement in performance over VSLMS algorithm.
RLS	-34.2 dB	$4N^2$	Has the greatest attenuation of any algorithm studied, and converges much faster than the LMS algorithm. This performance comes at the cost of computational complexity and considering the large FIR order required for echo cancellation, this is not feasible for real time implementation.

Figure 6.1. Summary of adaptive filter algorithm performances.

The real time acoustic echo cancellation system was successfully developed with the NLMS algorithm. The system is capable of cancelling echo with time delays of up to 75 ms, corresponding to reverberation off an object a maximum of 12 metres away. This proves quite satisfactory in emulating a medium to large size room.

There are many possibilities for further development in this discipline, some of these are as follows.

- The real time echo cancellation system was implemented successfully using the TI TMSC6711 DSK. However, this system can only cancel those echoes that fall within the length of the adaptive FIR filter. The length of this filter is currently at the maximum allowable without assembler coding. Coding in assembler could allow for further optimisation and an improvement in the systems performance.
- The echo cancellation system was developed on the DSK board, this has certain parameters such as sampling rate that are unalterable by the developer. Another possible way to increase the performance of the echo cancellation system is to use the C6711 digital signal processor in a custom made circuit which could properly utilise its full potential.
- This thesis dealt with transversal FIR adaptive filters, this is only one of many methods of digital filtering. Other techniques such as infinite impulse response (IIR) or lattice filtering may prove to be more effective in an echo cancellation application.
- The algorithms studied in this thesis perform best under purely stationary signal conditions. Strictly speaking speech does not fall into this catergory. Further work could be done in developing techniques specifically designed for non-stationary speech signals.

I feel that the goals of this thesis project have been accomplished. Unfortunately, like all things there is only so far one can progress in a finite amount of time. The field of digital signal processing and in particular adaptive filtering is vast and further research and development in this discipline could only lead to an improvement on the methods for acoustic echo cancellation systems studied in this project.

Again, my sincerest thanks go to my thesis supervisor, Dr Vaughan Clarkson, whose assistance was invaluable throughout this project.

7. BIBLIOGRAPHY

Bellanger, Maurice G., 2001. *Adaptive Digital Filters,* 2nd edition. Marcel Dekker Inc., New York.

Chassaing, Rulph. 2002, *DSP applications using C and theTMS320C6x DSK.* John Wiley and Sons, New York.

Cowan, C.F.N., Grant, P.M. 1985, *Adaptive Filters.* Prentice-Hall, Inc. New Jersey.

Diniz, Paulo S. R. 1997, *Adaptive Filtering, Algorithms and Practical Implementation.* Kluwer Academic Publishers, Boston.

Farhang-Boroujeny, B. 1999, *Adaptive Filters, Theory and Applications.* John Wiley and Sons, New York.

Gottfried, Byron. 1996, *Programming with C.* Schaum's Outlines, 2nd edition. McGraw-Hill Book Co., New York.

Hancock, Les., Krieger, Morris., Zamir, Saba.1990, *The C primer,* 3rd edition. McGraw-Hill, Inc. New York.

Haykin, Simon. 1991, *Adaptive Filter Theory.* 2nd Edition. Prentice-Hall Inc., New Jersey.

Homer, J. 1994, *Adaptive Echo Cancellation in Telecommunications*. PhD thesis, University of Newcastle , Newcastle.

Honig, Michael L., Messerschmitt, David G. 1984, *Adaptive Filters, Structures, Algorithms and Applications.* Kluwer Academic Publishers, Boston.

Oppenheim, Alan V., Schafer, Ronald W. 1989, *Distcrete-Time Signal Processing,* International edition. Prentice-Hall, Inc., New Jersey.

Texas Instruments: Code Composer Studio/ TMS320C6711, 2002. Documentation CD accompanying theTMS320C6711 DSP Starter Kit, Texas Instruments.

APPENDIX A: MATLAB IMPLEMENTATION CODE.

This appendix contains the source code for the Matlab simulations used in section 4.

They can be found on the CD accompanying this thesis.

A.1 LMS.m (LMS ALGORITHM)

```
function [input_signal, error_signal, desired_signal, filter_output, impulse, filter_current,
mse, db, db_avg]=LMS(filter_size, step_size, input_file, iterations)
% Function to perform the LMS algorithm on an input file.
% Inputs:   Filter order, step size, input wav file, number of iterations.
% Outputs:  Input signal, error estimation signal (echo cancelled), desired signal (echoed
signal), adaptive filter output, real impulse response
%           Estimation of impulse response, mean sqaure error, attenuation (dB), average
attenuation.

%Read in the input file
input_signal = wavread(input_file);
% Create the impulse response for the desired signal
impulse=zeros(filter_size,1);
for (i=1:5)
   impulse(((i-1)*filter_size/5)+1)=1/i;
end

% Convolve the impulse with the input signal to generate the desired signal
desired_signal = conv(input_signal, impulse);

% initialise adaptive filter impulse and input vector to zero vector of length specified at
command line
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);

% Loop for number of iterations specified in command line.
for i=1:iterations
   i
   input_vector(1)=input_signal(i);        % insert new sample at beginning of input
vector.
   filter_output(i)=dot(filter_current, input_vector);         %Caluculate adaptive filter
output
   error= desired_signal(i)-filter_output(i)       % Calculate estimation error
   filter_current = filter_current + 2*step_size*error*input_vector;       % Update filter
taps by LMS recursion

   % Shift values in input vector along.
   for j=filter_size:-1:2
```

```
        input_vector(j)=input_vector(j-1);
    end

    error_signal(i)=error;      % store estimation error
    cost(i)=error*error;        % calculate instantaneous cost square error

end

% Find moving average of error squared.
for i=1:iterations-100
    mse(i)=mean(cost(i:i+100));
    i
end

%find moving average of db attenuation (averaged to smooth output).
for i=1:iterations-2500
    db(i)=-
20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    i
end

%find total average db attenuation
db_avg=mean(db);
```

A.2 NLMS.m (NLMS ALGORITHM)

```
function [input_signal, error_signal, desired_signal, filter_output, impulse, filter_current,
mse, db, db_avg, ss]=NLMS(filter_size, input_file, iterations)
% Function to perform the NLMS algorithm on an input file.
% Inputs:   Filter order, input wav file, number of iterations.
% Outputs:  Input signal, error estimation signal (echo cancelled), desired signal (echoed
signal), adaptive filter output, real impulse response
%          Estimation of impulse response, mean square error, attenuation (dB), average
attenuation, step size.

% Read in the input file
input_signal = wavread(input_file);

%Generate acoustic impulse response and convolve to get echoed signal
impulse=zeros(filter_size,1);
for (i=1:5)
   impulse(((i-1)*filter_size/5)+1)=1/i;
end
desired_signal = conv(input_signal, impulse);

% initialise variables
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);

% perform the NLMS algorithm for set number of iterations
for i=1:iterations
   i
   % get input sample
   input_vector(1)=input_signal(i);
   filter_output(i)=dot(filter_current, input_vector);
   error= desired_signal(i)-filter_output(i)
   % calculate step size
   step_size=1/(dot(input_vector, input_vector))
   % update filter tap weight vector
   filter_current = filter_current + step_size*error*input_vector;
   % shift input vector
   for j=filter_size:-1:2
      input_vector(j)=input_vector(j-1);
   end
   error_signal(i)=error;
   cost(i)=error*error;
   ss(i)=step_size;
end

% Find moving average of error squared.
```

```
for i=1:iterations-100
   mse(i)=mean(cost(i:i+100));
   i
end

%find moving average of db attenuation (averaged to smooth output).
for i=1:iterations-2500
   db(i)=-
20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
   i
end

%find total average db attenuation
db_avg=mean(db);
```

A.3 VSLMS.m (VSLMS ALGORITHM)

```
function [input_signal, error_signal, desired_signal, filter_output, impulse, filter_current,
mse, db, db_avg]=VSLMS(filter_size, ss_upper,ss_lower, input_file, iterations)
% Function to perform the VSLMS algorithm on an input file.
% Inputs:   Filter order, step size, stepsize upper bound, stepsize lower bound, input wav
file, number of iterations.
% Outputs:  Input signal, error estimation signal (echo cancelled), desired signal (echoed
signal), adaptive filter output, real impulse response
%           Estimation of impulse response, mean sqaure error, attenuation (dB), average
attenuation.

% read in input wav file
input_signal = wavread(input_file);

% generate impulse response of acoustic environment and convolve with input to achieve
desired echo signal
impulse=zeros(filter_size,1);
for (i=1:5)
   impulse(((i-1)*filter_size/5)+1)=1/i;
end
desired_signal = conv(input_signal, impulse);

% initialise variables
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
stepsize_curr = zeros(filter_size, 1);
stepsize_prev = zeros(filter_size, 1);
gradient_curr = zeros(filter_size, 1);
gradient_prev = zeros(filter_size,1);

% perform VSLMS algorithm for set number of iterations
for i=1:iterations
   i
   % get input sample
   input_vector(1)=input_signal(i);
   % calculate filet output
   filter_output(i)=dot(filter_current, input_vector);
   % calculate error signal
   error= desired_signal(i)-filter_output(i)
   gradient_curr=error*input_vector;
   %perform step size update
   stepsize_curr = stepsize_prev + gradient_curr.*gradient_prev;
   % Maintain step_size vector between upper and lower limits.
   for j=1:filter_size
      if stepsize_curr(j) > ss_upper;
```

```
            stepsize_curr(j) = ss_upper;
        end
        if stepsize_curr(j) < ss_lower
            stepsize_curr(j) = ss_lower;
        end
    end

    % update filter tap weight vector
    filter_current = filter_current + 2*diag(stepsize_curr)*gradient_curr;

    % shift values in input vector along ready for next iteration
    for j=filter_size:-1:2
        input_vector(j)=input_vector(j-1);
    end

    %store values of error and cost function.
    error_signal(i)=error;
    cost(i)=error*error;

    %update variable vectors for next iteration
    gradient_prev=gradient_curr;
    stepsize_curr=stepsize_prev;
end

% Find moving average of error squared.
for i=1:iterations-100
    mse(i)=mean(cost(i:i+100));
    i
end

%find moving average of db attenuation (averaged to smooth output).
for i=1:iterations-2500
    db(i)=-
20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    i
end

%find total average db attenuation
db_avg=mean(db);
```

A.4 VSNLMS.m (VSNLMS ALGORITHM)

```
function [input_signal, error_signal, desired_signal, filter_output, impulse, filter_current,
mse, db, db_avg]=VSNLMS(filter_size, ss_lower, input_file, iterations)
% Function to perform the VSNLMS algorithm on an input file.
% Inputs:   Filter order, step size, stepsize upper bound, stepsize lower bound, input wav
file, number of iterations.
% Outputs:  Input signal, error estimation signal (echo cancelled), desired signal (echoed
signal), adaptive filter output, real impulse response
%          Estimation of impulse response, mean sqaure error, attenuation (dB), average
attenuation.

% read in input wav file.
input_signal = wavread(input_file);

% generate impulse response and convolve to get echoed signal
impulse=zeros(filter_size,1);
for (i=1:5)
   impulse(((i-1)*filter_size/5)+1)=1/i;
end
desired_signal = conv(input_signal, impulse);

% initialise variables
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
stepsize_curr = zeros(filter_size, 1);
stepsize_prev = zeros(filter_size, 1);
gradient_curr = zeros(filter_size, 1);
gradient_prev = zeros(filter_size,1);

% perform VSNLMS algorithm for set number of iterations
for i=1:iterations
   i
   % get input sample
   input_vector(1)=input_signal(i);
   filter_output(i)=dot(filter_current, input_vector);
   error= desired_signal(i)-filter_output(i)
   gradient_curr=error*input_vector;
   % calcualte variable upper limit
   ss_upper = 1/(2*(dot(input_vector,input_vector)));
   % update stepsize vector
   stepsize_curr = stepsize_prev + gradient_curr.*gradient_prev;

   % Maintain step_size vector between upper and lower limits.
   for j=1:filter_size
      if stepsize_curr(j) > ss_upper;
```

```
            stepsize_curr(j) = ss_upper;
        end
        if stepsize_curr(j) < ss_lower
            stepsize_curr(j) = ss_lower;
        end
    end
    % update filter tap weights
    filter_current = filter_current + 2*diag(stepsize_curr)*gradient_curr;

    % shift values in input vector along ready for next iteration
    for j=filter_size:-1:2
        input_vector(j)=input_vector(j-1);
    end

    %store values of error and cost function.
    error_signal(i)=error;
    cost(i)=error*error;

    %update variable vectors for next iteration
    gradient_prev=gradient_curr;
    stepsize_curr=stepsize_prev;
end

% Find moving average of error squared.
for i=1:iterations-100
    mse(i)=mean(cost(i:i+100));
    i
end

%find moving average of db attenuation (averaged to smooth output).
for i=1:iterations-2500
    db(i)=-
20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    i
end

%find total average db attenuation
db_avg=mean(db);
```

A.5 RLS.m (RLS ALGORITHM)

```
function [input_signal, error_signal, desired_signal, filter_output, impulse, filter_prev,
cost, mse, db,db_avg]=RLS(filter_size, lambda, input_file, iterations)
% Function to perform the RLS algorithm on an input file.
% Inputs:   Filter order, lambda (forgetting factor), input wav file, number of iterations.
% Outputs:  Input signal, error estimation signal (echo cancelled), desired signal (echoed
signal), adaptive filter output, real impulse response
%           Estimation of impulse response, cost, mean sqaure error, attenuation (dB),
average attenuation.

% Read in input file.
input_signal = wavread(input_file);

% Generate desired signal by creating impulse and convolving
impulse=zeros(filter_size,1);
for (i=1:5)
   impulse(((i-1)*filter_size/5)+1)=1/i;
end
desired_signal = conv(input_signal, impulse);

% Initialise varaibles
filter_prev = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
psi_inv_prev = eye(filter_size);
intermediate= zeros(filter_size, 1);
gain = zeros(filter_size, 1);

% Perform RLS algorithm for set number of iterations.
for i=1:iterations
   i
   input_vector(1)=input_signal(i);
   intermediate = psi_inv_prev*input_vector;
   gain = (1/(lambda+dot(input_vector, intermediate)))*intermediate;
   filter_output(i)=dot(filter_prev, input_vector);
   error= desired_signal(i)-filter_output(i)
   filter_prev = filter_prev + gain*error;
   psi_inv_prev = (1/lambda)*(psi_inv_prev - gain*((input_vector')*psi_inv_prev));
   % Shift along input vector
   for j=filter_size:-1:2
      input_vector(j)=input_vector(j-1);
   end
   error_signal(i)=error;
   cost(i)=error*error;
end
```

```
% Find moving average of error squared.
for i=1:iterations-100
   mse(i)=mean(cost(i:i+100));
   i
end

%find moving average of db attenuation (averaged to smooth output).
for i=1:iterations-2500
   db(i)=-
20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
   i
end

%find total average db attenuation
db_avg=mean(db);
```

APPENDIX B: TI CCS IMPLEMENTATION CODE

This appendix contains the source code for the files nlms_adfilt.c and NLMS.gel used in the real time implementation of the echo cancellation system. For their correct implementation these files and the support files listed below must be incorporated into a CCS project. Please refer to Chassaing 2002 for the correct way to do this.

B.1 nlms_adfilt.c

```
// nlms_adfilt.c
// Real time implementation of NLMS algorithm for echo cancellation system.
// Uses TI TMS320C6711 DSK.
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

// Defines
#define FILTER_LENGTH 2300
#define ECHO_DELAY 600
#define STEP_SIZE 5e-5
#define SCALE 5000.0
#define GAIN 15

// Declare variables
float input, echoed, af_output, error_signal;
float input_vector[FILTER_LENGTH];
float filter[FILTER_LENGTH];
float echo_buffer[ECHO_DELAY];
float step_size = STEP_SIZE; // fixed step size for standard LMS implementation
float nstep_size;
short output;
short output_type=1;
short delay_param=0;
short i, j, eb_index;


// Procedure for determining dot product of two arrays
float dotp (float a[], float b[])// based on Chassaing 2002, p. 242
{
        float suml, sumh;
        int j;
        suml=0;
        sumh=0;
        for(j=0; j<FILTER_LENGTH; j+=2)
        {
                suml += a[j] * b[j];
                sumh += a[j+1]*b[j+1];
        }
        return (suml+sumh);
}

// This interrupt is triggered at every sample instant
interrupt void c_int11()
{
        input=(float)(input_sample())/SCALE;          // newest input cast to float
        input_vector[0] = input;                  // put sample

        //Firsly calculate echoed output. Using delay-buffer method.
        echoed = input+0.4*echo_buffer[eb_index];
        echo_buffer[eb_index]=echoed;
        eb_index++;
        if (eb_index >= ECHO_DELAY-delay_param)
                eb_index=0;

        //calculate output of adaptive filter
        af_output=dotp(filter, input_vector);

        // calculate error value
        error_signal = echoed-af_output;

        // calculate variable step size
        nstep_size=1/dotp(input_vector, input_vector);

        //update tap weights
        for (i=FILTER_LENGTH-1; i>0; i--)
        {
                filter[i] = filter[i] + 2*nstep_size*error_signal*input_vector[i]; //calculate
taps
                input_vector[i]=input_vector[i-1]; //shift vector
        }

        // Switch for controlling the output type using lms.gel
        if (output_type==1)
        output=(short)(input*SCALE*GAIN);

        if (output_type==2)
        output=(short)(echoed*SCALE*GAIN);

        if (output_type==3)
        output=(short)(af_output*SCALE*GAIN);
        if (output_type==4)
        output=(short)(error_signal*SCALE*GAIN);

        //output the sample
        output_sample(output);
}


// This is main procedure executed on start up of program
main()
{
         // Initialise variables
        error_signal=0.0;
        echoed=0.0;
        af_output=0.0;
        eb_index=0;
        nstep_size=0;
        for (i=0; i<FILTER_LENGTH-1; i++) // initialise filter, input vector
        {
                filter[i]=0.0;
                input_vector[i]=0.0;
        }
        for (i=0; i<ECHO_DELAY; i++) // init echo buffer
        {
                echo_buffer[i]=0.0;
        }
        comm_intr();                            //init DSK, codec, McBSP
        while(1);                               //infinite loop
}
```

B.2 NLMS.gel

```
// GEL file for nlms control

menuitem "LMS"

// create slider for output selection
slider LMS_output(1,4,1,1,lms_output_type)
{
        output_type=lms_output_type;
}

// create slider to vary echo length
slider echo_length(0, 300, 50, 50, delays)
{
	delay_param=delays;
}
```

B.3 CCS support files.

The following files were utilised in the implementation of the echo cancellation system. They can be found on either the CD accompanying Chassaing 2002, or with the Texas Instruments DSK 2002.

- c6xdskinit.c
- c6x.h
- c6xdsk.h
- c6xdskinit.h
- c6xinterrupts.h
- vectors_11.asm
- rts6701.lib
- C6xdsk.cmd